CS 258, Midterm Exam

Terms and Conditions. This is a take-home, open-book, open-manual, open-shell exam. The solutions are due by morning of Tuesday February 21.

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/cs258/` 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 platform (virtual or physical). For OS 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 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: The default system for these problems is Illumos. For each of the Problems 1, 2, and 3, you may choose to do that problem on GNU/Linux instead of Illumos if you so desire; note, however, that Illumos provides more versatile and stable tools for examining the kernel, so it will likely be more work, unless you've been working on a Linux project idea and practiced with Linux tools already. Problem 4 is for Solaris only.

Problem 1.

- Using DTrace or MDB[3] or both, measure how many pages of dynamic libraries are shared by processes at some point in time. Start with *libc*, which is used by most processes. Document the process tree at the time of your measurement.
- Estimate what part of *libc* does not appear to be used while running a typical set of processes (compile some programs, run a webserver or some other server; script your load).
- Based on the above, make an estimation of the trade-offs of dynamic vs static linking in terms of RAM use.

[1] `http://src.illumos.org/source/`

[2] `mdb -k`

[3] If you choose to use GNU/Linux, use comparable tools; you mileage may vary.

Problem 2.

- Enumerate the functions in the kernel that parse the *ELF* binary format when a process is started from a binary executable file (e.g., by an *exec* system call). For each function, document briefly which part of the *ELF* format it deals with.
- Edit a binary file (of a simple program, say "Hello, world") to cause some of these functions to exit with an error. Extra points for causing this in deeper functions (further down the callstack).

Problem 3. Write a program that messes with its own dynamic linking. Make it call the dynamic linker for the same function over and over (for example, make it print a string a hundred times, resolving the *puts* or *printf* symbol every time). Use it to measure the overhead of dynamic linking. Extra points for other ideas that make the dynamic linker do something unusual. (Instant High Pass for any memory corruption vulnerabilities discovered in the dynamic linker.)

Problem 4. In Bonwick 2001 USENIX Tech paper,[4] Figure 3 shows three boundaries: 1) between the CPU and the Depot layer, 2) between the Depot and the Slab layer, 3) between the Slab layer and the *Vmem* arena layer. With DTrace, instrument the functions that actually perform these transitions in the kernel code and write a (userland) program that causes "worst case" allocation patterns (w.r.t. to timing, falling through to the next layer caching, or locking) in the kernel. You program need not make sense or output anything in particular; all it needs to do is cause many kernels objects to be created and freed, in such a way that it's *Vmem*'s "worst nightmare."

Note: there are many different types of data structures allocated in the kernel via the *Vmem* allocator. You may confer with your classmates and pick a particular kernel data structure to target. We can then have a competition between several "worst case" programs targetting the same structure. The winner gets extra points.

[4] `http://static.usenix.org/event/usenix01/bonwick.html`