Lab Exercise 2

Due: May 7, 2018 **Extended; due date is now May 9** **Points:** 100

For this laboratory exercise, you are to work in teams of 2–3 people. When you turn in your results, also upload a README file giving the names and UC Davis email addresses of all members of your group. Only one person needs to upload the answers; the other members should upload a file named README identifying the other members of their group, as above, and note who uploaded the answers.

This laboratory exercise has you implement two types of buffer overflows. The first is a simple overflow that causes a parameterless routine to execute. The second is a return-to-libc (or arc) attack.

You will need a virtual machine available via the web at http://nob.cs.ucdavis.edu/ecs153/lab2. The password is "ubuntu" (without the quotes, of course). When you start the virtual machine, you will find two programs, *bad.c* and *realbad.c* and two executables, *bad* and *realbad*, in your directory. Note the last executable is setuid-to-root.

A word of warning. Ubuntu Linux comes with a defense called "address space layout randomization" (ASLR). This must be off for you to complete this exercise successfully. It is turned off in the virtual machine you download, but it gets turned on automatically whenever you restart the machine. So, after you restart, log in and type the following command to turn it off:

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

Buffer Overflow I

In your home directory is a program *bad.c* (also see below). This program contains a buffer overflow vulnerability; see the call to *gets*(3) at line 13. Your job is to exploit the overflow by providing input to the running process that will cause the program to invoke the function `trap` (which, you may notice, is not called anywhere else). You will know you've succeeded when you run the program, give it your input, and it prints "Gotcha!"

The following questions will help guide you. Please turn in your answers to them, a hex dump of the input you use to call trap, and a typescript or screen shot of you running the program *bad*, giving it your input, and showing its output.

1. What is the address of the function `trap`? How did you determine this?
2. What is the address on the stack that your input must overwrite (please give both the address of the memory location(s), and their contents)? How did you locate this address?
3. What is the address of `buf`?
4. The sled is the input you give to alter the return address stored on the stack. What is the minimum length your sled must be?

Buffer Overflow II

Now you will extend the overflow attack. In your home directory is another program *realbad.c* (also see below). As before, this program contains a buffer overflow vulnerability. Your job is to exploit the overflow by providing input to the running process that will cause the program to invoke the function `runcom` and cause the *system*(3) function to be executed with a command embedded in the input you have given. You must pass in a parameter that is a Linux command, which the program will then execute. (I recommend the command *id*(1).)

Please turn in the following:

1. A hex dump of the input you use. Please also show where the parameter to `runcom` is in your input.
2. A screenshot of the program's output for that input.

Recovery

If you accidentally delete or change the executables, you can recreate them yourself. First, compile the source using *gcc* ***with the option `-fno-stack-protector`***; if you omit the flag, the attempt to overflow the stack will be blocked and so the lab will not work. That's it for *bad*. For *realbad*, once you compile it, do the following:

```
sudo chown root realbad
sudo chmod 4755 realbad
```

and enter the password given above when asked.

The Programs

bad.c

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

void trap(void)
{
        printf("Gotcha!\n");
        exit(0);
}

int getstr(void)
{
        char buf[12];
        gets(buf);
        return(1);
}

int main(void)
{
        getstr();
        printf("Overflow failed\n");
        return(1);
}
```

realbad.c

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

void runcom(char *cmd)
{
        system(cmd);
        exit(0);
}

int getstr(void)
{
        char buf[12];
        gets(buf);
        return(1);
}

int main(void)
{
        getstr();
        runcom("echo Overflow failed");
        return(1);
}
```