University of Chicago
CMSC 23200/33250: Introduction to Computer Security, Winter 2020
Course Staff: David Cash, Alex Hoover, Rohan Kumar, Blase Ur, Valerie Zhao

Assignment 8: Buffer Overflows

Due at 11:59pm Wednesday, March 11

Introduction

In this assignment you will implement buffer-overflow attacks to hijack victim programs, starting with the example program David attacked in class.

Rules (mostly unchanged from before)

Collaboration policy. Please respect the following collaboration policy: You may discuss problems with up to 3 other students in the class, *but you must write up your own responses and your own code. You should never see your collaborators' writing, code, flags, or the output of their code.* At the beginning of your submission write-up, you must indicate the names of your (1, 2, or 3) collaborators, if any. You may switch groups between assignments but not within the same assignment.

Sources. Cite any sources you use. You may Google liberally to learn about Linux commands, assembly, gdb, etc. You **must**, however, note at the end of your writeup for each task the sources you referenced. Please note any you found particularly helpful since we might include them in future versions of this writeup.

Campuswire. We encourage you to post questions on Campuswire, but do not include any of your code in the Campuswire posts. If you have a question that you believe will reveal secrets you have discovered while working on the assignment, post privately to just the instructors. If you have a question that you believe will be of general interest or clarifies the assignment, please post publicly. If you are uncertain, post privately; we will make public posts that we believe are of general interest.

Grading. Responses will be graded for correctness and clarity.

What and How to Submit

Submit a tar file `cnetid.tar` to Canvas containing:

1. A file `<cnetid>-writeup.pdf/txt` with information requested from the problems.
2. The six files `sploit0.c`, ..., `sploit5.c`.

Tech Set-Up

0.1 Install VirtualBox and the Class VM

For this assignment you will run a virtual machine locally; doing so allows for a controlled environment (e.g. 32 bit, ASLR turned off, etc) and makes carrying out the attacks much easier. Here are the instructions for setting up your VM:

1. Download and install VirtualBox at `https://www.virtualbox.org/`.
2. Download the class VM at `https://people.cs.uchicago.edu/~davidcash/CS232-Assignment-8.ova`.
3. Open the `.ova` file with VirtualBox, and follow the prompts to create a new VM. You should not have to change of any of the settings.
4. Start your VM and confirm that you can log in. There are two accounts: `user` with password `user`, and `root` with password `root`.

0.2 Build the Victim Programs

In the `user` home directory there is a directory called `assignment8` that contains the files you'll use. The directory `/home/user/assignment8/targets/` contains the source code of victim programs you'll attack. To build them, run `make`, and then `sudo make install`. This will copy the executable files into `/tmp/`, which is where the attacker programs will expect to find them. Be sure to always use to the makefile for this, to ensure the needed compiler flags and file permissions are set.

0.3 Quality-of-Life Recommendations

Logging into the terminal provided by VirtualBox is workable, but I find it annoyingly primitive. Instead, I `ssh` into the VM. This way I can easily open several terminals, and get nice features like syntax highlighting, scrolling in the terminal, and cut-and-paste.

The VM is configured to have the static IP `192.168.56.232`. Accessing the VM should work out of the box, via `ssh user@192.168.56.232` on your machine's command line. If this does not work, then do the following to check that the VM's network card is properly configured:

1. Click on the VM (default name is CS232-Assignment-8).
2. Click on "Settings" (the gear icon at the top).
3. Click on "Network" at the top.
4. Check that the network adapter is enabled, and set to "Host-only Adapter" in the "Attached to:" drop-down list.

If this still does not work, try asking for help in person or on Campuswire.

Note that you can avoid typing a password constantly by installing your SSH public key in the VM under the user account. A simple way to do this from your machine is to change to the directory containing your `pub` file (usually called `id_rsa.pub`) and run `ssh-copy-id -i id_rsa.pub user@192.168.56.232`. After this, you should be able to log in without typing a password.

We can make `ssh` even easier to use by creating a file on your host machine `~/.ssh/config` with the following lines:

```
Host a8
HostName 192.168.56.232
User user
```

After doing this, you should be able to access your VM by simply typing `ssh a8`. This is handy in general; For instance I have the following in my config file to save typing:

```
Host uc
Hostname linux.cs.uchicago.edu
User davidcash
```

You can put several such host configurations in the same file.

Finally, if you want to install additional packages on the VM, that is fine, but your attacks will need to work on a clean copy of the assignment VM. To install packages, you will need to connect your VM to the Internet to do this. The easiest way to do this is to shut down the VM and change the network adapter from "Host-only Adapter" to "NAT" or "Bridged Adapter", and then reboot. You'll need to undo this change to connect to the VM from your host machine.

0.4 Resources

You may find the following classic buffer-overflow readings useful:

1. Smashing The Stack For Fun And Profit `http://phrack.org/issues/49/14.html#article`
2. Basic Integer Overflows `http://phrack.org/issues/60/10.html#article`

The textbook also has good coverage of integer overflow bugs.

Problem 0-5

In `/home/user/assignment8/sploits/` you will find skeleton files `sploit0.c`, ..., `sploit5.c`, along with `shellcode.h`. These files can be compiled with the makefile included in that directory. When run, the skeleton code will execute their respective target from `/tmp/`. The `args` array are the command line arguments passed to the target. You shouldn't modify `env`, except perhaps for `sploit5` (and even then there are other ways to finish that problem).

You will modify files `sploit0.c`, ..., `sploit5.c` so that each, when compiled and run, executes the appropriate target and obtains a shell (since the targets are all root setuid binaries, the shell will be a root shell; you can test this with `whoami`). For your convenience, `shellcode.h` contains the classic NULL-free shellcode that I used in class.

You may find it useful, especially at the very start, to try attacking the targets directly at the command line instead of via the `sploit` files. But ultimately you must implement your attacks in the `sploit` files because grading will assume this format. (I find this easier, since working in C makes it is to feed binary garbage to the program as an argument.)

Note that while debugging your attacks, you will need to run `gdb sploitx`, and then set `gdb` to follow the child `target` process by typing `set follow-fork-mode child` into `gdb`. Note that when

you set a breakpoint in `main`, `gdb` will break at both the `main` in the sploit *and* the `main` in the target. Once you're in the target, you can set further breakpoints. (Another quality-of-life recommendation: Put the `set follow-fork-mode child` command in a file `/home/user/.gdbinit`, and it will run every time you start `gdb`. You can put other startup commands there, and also alias your favorite commands too.)

You may modify the target files as you like. But be sure to check that your final sploits work against the original files. For your convenience, the assignment includes a tar file with the original code so you can reset the target files if needed.

Problem 5. The last problem is built to have a non-executable stack. You will need to implement a return-to-libc attack for this sploit. **For full credit, your program should exit gracefully after closing the shell instead of crashing the target.** This is typically accomplished by calling `exit()` after the call that produces the shell.

What to submit. In addition to your `sploit` files, in you write-up, include for each problem a description of how your attack works. You don't need to include the binary input itself, but describe its structure and how it hijacks control of the program.

Some Hints for Working with `gdb`

The attached `gdb-cmds.txt` file contains all of the commands I needed for all of my attacks. I will hold a tutorial session soon; Look on Campuswire for details.