Program #2: Using a Source Code Analyzer

Due: April 27, 2015 **Points:** 100

This problem requires you to do some source code analysis. To do this assignment, you need to download the file *lsu.tar* from the homework web site, copy it onto a system at the CSIF, and unpack it. Once on the CSIF, type

```
tar xvf lsu.tar
```

and you will find a directory named *lsu* containing the program *lsu*. It consists of several source files, a *Makefile*, and a large number of configuration files. The version you have downloaded is preconfigured to compile on the CSIF systems. For the purposes of this assignment, assume that *lsu* will run as a setuid-to-*root* program; that is, when you execute it, it runs as though *root* executed it.

This assignment consists of several steps. First, we'll run a source code analyzer over the program to find possible errors. Then, we will examine a couple of error reports.

The program *sourceanalyzer* is a tool written by Fortify Software. It analyzes programs for possible vulnerabilities, and when it finds one, it describes it and gives a trace of the routines involved in an attack exploiting the vulnerability.

First, run the program. See the document **The Fortify Source Code Analyzer** for details on how to do this. Remember, this program uses *make*, so you want to follow the directions in the section "More Complex Programs". The document also explains how to interpret the results.

When you run the second command, be sure to save the output in a file; it's long, and you'll need to refer to it later.

Now, look at the output. You will see many potential errors. We're going to focus on three.

1. First, search in the output for the string "27818011B74AAF7F0FFC2673186E546A"; you should see that the next line begins with "util.c(60)" (or some path name ending in this). This indicates that there is an easily-exploitable ("high") vulnerability through which a user overflow a buffer ("Buffer Overflow") or use a format string to change something ("Format String") in the call to `sprintf`, argument 2, at line 60 of the file *util.c*.[1]
2. Next, search in the output for the string "ADE01E7CEF8599C1929156D862BFD432"; the next line should begin with "syntax.c(61)" (or some path name ending in this). This indicates that there is a way to manipulate the path name of a file ("Path Manipulation") to have the *fopen*() function at line 61 of the file *syntax.c* open a file that you should not be able to, but that this requires some skill to do ("medium").
3. Finally, search in the output for the string "A74104DA59CB2F399724D04ABB303230"; the next line should begin with "lsu.c(136)" (or some path name ending in this). This indicates that the system call *setgid*() at line 136 of *lsu.c*' makes a change in privileges ("Privilege Management") that is often misused. But this is hard to exploit ("low").

For the first two, describe in detail the flow of data that would allow an attacker to exploit the flaw. For example, for an integer overflow, you would say something like this:

> The attacker passes $2^{31} - 1$ arguments to the program (via the function `main` in file *main.c* at line 25). The program then appends 3 more arguments (see line 36 of *main.c*). This causes the counter for the loop in which the arguments are appended to overflow (see lines 34–38 in *main.c*).

Of course, this is a made-up example, not one drawn from *lsu*. But it gives you the level of detail we want.

Please copy the messages from the run of *sourceanalyzer* describing the vulnerability and the trace of functions and files below it. Then write your description beneath.

For the third one, analyze the report. Is this an exploitable vulnerability? If so, explain how to do it; if not, please explain why it is not exploitable (or not a vulnerability).

[1] Remember, the first argument is argument 0, so this is really the third argument