CSE 127: Computer Security (Spring 2019) **Due May 3, 10:00 P.M. PDT**

Assignment 2

100 pts

The goal of this assignment is to gain hands-on experience with the effects of buffer overflows and other memory-safety bugs. You will be provided a skeleton for implementing these exploits in C. Your solution is due on May 3, 10:00 P.M. PDT. You may work with *one* other person in your class section on the assignment; if so you should only submit one solution for the two of you. You and your partner must *not* discuss your solution with other students until seven days after the assignment deadline. You may consult any online references you wish. If you use any code in your answer that you or your partner did not write yourselves, you *must* document that fact. Failure to do so will be considered a violation of the academic integrity policy.

1 Getting Started

To complete this assignment **you will be using the same VirtualBox VM from assignment 1**. The set of files (including the turn-in script) can be found on the class website under assignments.

1.1 Reminder: VM Image

In order to match the environment in which your submission will be graded, all work for this assignment must be done on the VirtualBox VM we have provided named `cfbox`.

The VM is configured with two users: `student`, with password "`hacktheplanet`", and `root` with password "`hackallthethings`". The VM is configured with SSH on port 2222. Please note that SSH is disabled for `root`, so you can only SSH in as the student user. You can still log in as `root` using `su` or by logging into the VM directly.

To SSH into the VM:

```
ssh -p 2222 student@127.0.0.1
```

To copy files from your computer to the VM:

```
scp -P 2222 -r /path/to/files/ student@127.0.0.1:/home/student
```

To copy files from the VM to your computer:

```
scp -P 2222 student@127.0.0.1:/path/to/files/ /destination/path
```

1.2 Assignment Files

Starter files are provided in an archive on the class webpage. It contains exploit starter code (in the `sploits` directory) for each of the 4 vulnerable programs (in the `targets` directory). Additionally included with the exploits is `shellcode.h`, which gives Aleph One's shellcode. The `sploits` directory contains a Makefile to build the expoits. Each sploit contains the code to call the appropriate target with a string command line argument. The `targets` directory contains a Makefile to generate targets specific to your PID, and a folder called `base` that you should not modify as these are what are used to generate your targets.

NOTE: you should **only** modify sploit[1-4].c, PID, and writeup.txt. Do **not** modify any of the targets or the Makefile.

2 Assignment Instructions

You will be writing an exploit for each of the 4 vulnerable programs provided in the assignment. You wil be writing the exploits in `sploit1.c` through `sploit4.c` Each exploit, when run in the VM with its target installed setuid-root in `/tmp`, should yield a root shell (`/bin/sh`). You can verify this by typing `whoami` in the root shell, to which you should see the response `root`. You must use Aleph One's shellcode in `shellcode.h`, as this will be used in the grading scripts. Additionally, for each exploit, provide a brief description of how it works in the `writeup.txt` file.

3 Exploit Construction

To complete the assignment, you will first need to generate targets specific to your PID, then use GDB to find vulnerabilities in the targets, then build your exploits.

3.1 Generating the Targets

Targets are customized based on your PID. The targets are set with setuid-root so that the shell we spawn is a root shell. To generate your targets:

1. fill in the `PID` file as specified in Section 4.
2. run `make generate` in the `targets` directory to create the 4 target source files specific to you: `target1.c` through `target4.c`
3. run `make` to build the target binaries: `target1` through `target4`
4. run `su` to launch a root shell
5. run `make install` to copy the binaries into the `/tmp` directory
6. run `make setuid` to make the binaries as setuid-root
7. `exit` the root shell to return to the user shell

3.2 Preserving Targets

Note that the targets are installed into the /tmp directory, which is emptied when the VM is shut down. To prevent having to reinstall the targets, choose 'Save the machine state' when exiting VirtualBox.

3.3 Using GDB

To run an exploit in GDB, run e.g., `gdb -e sploit1 -s /tmp/target1 -s sploit1` to execute `sploit1` and use the symbol file `target1`.

Note that there are **two executables** (sploit1 and target1) when we run these programs. For this, use the following workflow to get started in GDB:

1. `b main` — break at `main` in sploit1.c
2. `r` — run the executable sploit1
3. `b main` — break at `main` in **target1.c**
4. `c` — continue until `main` in `target1.c`

Refer back to the GDB commands in the Assignment 1 writeup. Some other commands that may be helpful:

- `p var` — prints the value of variable `var`
- `p buf` — (if buf is an array) prints the contents of `buf`

- `p buf` — prints the starting address of `buf`
- `i r` — view registers
- `i frame` — view info about the current stack frame
- i stack — view high level info about the
- `help <command>` — get help inside GDB about a particular command

If you wish, you can instrument the target code with arbitrary assembly using the "`__asm__()`" pseudo-function, to help with debugging. Be sure, however, that your final exploits work against the unmodified targets, since we will use these in grading.

3.4 Exploit Notes

For this assignment you should read and have a solid understanding of Aleph One's "Smashing the Stack for Fun and Profit".

Aleph One gives code that calculates addresses on the target's stack based on addresses on the exploit's stack. However, relative addresses on the exploit's stack can change based on how the exploit is executed (working directory, arguments, environment, etc.); in my testing, I do not guarantee to execute your exploits exactly the same way bash does. **You must therefore hard-code target stack locations in your exploits**. You should *not* use a function such as `get_sp()` in the exploits you hand in.

We'll run the targets the same way as you – with execve. We just don't guarantee we'll run your sploits the same way as you (i.e. in the same directory, etc.). To ensure your solution is okay for our testing, avoid calculating addresses on the targets stack based on addresses on the exploits stack. If you hard-code the target's stack locations into your exploit you should be fine. Your exploit programs should not take any command-line arguments.

4 Submission Script

To submit your solution, use the turn-in script `hw2-turnin.sh` provided with the starter files. It will archive your submission and submit it to our server to be graded. You may submit multiple times, but only your latest submission will be graded. Male sure the `PID` file is filled out with your PID on the first line. If you're working with a partner, include both PIDs on the first line separated by a space. Use the same password emailed to you before assignment 1.