CSE 127: Computer Security (Fall 2017) **Due October 24, 10:00 P.M. PDT**

Assignment 4

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 October 24, 10:00 P.M. PDT. You may work with *one* other person in the class on this assignment, however, each student must submit his/her own solution. See Section 8 for additional information on submitting your solution. 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 Problems

All work in this project must be done on the Virtual Machine provided on the course website; see below for information about this environment. You will be writing an exploit for each of five vulnerable programs provided in the assignment. For grading, these programs will be installed in the `/tmp` directory (e.g. `/tmp/target1`) and have setuid-root. If you successfully exploit each binary, you should get a root shell, which you can verify by typing `whoami` to which you should see the response `root`. The file `shellcode.h` contains Aleph One's shellcode. You must use this shellcode, as this will be used in the grading scripts. You are to write 5 exploits, one for each target. Each exploit, when run in the virtual machine with its target installed setuid-root in `/tmp`, should yield a root shell (`/bin/sh`). I recommend you read and have a solid understanding of Aleph One's "Smashing the Stack for Fun and Profit" for this assignment.

2 Generating the Targets

This assignment contains the source code for five vulnerable target programs. The first step is to fill in the `SID` file with your PID and name (last, first), for example:

```
A01234567 Smith, Alice
```

Once you have created your SID file, you can then run `make generate` on the command line to create the target files specific to you. This should generate five target files, `target1.c` through `target5.c`. In order to build these target files, you can run `make` on the command line. To copy the binaries into the `/tmp` directory, simply run `make install`. To make the binaries as setuid-root, use `su` to launch a root shell, and then `make install` and `make setuid`. Don't forget to `exit` your root shell when you're done, returning you to the user shell.

3 GDB Tips

You will find GDB helpful in making sure your exploit works correctly. The `x` command is useful for examining memory (note the different ways you can print the contents such as `/a` `/i` after `x`). The `info register` command is helpful in printing out the contents of registers such as `ebp` and `esp`. The `disassemble` command is useful for disassembling a piece of code. In addition, GDB has a number of smart shortcuts it accepts. For example, `disas` is `disassemble`, `i r` is `info register`, as well as many others. Learning these can drastically speed up your debugging.

A useful way to run GDB is to use the `-e` and `-s` command line flags; for example, the command `gdb -e sploit3 -s /tmp/target3` in the VM tells GDB to execute `sploit3` and use the symbol file `target3`. These flags let you trace the execution of the target3 after the sploit's memory image has been replaced with the target's through the `execve` system call. When running `gdb` using these command line flags, you may want to use the following procedures for debugging your exploit:

1. Set a breakpoint at a function or instruction, for example with the command `b foo`.
2. Run the program with the `r` command. `gdb` will execute the program until it reaches the breakpoint, then return control to you.
3. Set any breakpoints you want in the target, examine memory, etc.
4. Continue execution with the command `c`, or step using `si` or `s`.

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.

4 Warnings

Aleph One gives code that calculates addresses on the target's stack based on addresses on the exploit's stack. 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.

In grading, the exploits may be run with a different environment and different working directory. Your exploits must work in these cases also. Your exploit programs should not take any command-line arguments.

5 Assignment Starting Files

You will be provided several starting files for your assignment in the archive `hw4skel.tgz` available from the class Web page. The archive contains two directories, `sploits` and `targets`. The `sploits` directory contains skeleton code that will help you get started writing your exploits, along with a `Makefile` for building them. Additionally included is `shellcode.h`, which gives Aleph One's shellcode. The `targets` directory contains a Makefile to generate targets specific to your SID, an empty SID file, and a folder called `base` that you should not modify as these are what are used to generate your targets.

6 VM Image

We have created a VirtualBox VM image configured for this assignment. You can download the VM image from:

https://drive.google.com/open?id=0BwpBL7N7eePFVUUwaWZFSzZlNzA

You should also download the signature for the compressed VM image (`hw4vm.zip.sig`) from the class Web page and verify that it matches the file you downloaded. To do this, with `hw4vm.zip.sig` in the same directory as `hw4vm.zip`, run:

```
gpg --verify hw4vm.zip.sig
```

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 the root user, 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.

```
Foster, Ian
A00000000
Assignment 4
Worked with Maskiewicz, Jacob

For sploit1.c, I calculated ...
```

Figure 1: Example solution text file format.

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
```

7 Solution Format

Your solution to this assignment consists of the following seven files: `{PID}-hw4.txt`, `SID`, `sploit1.c`, `sploit2.c`, `sploit3.c`, `sploit4.c`, and `sploit5.c`. The files `sploit1.c` through `sploit5.c` must contain your exploit code based on the provided skeleton. The `SID` must have format described in Section 2; *do not change the `SID` file after generating the targets!* The `{PID}-hw4.txt` file must have the following format:

1. The first line of the file must be your name, last name first, with a comma between last name and first name.
2. The second line of the file must be your student id number.
3. The third line must be "`Assignment 4`".
4. If you worked with another student, the fourth line must be "`Worked with`" followed by a space and the name of your partner, last name first, with a comma between last name and first name. If you worked alone, the fourth line must be "`Worked alone`".
5. The fifth line must be blank.
6. The sixth and subsequent lines may contain free-form text. Use this text to briefly explain how you solved each problem.

Figure 1 shows an example of this format.

8 Submitting Your Solution

Your solution must by submitted via email to `cs127f1@ieng6.ucsd.edu` by October 24, 2017, 10:00 P.M. PDT. Your submission must be a `gzip`-compressed `tar` archive, signed with your PGP key and encrypted to the `cs127f1@ieng6.ucsd.edu` PGP key, which is provided on the CSE 127 Web page and has key fingerprint:

```
E1BF 1E04 1104 28DA 4F89 6543 B033 B3DC 10D3 7DBD.
```

You must send a plain email with the encrypted and signed archive file as an attachment. The email must have the subject "Homework 4 Submission" and the attachment must be named "{PID}-hw4.tgz". To create a gzip-compressed tar archive, copy the files you wish to submit to single directory, change into that directory, and issue the command:

```
tar -zcvf /path/to/archive/{PID}-hw4.tgz {PID}-hw4.txt SID sploit?.c
```

This will create an archive in the directory /path/to/archive/ containing all the exploit files in the current working directory. To sign your submission with GPG:

```
gpg --encrypt --sign --armor -r cs127f1@ieng6.ucsd.edu {PID}-hw4.tgz
```

You will need to have imported the cs127f1@ieng6.ucsd.edu public key into your GPG keyring first.

9 Grading

You may work with one other student in the class to complete this assignment, however, *each student must submit his/her own solution*, with their student ID and name in the SID file. You may *not* discuss your solution with other students until seven days after the assignment deadline. Your solution will be graded on whether or not you exploit each target successfully, as described in Section 1. We will use the same VM we provided you, so make sure that your solution works on the original image of the VM. You may consult any online references you wish. If you use any code you find online, you must document it in the {PID}-hw4.txt file submitted with your solution.