CS 108, Midterm 1

Terms and Conditions. This is a take-home, open-book, open-manual, open-shell exam. The solutions are due by morning of Wednesday February 24.

You are expected to use (at least) DTrace, the OpenGrok source code browser[1], and the Modular Debugger's running kernel inspection capability[2], and any other tools you find useful. The documents in the class directory `http://www.cs.dartmouth.edu/~sergey/cs108/` and the textbook's index might be useful, too.

For each problem, you should "show your work": the output of the above tools on your actual OpenSolaris platform (virtual or physical). For OpenSolaris kernel code lines, provide the filename and the line number, or the OpenGrok URL pointing to the right line.

You are allowed to discuss the use of tools with your fellow students, *but not the problems or solutions themselves.* For example, sharing a tracing trick is OK, but sharing part of a solution as such is not. Note that in most exercises you are free to choose your targets (which may help you avoid conflicts with the above rule). If in doubt, ask.

Note that Problem 4 is optional, but is intended to provide a direction for a good class project.

Problem 1. Dissect the virtual address space of a UNIX process (say, `ps(1)` or `ls(1)`), both in user and kernel context.

Show where various parts of the process and supporting shared libraries are loaded[3]. Show where ELF segment headers are loaded. Show where the kernel data structures describing the process are. Show where the kernel stack is. Show where the page tables of the process are. Show any other data structures of interest.

Bonus points: Perform surgery on the process.

For example, suppose that you are a system administrator, and you mistakenly started editing a configuration file without `pfexec` or `sudo`. When it's time to save the file, you discover that your process lacks the permission(s) to do so. Can you temporarily give the process that power without having to restart it? (Of course, in reality you have other choices, such as saving the file elsewhere and then copying it, etc.)

For more bonus points, perform a rootkit trick: hinder some OS functionality without crashing or incapacitating the process, such as hide it, make it unkillable, or hide its specific open file or socket.

Problem 2. OpenSolaris supports multiple executable file formats (e.g., *ELF*, the older *a.out*, and the Java environment's feature *Javaexec*). Identify the data structures and code line(s) involved in this support.

Problem 3. Using the DTrace's *lockstat* provider[4], find the most contended lock on your platform for the UNIX tool of your choice (ps(1), ls(1), or anything else in standard UNIX (1))

[1]`http://src.opensolaris.org/source/search?path=uts&project=onnv`
[2]`mdb -k`
[3]Some hints can be found at `http://hub.opensolaris.org/bin/view/Community+Group+observability/procfs`
[4]For some hints on lockstat, see `http://wikis.sun.com/display/DTrace/lockstat+Provider`

and the code path on which most contention occurs. You will, of course, need to script running the tools enough times to be statistically “sure”.

Problem 4. (optional, open-ended, to start you thinking about a possible project):

Imagine a visualization of the physical RAM use by different processes and the kernel, in which each physical page is represented by a pixel. The color of the pixel describes the current use of the page (free, used for file I/O cache, kernel data structures, or for some user process’ code, data, or stack).

Create such a visualization prototype based on DTrace and a scripting language that can handle DTrace output (Ruby, Python or Perl)[5]

[5]For Ruby, see `http://ruby-dtrace.rubyforge.org/` for DTrace bindings.