CSci 5271: Introduction to Computer Security

Homework 1 **due: September 27th / October 4th, 2013**

Ground Rules. You may choose to complete this homework in a group of up to three people; working in a group is not required, but strongly recommended. If you work in a group, only one group member should submit a solution, listing the names of all the group members. Use the Moodle to submit a tarred and gzipped directory containing all the files mentioned in the questions, by 11:55pm Central Time on October 4th. To help encourage you to get started working early on the assignment, there is also an early submission due one week ahead of the main deadline, by 11:55pm Central Time on September 27th, described below. You may use any written source you can find to help with this assignment, on paper or the Internet, but you **must** explicitly reference any sources other than the lecture notes, assigned readings, and course staff.

Hackxploitation. 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 "Badly Coded Versioning System," written, maintained, and handed down by CSci 5271 course staff across the ages. You can download the source code for BCVS, as we call it for short, from the course web page near where you got this assignment. For its intended mode of use, BCVS should be installed in a system-wide directory, along with a private data directory, and setuid root.

Because BCVS is intended to run setuid-root, and breaking it lets you get root, we can't have you doing so directly on a CSE Labs machine. Instead, we will provide each group with a setup to run a virtual machine, and you will have root access (using the `sudo` command) inside the VM. The VMs will run on a CSE Labs cluster: we'll provide more information about running them once they're available. You can start looking for vulnerabilities in the BCVS source, and even testing many kinds of attacks, without a VM: BCVS can still run in a location like your home directory if you set up the right data files, and you can test attacks that culminate in getting a shell: it will just be a shell with your own UID, rather than a root shell. You can also recompile it if you'd like, but don't run `make install`.

The idea behind BCVS is to provide a very simple file repository system. The word "versioning" in its name is a bit of a misnomer, since it doesn't actually support keeping multiple versions of a file, which is usually the idea behind version control; instead there's just a single version in the repository. 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 BCVS maintains a list of directories and files that are forbidden from being either checked in or written to on check-out. BCVS also maintains a log of comments, so that users can report what changes they have made. Of course, we've told you that the code is buggy, so you should apply an appropriate degree of skepticism to any security claims made here in the documentation.

1. [**90 points**] BCVS is intentionally sloppy code; please never use this code anywhere else! 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 BCVS. For each hole you find, you should produce:

(a) A UNIX shell script (for the `/bin/bash` shell) that exploits this hole to open a root shell. In fact more specifically, just so there's no confusion about what's a root shell, we've created a new program named `/bin/rootshell` specifically for your exploit to invoke. If you invoke `rootshell` as root, it will give you a root shell as the name suggests; otherwise it will print a dismissive message.

Name your scripts `exploit1.sh`, `exploit2.sh`, `exploit3.sh`, and `exploit4.sh` respectively. We will test your exploit scripts by running them as an ordinary user named `test`, starting from that user's home directory `/home/test`, with a fresh install of BCVS in `/opt/bcvs`. So your scripts will need to create any supporting file or directory structures they need in order to work, and they need to run completely automatically with no user interaction. On the same CSE Labs machines with the VMs we will also provide you with a tool `test-exploit` you can use to test your exploit scripts.

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

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

- Each exploit script must exploit a different vulnerability in the code. How can we judge whether two scripts, `exploit1` and `exploit2` exploit different vulnerabilities? Imagine that you are a lazy programmer for BCVS, Inc, and someone shows you `exploit1`: a patch is in order! If there's a plausible patch the lazy programmer might write which would protect against `exploit1`, but still leave the program vulnerable to `exploit2`, then the two scripts count as exploiting different vulnerabilities. If there could be any doubt about whether two of your exploits too similar in this way, for instance if they rely on the same or overlapping line(s) of code, you should argue for why they are distinct in your `readme` files. If you're not sure about whether two exploits are distinct, please ask us before the assignment due date. (Or of course you could also keep looking for more vulnerabilities: there are enough that clearly distinct if you can find them.)

- Each exploit is worth up to **18** points, depending on the quality of your explanation, but an exploit that does not run `/bin/rootshell` as root is not an exploit as far as we're concerned. A non-working exploit will be eligible for at most 3 points of partial credit. Make sure to test your exploits carefully.

- For a further **10 points**, at least one of your exploits should be a control-flow hijacking attack as discussed in lecture. The classic tutorial on building such attacks is is $\aleph_1$'s "Smashing the stack for fun and profit," which can be downloaded from `http://www.insecure.org/stf/smashstack.txt`. Though it's detailed, it will still take some work to apply this tutorial to BCVS: for instance to find out the locations of things you'd like to overwrite, you'll need to do something like use GDB, add `printf` statements, or examine the assembly-language code.

- To help encourage you to start early, there is an early submission deadline of September 27th by which you must submit a short writeup of three potential attacks your group is working on. The attacks don't have to be working yet, and you don't need as much detail as in your final `readme` files, but you should describe the general kind of attack and which lines in the BCVS source have the vulnerability. If you're already working on more exploits or have more details to share please do so, and we'll give you more feedback. This early submission is worth **8 points**.

2. [**10 points**] Even if we fixed all of the sloppy coding mistakes in BCVS, the *design* of the system leaves it vulnerable to some kinds of attacks. In a file called `design.txt` you should choose two or three secure design principles (for instance, among the ones discussed in lecture) which are most blatantly violated by BCVS. For these design principles, discuss how BCVS violates them and how you would change the design of BCVS 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! For each additional distinct, working (`exploitN.sh`, `readmeN.txt`) pair you submit, you will get one point of extra credit. In addition, the group or groups that find the largest total number of exploits, as well as any groups that find particularly clever exploits or ones the course staff didn't know about, will be recognized in class. It's hard to know whether you've ever found all the vulnerabilities in a piece of software, but by your instructor's count, this year's version of BCVS contains at least 10 kinds of security bug appearing in 14 different code locations, and at least 10 kinds of exploits are possible, some with multiple variations.

Happy Hacking!