Homework Assignment 3

Cybersecurity

COSC 3371 / 2022 Spring

In this homework assignment, you will find and exploit software vulnerabilities in C programs. Since some software vulnerabilities are platform dependent, the vulnerable programs will be running on a virtual machine that provides the particular platform which we use in this assignment. As your first task, please

- download and install Oracle VM VirtualBox (https://www.virtualbox.org/ or using your package manager on Linux),
- download the virtual machine from Blackboard and import it into VirtualBox,
- start the virtual machine.

After a couple of seconds, you should see a text-based Ubuntu Linux login screen. Enter `student` as the username and `password` as the password.

You can also log in to the virtual machine using SSH.

- First, make sure that the networking settings of the virtual machine allow connection from your host: right-click on the virtual machine in the main screen of VirtualBox to open the virtual-machine settings window (note that these settings are different from the general preferences of VirtualBox), select the "Network" tab, make sure that "Enable Network Adapter" is checked, and select an adapter that enables connection from your host (e.g., "Bridged Adapter").
- Once you have logged in to the virtual machine through the VirtualBox window, you can determine the IP address of the virtual machine by running the command `ifconfig` and reading the address after `inet addr:` (make sure that you are looking at the real interface, not the loopback `lo` interface).
- Then, you can log in using SSH from your host by running the command `ssh student@<IP address>` (or similarly on a graphical interface if you are using a graphical SSH client).

In your home directory, you will find (type `ls -lt` into the terminal to list the contents of the directory) five programs: `problem1`, …, `problem5`. You can run program `problemX` by typing `./problemX` into the terminal. Attached on Blackboard, you will find the corresponding C source files: `problem1.c`, …, `problem5.c`. Please note that you will not have to compile these source files; however, you will need them for finding the vulnerabilities.

To solve each problem, you will have to read the contents of a secret file:

```
 /home/secret1.txt
 …
 /home/secret5.txt
```

You will be able to read the contents of each file by finding and exploiting a vulnerability in the corresponding program. Note that each file contains a simple, one-sentence message.

As your solution, you should submit a single text file on Blackboard, which contains for each problem (1) **program input** that exploits the vulnerability, (2) **brief explanation of why the exploit works**, and (3) contents of the secret file.

Problem 1 (2 points): Buffer Overflow

Find and exploit the buffer-overflow vulnerability by studying the source code `problem1.c`! Note that you will not have to inject executable code or change any return addresses.

Since the layout of local variables and buffers is determined by the compiler, it may not be easy to tell how the stack will look like based on the source code. Fortunately, you can observe the stack layout by debugging the program. Type in the following command to launch the GDB debugger for program `problem1`:

```
gdb problem1
```

At this point, the debugger has loaded the program into memory but has not started it yet. Before we run the program, it will be a good idea to set a breakpoint for function `problem1()`:

```
(gdb) b problem1
```

Once the breakpoint is set, we can run the program:

```
(gdb) r
```

Next, we see that execution has stopped at the beginning of function `problem1()`. At this point, we can read the values of local variables, e.g.,

```
(gdb) p command
```

Note that since this variable has not been used yet, its value is garbage left over from previous use of the stack. More importantly, we can also read the addresses of local variables, e.g.,

```
(gdb) p &command
```

The output is the memory address where the variable is stored (i.e., address of the beginning of the buffer) in hexadecimal format. Let's read the addresses of `command`, `dataset`, and `buffer`, and calculate how many bytes apart they are (just subtract the addresses from each other)! This should give you an idea of which buffers can overflow into which other buffers and how many bytes you need for the overflow to reach them. Note that the exact addresses may change between executions, but the layout of the stack within one function typically does not change. Also note that while you are debugging a program, `setuid` is ignored; so once you figure out the exploit, run the program without debugging to read `secret1.txt`.

Problem 2 (2 points): Integer Overflow

Find and exploit the integer overflow vulnerability!

Problem 3 (2 point): Integer and Buffer Overflows

Find an exploit the vulnerabilities!

Problem 4 (2 points): Format String Reading

Find and exploit the format string vulnerability! Hints:

- use GDB to figure out the stack layout (e.g., how far variables `secret` and `length` are from each other);
- use lots of %d placeholders to determine how deep in the stack some variables are (e.g., how deep variable `length` is);
- secret message is on the heap, not the stack;
- you will need only memory reading to exploit this vulnerability.

Extra Credit:

Problem 5 (2 extra point for the course): Format String Writing

Find and exploit the format string vulnerability by changing your `authorization_level`! Hints:

- use GDB to figure out the stack layout;
- `authorization_level` is on the heap, but one of the local variables points there (technically, it points to the `struct User`, but luckily `authorization_level` is the first field in the structure);
- use lots of %d placeholders to find how deep some variables are (e.g., `minimum_level`).