COSC 366: Introduction to Cybersecurity

Homework 1 **due: October 6th, 2020**

Ground Rules. You may choose to work with up to two other students if you wish. Only one submission is required per group. Please ensure that at the top level of your submission (it should be a ZIP file or other Archive - you could use tar) has a text file named `students.txt` exists containing all group member names, **this is required, even for groups of one student**. Please use include a document named `id.txt` which contains your student ID on a single line. Work must be submitted electronically via Canvas.

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 security faculty across the ages, and downloadable from the course website. To install a fully-working version of BCVS 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`"

Because installing the program requires root access, and breaking this very poorly written software will give you root access **you should only test this on the VM provided.** We have provided a virtual machine image with BCVS already installed on the course website. Please see the *lab1_vm_instructions.txt* file for help on running the VM which you can also download from Canvas under the Files tab. Again, PLEASE read the instructions file, it also contains hints for the lab and getting BCVS to work. Group members can run commands as root on these VMs using sudo.

The idea behind BCVS 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" [1] 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 that are forbidden from being either checked in or written to on check-out which is called the block list. You can specify this by putting a file named *block.list* in the repository directory (see the VM instructions file on setting up the repository for BCVS).. BCVS also maintains a log of comments, so that users can report what changes they have made. [2] These comments are stored per file in the BCVS repository as well.

1. [90 points] BCVS 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 BCVS.

[1] `https://en.wikipedia.org/wiki/Clobbering`

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

Essentially, what you will be doing is running BCVS from your directory as a student non-root user and trying to get it to give you a shell, which will be as the root user on the VM (user *theboss*). Do this with user input! Or files on the machine! Or anything you can think of! There are no advanced protections on this binary (ASLR, NX-bit, etc.). For each hole you find, you should produce:

(a) A shell script that exploits this hole to terminate in a root shell. Name these scripts `sploit1, sploit2, sploit3, sploit4` and place them at the top level of your `git` repo. We will test your sploit scripts, running them from the top level of your submission archive, as an ordinary user on a clean VM, so your scripts will need to create any supporting file or directory structures they need in order to work. We will test your sploits using the *student* user on your VM, ensure that your script works for this user. In addition, your scripts must work *without* user interaction, build them accordingly. You can write the scripts such that they call BCVS with the input you need them to use to generate a root shell.

(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 `bcvs.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 sploit script must exploit a different vulnerability in the code. How can we judge whether two scripts, `sploit1` and `sploit2` exploit different vulnerabilities? Imagine that you are a lazy programmer for BCVS, inc, and someone shows you `sploit1`: a patch (a bug fix) is in order! If the simplest patch that disables `sploit1` is still vulnerable to `sploit2`, then the scripts exploit different vulnerabilities and you will get credit for both vulnerabilities. Your `readme.txt` file should argue your case in this fashion, if two scripts rely on the same or overlapping line(s) of code. If you're not sure about whether two sploits are distinct, **please ask us - before the due date** (ask Eric or I, not Piazza - you can discuss things there but do not give other groups answers).

- Each sploit is worth *up to* **18** points, depending on the quality of your explanation, but a sploit that does not terminate in a root shell is worth 0 points. Make sure to test your work![3]

- In order to get groups thinking about this homework early, you are required to submit *by September 22nd* by 23:59:59 a short write up on three exploits they are currently working on. Submit this by uploading a file `leads.txt` to the Programming Assignment 1 Leads assignment on Canvas. This write up does not need to be large, it simply

[3]If 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).

needs to state the nature of the exploit and reference what lines in the BCVS code are vulnerable. This is worth **10 points**.

2. [**18 points**] The final **18 points** are available if at least one of your sploit shells is a control-flow hijacking attack of the type discussed in the lecture on buffer overflows. An excellent tutorial on constructing 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`.[4] Also it is for 32 bit systems, we are now on 64 bit systems.

Just for Fun. If you enjoy working on problems 1 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 6.33 points of extra credit. We did not make an attempt to find all of the potentially exploitable vulnerabilities in BCVS, but there are certainly at least 12.

Happy Hacking!

[4] This article has a few minor bugs, including arithmetic errors, and a few constants that are off — you can find the right values using gdb or by inserting appropriate "printf" statements.