CS 161

Computer Security

Dawn Song

Spring 2011

Lab 1

Buffer Overflows v.01

Due Date: Friday February 3, 2012 by 11:59pm

Document Revision History

Feb 1, 2012 Submission instructions have been updated. Please read them carefully!

Jan 26, 2012 The VM image was updated! Please redownload it immediately. Find it here: `http://tinyurl.com/ucbcs161lab1-2`.

Jan 25, 2012 Lab released.

Background

It is a time of rebellion. The evil empire of Caltopia oppresses its people with relentless surveillance, and the emperor has recently unveiled his latest grim weapon: a supremely powerful botnet, called *Calnet*, that aims to pervasively observe the citizenry and squash their cherished Internet freedoms.

Yet in the enlightened city of Birkland, a flicker of hope remains. The brilliant University of Caltopia alumnus Neo, famed for not only his hacking skills but also the excellent YouTube videos he produces illustrating his techniques, has infiltrated the empire's byzantine networks and hacked his way to the very heart of the Calnet source code repository. As the emperor's dark lieutenant, Prof. Evil of Evil University, attempts to hunt him down, Neo feverishly scours the Calnet source code hunting for weaknesses. He's in luck! He realizes that Prof. Evil enlisted ill-trained CS students from Evil University in writing Calnet, and unbeknownst to the empire, the code is assuredly not memory-safe.

Alas, just as Neo begins to code up some righteous exploits to pwn Calnet's components, a barista at the coffeeshop where Neo gets his free WiFi betrays him to Prof. Evil, who brutally cancels Neo's YouTube account and swoops in with a SWAT team to make an arrest. As the thugs smash through the coffeeshop's doors, Neo gets off one final tweet for help. Such are his hacking skillz that he crams a veritable boatload of key information into his final 140 characters, exhorting the University of Birkland's virtuous computer security students to carry forth the flame of knowledge, seize control of Calnet, and let freedom ring once more throughout Caltopia ...

Overview and Getting Started

Please note: No collaboration beyond your lab partner is allowed! Max group size is 2. No exceptions unless there are an odd number of students.

To get started, read "Smashing The Stack For Fun And Profit" by AlephOne [1] and "Basic Integer Overflows" by Blexim[2]. Neo recommended that you try to absorb the high-level concepts of exploiting stack overflows rather than every single line of assembly. He also warned you that some of the example codes are outdated and may not work as-is.

In this project you will write exploits for 3 vulnerable Calnet components, each owned by a different calnet user. Each program forms one part of the nefarious botnet. Due to the extreme secrecy of the project and the exploits contained therein, each vulnerability has been made into a simple stand alone program. But, don't ignore the gravity of the situation! If you solve these three overflows, Calnet will surely fall. All you have to go by are your wits, your grit, and Neo's legacy: guidelines on how to proceed, and, most precious, a virtual machine (VM) image that contains code samples from the main Calnet components.

You are to write exploits, one per target. Each exploit, when run in the virtual machine with each target installed as setuid[1] as user calnet1, calnet2, calnet3 and when exploited, each component will yield a shell as the respective Calnet user. All of the components are found in the `/calnetsrc` directory. The Calnet components each have secret codes required to disable them. After exploiting a target, in that shell, extract the secret code by typing `cat /calnetN/code`, where N is the current target. If your exploit has worked, and you have the proper permissions, the code will be output. SAVE THIS FOR YOUR SUBMISSION.

The sploits/ directory in the assignment tarball contains skeleton source for the exploits which you are to write, along with a Makefile for building them. We also included shellcode.h, which gives Aleph One's shellcode.

The goal of this assignment is to gain hands-on experience with the effect of buffer overflows. Your goal is to write three exploit programs (sploit1, ..., sploit3). Program sploit[i] will execute `/calnetsrc/target[i]` giving it certain input that should result in a privileged shell on the virtual machine.

Although no additional libraries are needed to solve this lab, you may incorporate whatever libraries necessary as long as your exploits build and run on a clean version of the VM.

How to set up the Environment

VMware Setup

Neo placed the image at `http://tinyurl.com/ucbcs161lab1-2` and which contains the image, project targets, and stub exploits. Download the image and extract it on your local machine. You need VMware to launch the VM. VMware Player is installed on the instructional machines and is also freely available for Windows and Linux. The Mac version of VMware (VMware Fusion) is available as a free 30-day trial. To use the image, start VMware, select *Open a Virtual Machine*, and browse to where you've stored the image. If it asks whether the VM was moved or copied, select *I copied it*.

[1] setuid allows a program to run with privilege as a specific user as opposed to the privilege of the user who launched it.

Important! In order for each group to have unique solutions, there is a file "setup" in user's home directory. Before attempting the exercises, run `./setup` and enter a berkeley email address of you or your partner. This sets up the secret code files whose contents you will discover.

You will run the vulnerable programs and their exploits in the VM. The image is a barebones Linux Debian installation on a 32-bit Intel architecture. A valid login is `user`, with password `user` The relevant files are located in the home directory of the user. Further, a convenient way to use the VM is to launch it, find out its IP address via `ifconfig eth0`, and then work with it remotely via SSH. This removes the small window restriction which you may find inconvenient within the VM.

The virtual machine is configured to use NAT (Network Address Translation) for networking. From the virtual machine, you can type `ifconfig` as root to see the IP address of the virtual machine. It should be listed under the field `inet addr:` under **eth0**.

Ensure that networking is working by typing ifconfig and checking that the inet addr field of eth0 has a valid IP address. Make sure you can reach the machine by attempting to SSH into it.

Setting up the Exploit Environment

Build and install the targets. Write, build, and test your exploits with the code given. Recall, logins is `user`, with password `user`.

```
user@box:~/$ tar xzvf lab1.tgz
user@box:~$ cd targets
....read, understand....
...
...
user@box:~targets$ cd sploits
...edit, test, exploit......
user@box:~/sploits$ make
user@box:~/sploits$ ./sploit1
#cat /calnet1/code
YAAAAAY!
```

The GNU debugger

The GNU debugger `gdb` will prove useful for this project, and worth some time spent becoming comfortable with it. A basic `gdb` workflow begins with loading the executable in the debugger:

`gdb` *`executable`*

You can then start running the program with:

`$ run` *`[arguments-to-the-executable]`*

(Note, here we have changed gdb's default prompt of `(gdb)` to `$`).

In order to stop the execution at a specific line, set a breakpoint before issuing the "`run`" command. When execution halts at that line, you can then execute step-wise (commands `next` and `step`) or continue (command `continue`) until the next breakpoint or the program terminates.

```
$ break line-number or function-name
$ run [arguments-to-the-executable]
$ step        # branch into function calls
$ next        # step over function calls
$ continue    # execute until next breakpoint or program termination
```

Once execution stops, you will find it useful to look at the stack backtrace and the layout of the current stack frame:

```
$ backtrace
$ info frame 0
$ info registers
```

You can navigate between stack frames using the `up` and `down` commands. To inspect memory at a particular location, you can use the `x/`*FMT* command

```
$ x/16 $esp
$ x/32i 0xdeadbeef
$ x/64s &buf
```

where the `FMT` suffix after the slash indicates the output format. Other helpful commands are `disassemble` and `info symbol`. You can get a short description of each command via

```
$ help command
```

In addition, Neo left a concise summary of all `gdb` commands at:

`http://vividmachines.com/gdbrefcard.pdf`

You may find it very helpful to dump the memory image ("*core*") of a program that crashes. The core captures the process state at the time of the crash, providing a snapshot of the virtual address space, stack frames, etc., at that time. You can activate core dumping with the shell command:

```
% ulimit -c unlimited
```

A crashing program then leaves a file `core` in the current directory, which you can then hand to the debugger together with the executable:

```
gdb executable core
$ bt      # same as backtrace
$ up      # move up the call stack
$ i f 1   # same as "info frame 1"
$ ...
```

Lastly, here is how you step into a second program `bar` that is launched by a first program `foo`:

```
gdb -e foo -s bar            # load executable foo and symbol table of bar
$ set follow-fork-mode child # enable debugging across programs
$ b bar:f                    # breakpoint at function f in program bar
$ r                          # run foo and break at f in bar
```

Submission and Grading

Please work in pairs, but the final submissions will require each student to individually submit. Any problems with submission should be redirected to a single thread in the discussion forum. We will attempt to tackle any issues that arise.

Each partner will submit their own version of the lab. Each will submit two files for final grading: `submission.txt` and an archive of the exploits. You will upload these files to the Lab 1 section of the course website.

Creating the archive:

1. Run tar/gzip within the sploits/ directory. All files in the tarball should be in its root directory. Make sure you include *EVERYTHING* required to build your exploits in this tarball: Makefile, source files, header files, etc. Please test this untarring your source code into a fresh directory and typing `make` to ensure it works as you expect. You should ensure that your exploits work on a fresh copy of the VM! Please include the following explanations as text files within the tar file as well (explained below). **Please name the tar file firstname_lastname.tgz**.

2. For each exploit you write, for each partner, submit an additional file `explanationN.txt`, where N corresponds to the exploit being explained. For each of your submissions, in 500 words or less, explain how you arrived at your answer. This should include a detailed explanation of your exploit strategy, how you determined offsets, and any other relevant information. Submissions without explanations with receive **NO CREDIT**. Submissions without adequate explanations will receive no credit. Specifically, we are aware of ways to circumvent operating system permissions in the VM image. These files should be included in the tar file you are creating for submission.

   As a guideline, imagine you are describing how to reproduce the attack to someone with knowledge of the subject, but lacks complete expertise. In the event where students are unable to reach a solution, this file we be used to determine if we should award partial credit.

Creating `submission.txt` First, download `http://tinyurl.com/cs161lab1submitscript`, this is a simple bash script which will generate the submission file. Don't forget to `chmod +x submit.sh`, before executing it.

Example output after executing `./submit.sh`

```
Enter your name:
Steve Hanna
Enter your email address:
sch@eecs.berkeley.edu
Enter your partner's name:
Al L. Bymiself
Enter your partner's email address:
alb@eecs.berkeley.edu
Enter secret codes below found in /calnet*/code.
Enter code1:
sch@eecs.berkeley.edu:secretcodehere
Enter code2:
sch@eecs.berkeley.edu:secretcodehere
Enter code3:
sch@eecs.berkeley.edu:secretcodehere

Done writing to submission.txt
```

TEST YOUR SUBMISSION In order to avoid losing points, please ensure all examples work within the VM provided and compile correctly and without warnings after typing `make`. Graders reserve the right to deduct points on any submissions which require manual examination or intervention.

Questions and Problems

If there are any questions about this lab, please post to the student discussion group so that others may benefit from your question. Please do not email any of the instructors personally!

Hints

As the deadline approaches, selective hints will be released. Please start early, these hints will only be helpful to those who are having extreme difficulty with the lab. Most importantly, HAVE FUN. This is an opportunity to learn what it's like to hack software and divert control flow.

Acknowledgments

This document was prepared by Steve Hanna. Background story and GDB tutorial provided by Matthias Vallentin.

References

[1] Aleph One. Smashing the stack for fun and profit. http://www.phrack.org/phrack/49/P49-14, November 1996.

[2] Blexim. Basic Integer Overflows. http://www.phrack.org/issues.html?issue=60&id=10, December 2002.