CS 491/591 Security and Privacy Spring 2010 Lab 1

100 points, 10 points if your username appears in /etc/sudoers on starbuck before the deadline, 10 points for your exploit source code, 10 points for your annotated hexdump, and 70 points for your writeup.

The purpose of this lab is to understand vulnerabilities and exploits, particularly remote memory corruption, in detail---especially how different interpretations of raw data makes attacks possible. The lessons you learn should be generally applicable, and relevant not just to remote memory corruption but computer security and privacy in general.

The deadline is March 1st at 11:59pm. You should submit a *.tar.gz gzipped tarball to my gmail account, *jedcrandall@gmail.com*, that contains the following:

- A *.pdf of your writeup.
- The source code of your exploit (C or Perl recommended) in text format
- A *.pdf of your annotated hexdump

The *.tar.gz and *.pdf formats are required, I'll deduct points if you submit another file format and if I can't open it you may get a 0. For MS-Word users, I suggest creating your PDFs with a freeware program called Bullzip. You may also include your nasm file for your shellcode, or you can put the assembly code as comments in your exploit source code file.

A note for this lab and future labs: save your work often, your ssh connection to shasta may be suddenly cut off at any time if I need to put shasta into maintenance mode.

The lab assignment is to develop an exploit to be launched from apollo (192.168.33.1) to starbuck (192.168.33.44), which are VMs on shasta. You'll need to get a root shell on starbuck, create an account for yourself (using your assigned username), and add yourself as a sudoer so you can have root privileges for lab 2. We can talk about how to do these things in class, and remember you can always get help by emailing me, coming to office hours, or emailing the secpriv-chat mailing list.

For this lab in particular, developing the exploit and doing the writeup are expected to be an individual effort. It's okay to get help from myself and others, but the process should be carried out by yourself and you shouldn't copy the exploits of others wholesale, only in snippets as needed. You can copy somebody else's assembly code as an input file to nasm, but I want you to go through the process of assembling it and cutting and pasting the machine code into your exploit source code yourself.

Once I have your ethics form you should have an account on shasta, you can ssh to shasta.cs.unm.edu

and then from there ssh to apollo (just type "ssh apollo" on shasta).

All of your exploit development can and should be done on apollo. You can set up your own VM to attack on your own machine if you wish, but note that you take personal responsibility for containment in this case. The exploit is fairly innocuous because nobody runs the capitalization service in the real world, but if, *e.g.*, you set off NIDS systems at your work because your shellcode is accidentally sent on a physical interface and matches some NIDS signature, that's your problem and your responsibility to explain the alert that is raised to your boss, not mine.

The first thing you should do when you log into apollo is test the capitalization service to see if it is listening on the socket and capitalizes letters, type this command on apollo where NNNN is 8080 plus your student number:

```
nc starbuck NNNN
```

You'll need to develop your own shellcode for three reasons. One is that mine has a bug which prevents it from working this year even if you get the order and magic numbers right, due to a change in the environment. Two is that my shellcode last year was messy and not very efficient. Three is that I strongly suggest using the dup2 system call to redirect stderr to stdout, otherwise the root shell you get won't allow you to see errors. If you crash the service on your port it should reappear within a couple of minutes. Even if you put it in an infinite loop, I have a new perl script that should kill it if the CPU utilization is too high for too long. You may use the unassigned student numbers 10 through 19 if you need to, but keep in mind that other students may be using those numbers.

I'll give you some files you need in a separate tar ball, start by reading notes.txt. You can also look at last year's web page and assignment (same username and password for protected files as this year) for hints.

Once your exploit is working and your username appears in the /etc/sudoers file on starbuck (double check this after you edit the file, since there could be race conditions if other students try to edit the file at the same time as you), you should do a writeup and create an annotated hexdump of your exploit's output that is piped over the network socket (*e.g.*, "`./lab2exploit > myexploit.out && hexdump myexploit.out`").

For the writeup, you should answer the following questions:

1. (15 points) In what ways were each of the two magical numbers interpreted at various stages of the exploit?

2. (15 points) In what ways was your shellcode interpreted at various stages of the exploit?

3. (15 points) What security mechanisms, *e.g.*, chroot(), did you have to overcome? For each, did you overcome it because the implementation of it is flawed or because that security mechanism was designed for a different threat than the one that your integer overflow exploit presented to the system?

4. (15 points) What did you learn about vulnerabilities and exploits in general (*e.g.*, everything from SQL injection to memory corruption to TOCTTOU vulnerabilities) by doing this lab?

5. (10 points) Stack randomization didn't stop us from developing an exploit for Fedora Core 8. NX pages wouldn't have either, because we probably could have done a return-into-libc attack (the libraries being mapped into addresses with "00" bytes in them would not necessarily stop us). Do you think it's worth it to add more randomization and more barriers for memory corruption exploits, *e.g.*, in the next release of Fedora Core, or will that simply add a couple of days to the time it takes to develop an exploit?

For the annotated hexdump, you should clearly mark every part of your exploit in the hexdump, including the magic numbers and the different parts of your shellcode. In other words, don't just mark the whole shellcode as "shellcode," point out the different system calls and loops and such and say what they do. Your hexdump should be in hex, you can have an ASCII or other alternative format in another parallel column (*e.g.*, "`hexdump -C`") if you like.