Program

Due Date: June 7, 2005 **Points**: 100

The goal of this program is to give you some experience in analyzing software for vulnerabilities. To do this assignment, you need to download the file *lsu.tar* from MyUCDavis, 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. The ones of interest to us end in ".c".

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

Step #1: Generating the Analysis File

Run the program

```
/usr/local/FortifySoftware/SCAS3.1-EE/sourceanalyzer gcc *.c
```

from *within* the *lsu* directory. Save the output in a file; it's long, and you'll need to refer to it later.

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.

Step #2: Find some problems

Look at the output. You are to choose one flaw from each of the following categories:

- Buffer overflow (pick one marked "high". Note: do ***not*** pick the one at util.c(36)!)
- Process control
- Integer overflow
- Race condition: TOCTOU

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

1. Attacker passes 2^{31}–1 arguments to program (via main function in file main.c line 25)
2. Program appends 3 more arguments (see main.c line 36)
3. Counter for loop in which the arguments are appended overflows (see main.c lines 34-38)

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. You can pick any instance of the type of flaw except for the ones noted for buffer overflow, above.

Step #3: Look at a Limit

Now go back to the buffer overflow at util.c(36). It's the first buffer overflow marked "high" in the output, assuming you used the command given above. Analyze this report. Is it an exploitable buffer overflow flaw? If so, please explain exactly where the overflow occurs, and why. If not, please explain why it is not exploitable (or not a vulnerability).

Extra Credit

(*10 points each*) State how to fix each of the vulnerabilities you discussed in step 2. Please fix the program files, and submit them along with your assignment.