1 Introduction

Turn to the dark side of the force! This assignment will have you exploit a buffer overflow in one program to make it execute another program.

This assignment is a *group* assignment. Each group should have no more than *two* students. Your partner should be different from previous MP partners.

2 Creating Your Exploit

You are given the code to a program name `count`. This program prints the number of bytes that it reads from standard input. It also has a buffer overflow. Your task is to exploit this buffer overflow to execute the `/usr/bin/echo` program to print the string "I've been hacked!"

For example, a succesful run appears as follows:

```
%  echo abc | ./count
Read 4 bytes
```

An execution that leads to exploitation appears as follows:

```
%  cat badInput | ./count
I've been hacked!
```

You are to submit an input file named `badInput` that triggers the exploit.

Your attack will be a code reuse attack. You should look for code within the application and see if there is any code that you can use for your attack (hint: there is).

3 Source Code Access

The instructor will provide access to the source code upon request. However, access to the source code will result in a loss of points.

4 README

You should submit a README file with your submission. Your README file should provide your name as well as the name of your partner. Your README file should also answer the following questions:

1. What did each person contribute to the solution submitted?

2. In what function is the buffer overflow?

3. Explain how your input exploits the buffer overflow to run the `/usr/bin/echo` program. To what program location do you jump to make the exploit work?

4. Suppose that the location of the code to which you jump in your attack is chosen randomly when the program starts. How long would it take you to mount a brute force attack? To answer this question, you should measure the amount of time each attempt to attack the program takes and calculate the expected amount of time to exploit the program.

5 Helpful Resources

The `gdb` debugger is installed on the lab machines and will be useful in analyzing the dynamic behavior of the `count` program.

The `objdump` tool can be used to disassemble a program so that you can examine its assembly code instructions. The `nm` program can print the symbol table of an executable.

It may be helpful to know the x86-64 calling conventions.

The Intel Architecture Reference Manual will be useful for students not familiar with the x86-64 instruction set. As a quick reminder, register `%rsp` is the stack pointer, `%rip` is the program counter, and registers `%rax`, `%rbx`, `%rcx`, `%rdx`, `%rsi`, and `%rdi` are general purpose registers.

6 Administrative Policies

6.1 Submission

You are to create a directory named `mp3` and within this directory place the following items:

1. The file named `badInput` which contains the input which triggers the exploit.

2. The README file listing all partners in the project and their contributions and the answers to the question in Section 4.

You should submit your files as a single GNU zipped tar archive named mp3.tar.gz that, when unzipped and extracted, creates a directory named mp3 that contains the above files.

If you are unfamiliar with the tar and gzip/gunzip programs, please consult their man pages. You will submit your GNU zippped tar archive via Blackboard.

6.2 Grading

- *40 points:* your input file `badInput` successfully exploits the `count` program
- *30 points:* your README file contains correct answers to the questions in Section 4
- *10 points:* your solution performs a ROP-style attack instead of a simpler code injection or ret2libc attack
- *10 points:* your solution does not require knowledge of the program's source code
- *10 points:* you turn in all requested files in the requested format