

Lab Exercise 2

Due: May 5, 2021

Points: 100

This laboratory exercise has you implement a simple overflow that causes a parameterless routine to execute. The extra credit passes an argument to the routine to be executed.

Setup

You will be doing this on a virtual machine. The hypervisor is VirtualBox, available at <https://www.virtualbox.org/>

Also, get the VirtualBox Extension Pack. Install both VirtualBox and the extension pack.

You will also need the virtual machine for this exercise. It is available via the web at <http://nob.cs.ucdavis.edu/ecs153/lab2>

Download it. It's big — about 2.8 GB — so it may take some time.

To run it, start VirtualBox, and go to Import Appliance (it's under the File menu item). Select the file that you downloaded, *bufov-1.ova*. Then follow the directions. Your imported machine will show up in the list of virtual machines.

Then in the settings, go to System, and change RAM to be 4096 MB.

When you start the machine, you will need to enter the password for the account “Ubuntu”. 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
```

Regular Buffer Overflow

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!” This is called a “return-to-libc” or “arc” attack.

The following questions will help guide you. Please turn in your answers to them and how you got them (specifically, the *gdb(1)* input and output and any computations), 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?

Extra Credit: Buffer Overflow with Program Execution

Now 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.
3. How you got these answers (specifically, the *gdb(1)* input and output and any computations)

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 this 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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void trap(void)
5 {
6     printf("Gotcha!\n");
7     exit(0);
8 }
9
10 int getstr(void)
11 {
12     char buf[12];
13     gets(buf);
14     return(1);
15 }
16
17 int main(void)
18 {
19     getstr();
20     printf("Overflow failed\n");
21     return(1);
22 }
```

realbad.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void runcom(char *cmd)
5 {
6     system(cmd);
7     exit(0);
8 }
9
10 int getstr(void)
11 {
12     char buf[12];
13     gets(buf);
14     return(1);
15 }
```

```
15 }
16
17 int main(void)
18 {
19     getstr();
20     runcom("echo_Overflow_failed");
21     return(1);
22 }
```