

CSci 5271: Introduction to Computer Security

Hands-on Assignment 1

due: September 12th - October 10th, 2014

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 each week, listing the names of all the group members. You'll be submitting answers once a week, by 11:55pm Central Time on Fridays running from September 12th through October 10th. Each week, one member of your group should use the appropriate Moodle activity to upload a tarred and gzipped directory containing all the files mentioned as required for that week. 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 Print Server,” or BCLPR (because it is loosely similar to the classic Unix printer programs LPR and LPD; “LP” originally stood for “line printer” though that term is no longer an accurate description of most printers the software is used with). BCLPR is inspired by a similarly buggy program BCVS (the Badly Coded Versioning System) used in previous years’ 5271s. You will be able to download the source code for BCLPR from the course web page near where you got this assignment. For its intended mode of use, BCLPR should be installed in a system-wide directory, along with a private data directory, and setuid root.

Because BCLPR 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 (e.g. 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 BCLPR source, and even testing many kinds of attacks, without a VM: BCLPR 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` on a real machine.

BCLPR is much simpler than a real printer server would be, enough so in fact that it is of little to no real use. It does not have any networking support, so it can only be used on a single machine. And because the virtual machines you'll be using it with don't actually have printers attached, it will never lead to the use of real paper: instead “printing” is simulated by creating files in a particular directory. The basic usage is that users invoke the `bclpr` command with the name of a plain text or PDF file they wish to print, perhaps along with other options; some time later, the “printed” version of the file (in version 1.0, just a plain text file) will appear as the simulated printer output. Just like a real printer should

only print one document at a time, BCLPR will serialize print jobs so that the pages from different documents are not intermixed. Because users should not be able to read or write other users' documents, but the print server needs to read and write all the documents, the designers of BCLPR have made it `setuid root`, so it runs with superuser privileges. BCLPR also maintains a log of the print jobs performed, which you could imagine using for billing, for instance. 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 or in comments in the source code.

The `setuid` BCLPR binary is installed in `/usr/bin/bclpr`, and BCLPR keeps its files under a directory `/var/bclpr`. In the default configuration, BCLPR supports a single printer named `lp0`; files to be printed are copied temporarily to `/var/bclpr/spool/lp0`, and the final printouts appear in `/var/bclpr/printouts/lp0`. The log of print jobs is in `/var/bclpr/bclpr.log`.

BCLPR is intentionally sloppy code; please never copy or 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. The main part of the assignment is for you to find four or more ways to get a running command shell with UID 0 as a result of sloppy coding and/or design in BCLPR. Another way of classifying the vulnerabilities is that some of them are logic errors or problems with the program's interaction with the operating system (for instance these would arise in just the same way if the program were written in Java), while others are related to the unsafe low-level nature of C which lead to control-flow hijacking. To make sure you get experience with both of these kinds of vulnerabilities, we will require that you have exploits of both kinds among your first three exploits.

To give you a feel for how security vulnerabilities evolve over time, and to provide a reason not to put all the project work off until the last minute, this year we'll be running the assignment in a weekly "penetrate-and-patch" format. Each week you'll be responsible for finding one vulnerability in BCLPR, and producing an exploit for it; this exploit is due on Friday. Then, by the following Monday, we'll post a new version of BCLPR with one or more previous security vulnerabilities fixed ("patched"), and the cycle will repeat. (Note that in addition to the usual rules about partial credit for late submissions, you will get zero credit for an exploit if you submit it after we release a patch that fixes the same vulnerability. Just another reason to submit on time.) Over time the more obvious or easy-to-exploit bugs in BCLPR will get fixed, so you will have to find more subtle bugs and more sophisticated exploits.

For each hole you find, you should submit:

- (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 script `exploit.sh`. 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 BCLPR. 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 should use to test your exploit scripts.

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

In choosing which vulnerabilities to patch each week, we will start by looking at which vulnerabilities were most commonly exploited, so there is a good chance that your old vulnerabilities will no longer work at all after the patch. However even if an old vulnerability happens to still work, you still need to submit an exploit for a new vulnerability each week. How can we judge whether two scripts, `exploit1` and `exploit2`, exploit different vulnerabilities? Imagine that you are a lazy programmer for Badly Coded, 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 turning the second one in. (Or of course you could also keep looking for more vulnerabilities: there are enough that are clearly distinct if you can find them.)

Because we won't be patching the vulnerabilities all at once, you have some flexibility in when you spend your time on this project: you might be able to save time later by finding a lot of different vulnerabilities early on. Since we'll be patching roughly in order of increasing difficulty, you'll want to use your simplest exploits first. Of course you always run a risk that vulnerabilities will be patched if you save them: and even if the vulnerability still exists in a newer version, other changes to the program might mean that the exploit needs to be a bit different.

At least one of your first three exploits needs to 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 BCLPR: 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.

In the course of the assignment there are a total of 100 regular points available of the first four weeks, and then extra credit points in the fifth and final week. Specifically the points are split up as follows:

- Week 1: 10 points, due Friday September 12th.

In the first week of the assignment, you should familiarize yourself with reading the BCLPR source code and writing automatic exploit scripts. But to make the searching a bit easier in the first week, we've included a vulnerability that's particularly easy to exploit: it's really just a "backdoor," a behavior left in the code by a disgruntled former employee that will give you a root shell directly if you can figure out the right way to execute it. It will be a little bit camouflaged, but it shouldn't be too hard to notice if you read carefully.

- Week 2: 20 points, due Friday September 19th.

In the first patch, the security experts at Badly Coded, Inc., will definitely fix the backdoor mentioned in the first week. But other than what was patched, you'll have your choice among the remaining vulnerabilities in the second week; probably some pretty simple exploits will be possible.

- Week 3: 30 points, due Friday September 26th.

The third week is your deadline for submitting exploits of both kinds. So if your first two exploits were for OS/logic errors (the backdoor counts in this category), your third week exploit needs to be control-flow hijacking. (Conversely, if your first two attacks were control-flow hijacking, your third week exploit needs to be something else.)

- Week 4: 40 points, due Friday October 3rd.

If you successfully found exploits of both types by the third week, in the fourth week you can once again use an exploit of either type: but probably most of the easier exploits will have been patched by now. This exploit is worth 30 points.

But even if we fixed all of the sloppy coding mistakes in BCLPR, the *design* of the system leaves it vulnerable to some kinds of attacks. So taking the white-hat perspective, for the remaining 10 points, 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 BCLPR. For these design principles, discuss how BCLPR violates them and how you would change the design of BCLPR to mitigate these vulnerabilities. If you feel it will be helpful, you can include pseudocode or working C to illustrate your changes.

- Week 5: $5 \cdot n$ points extra credit, due Friday October 10th.

After the fourth week we will definitely have fixed all the easily exploitable bugs in BCLPR, and we might even re-enable some of the security hardening mechanisms we'd earlier removed. But it's still not really secure. So if you're enjoying finding and exploiting bugs, you can keep going. You'll earn 5 points of extra credit for each additional unique exploit you find, limited only by the total number of remaining security bugs in BCLPR. In addition to the extra credit, the team(s) that find the largest total number of bugs may also receive some special in-class recognition.

A portion of your grade for each exploit will depend on the quality of your explanation, to make sure you really understand what's going on. But an exploit that does not run `/bin/rootshell` as root when invoked by `test-exploit` 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.

Happy Hacking!