

CSci 5271: Introduction to Computer Security

Homework 1

due: October 9, 2006

Ground Rules. You may choose to complete this homework with a partner or by yourself. If you work with a partner, submit **one** copy for your group. Use the ITLabs submit tool to submit (with submission name “hw1”) a tar-gzipped directory containing all files mentioned in this homework, by 11:59 pm Central Time on October 9. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or textbook.

Hackexploitation. This homework involves finding many different ways to exploit a poorly-written program that runs as root, to “escalate privileges” from a normal user to the super-user. The program we will exploit is the “File Clobber Versioning System,” written by Eugene, and downloadable from: <http://cs5271.net/fcvs.tgz>. To install a fully-working version of FCVS you will need to have the ability to run programs as root; un-tar the archive and (in an account with `sudo` access) type “`./install.sh`”

We will provide a virtual machine for each group (one or two students) for this homework; group members can run commands as root on these VMs using `sudo`. In order to set up the virtual machines as soon as possible, we will need to know by noon on Wednesday, September 13, what the groups are. As soon as you determine your group, and no later than noon on 9/13, send email to hw1-group@cs5271.net with the ITLabs account of all group members. After the VMs are set up, we will post instructions for accessing them.

The idea behind FCVS is to provide a very simple file repository system. Every user on the system may check their own copy of a file into or out of the repository; when a file is checked-in it “clobbers” the previous version stored in the repository, while checking out clobbers the user’s local copy. Of course this is dangerous, so FCVS maintains a list of directories that are forbidden from being either checked in or written to on check-out. FCVS also maintains a log of comments, so that users can report what changes they have made.¹

1. [80 points] FCVS is intentionally sloppy code; please never use this code anywhere else! In fact, it is so sloppy that when installed as intended (setuid root), it is full of ways that allow a user to become root. For this part of the assignment, you should find four ways to get a running command shell with UID 0 as a result of sloppy coding or design in FCVS. For each hole you find, you should produce:

- (a) A UNIX shell script that exploits this hole to terminate in a root shell. Name these scripts `spl0it1.sh`, `spl0it2.sh`, `spl0it3.sh`, `spl0it4.sh`. We will test your spl0it scripts as an ordinary user on a clean VM with a fresh install of FCVS, so your scripts will need to create any supporting file or directory structures they need in order to work.

¹As an adversary, of course, you should be skeptical about whether these measures accomplish their goals!

- (b) A text file that explains how each of the 'sploits works, named `readme1.txt`, ..., `readme4.txt`. The text file `readmeN.txt` should identify what mistakes in the source code `fcvs.c` make the exploit possible, explain how you constructed your inputs, and explain step-by-step what happens when an ordinary user runs `sploitN.sh`.

In order to get full credit, the following criteria must hold:

- each exploit script must exploit a different vulnerability in the code. We define a vulnerability by the line(s) of code that makes it possible, for example, the system call that overflows a buffer. If you're not sure about whether two sploits are distinct, **please ask us**.
- At least one of your exploit shells should be a control-flow hijacking attack of the type discussed in lecture 3. Viega & McGraw have an excellent tutorial on constructing such attacks; another tutorial on such attacks is \aleph_1 's "Smashing the stack for fun and profit," which can be downloaded from <http://www.insecure.org/stf/smashstack.txt>.² Failure to execute at least one control-flow hijacking attack will cost 10 points.
- Each exploit is worth *up to* 20 points, depending on the quality of your explanation, but a exploit that does not terminate in a root shell is worth 0 points. Make sure to test your work!³

2. [20 points] Even if we fixed all of the sloppy coding mistakes in FCVS, the *design* of the system leaves it vulnerable to some kinds of attacks. In a file called `design.txt` you should go through each of the secure design principles from lecture 4, and decide which two or three are most blatantly violated by FCVS. For these design principles, discuss how FCVS violates them and how you would change the design of FCVS to mitigate these vulnerabilities. If you feel it will be helpful, you can include pseudocode or working C to illustrate your changes.

Just for Fun. If you enjoy working on problems 1 and 2 above, keep looking for exploits! The group that turns in the largest number (greater than 4) of distinct, working (`sploitN.sh`, `readmeN.txt`) pairs will get 10 points of extra credit and the notoriety of having your name(s) mentioned in class. Assuming that more than one group turns in more than 4 sploits, the group with the second largest number will receive 5 points of extra credit, and we will give 4, 3, 2, and 1 points of extra credit to the groups that that turn in the 3rd, 4th, 5th, and 6th most working exploits (assuming 3, 4, 5, and 6 groups turn in more than 4 exploits, of course). In case of a tie we will give each group the average of the point totals – e.g. if there is a three-way tie for first, each group will get 6.67 points of extra credit. We did not make an attempt to find all of the potentially exploitable vulnerabilities in FCVS, but there are certainly at least 10.

²The smashing the stack article has a few minor bugs, including arithmetic errors, and a few constants that are off — you can find the right values using `gdb`.

³If you aren't sure whether a shell is running as root or not, you can use the unix command `id` to check — running `id` at the prompt will print your real and effective uid and gid(s).