CS 258, Midterm Exam, Winter 2014

Terms and Conditions. This is a take-home, open-book, open-manual, open-shell exam. The solutions are due by noon of Monday February 17.

You are expected to use (at least) DTrace, the OpenGrok source code browser[1] and the Modular Debugger's running kernel inspection capability (`mdb -k`), 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).[2] 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.

Submit your work as a text or PDF file, or as a tarball (.tar.gz file) that contains a directory named `<YourName>-midterm` (with your solutions inside :)).

Note: The default system for these problems is Illumos, due to the great flexibility of DTrace and MDB. You may choose to do some or all or the problems on GNU/Linux instead of Illumos if you so desire (e.g., using *SysTrace* and *Kprobes*); note, however, that Illumos' tools for examining a running kernel are much more versatile and stable. Linux will likely be more work, unless you've been working on a Linux project idea and practiced with Linux tools already.

Problem 1. *Hide a process: a poor man's rootkit.*

- Using MDB or DTrace, manipulate kernel data structures so that a running process doesn't show on `ps ax` but still runs normally (check that by interacting with the process after your kernel manipulations).
- Using the same tools, make it so that signals sent to the process by `kill -s <signal> <pid>` are ignored. Then find another way to send a signal so that it is *not* ignored.

Hint: In MDB, use the command `$W` to enable writing memory, and writing formats (see `::formats ! grep write`) to actually write memory. In DTrace, use the `-w` to enable "destructive" actions to write memory.

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

[2] You can record your entire shell session with the `script <filename>` command, or, when working in MDB, with the `::log <filename>` command.

Problem 2. *Battle of the debuggers.*

Start a process under `gdb` or attach `gdb` to a running process. Set a breakpoint in the process. In MDB, navigate to and show the contents of the process' memory where the breakpoint is set. Using memory examination commands, describe and demonstrate how `gdb` implements its breakpoints.

Extra credit: Explain in detail (showing traces and relevant data structures) how `gdb`'s breakpoint handing code gets invoked when a breakpoint is hit.

Problem 3. *The cow jumped over the loop.*

In the following C program, add code where indicated to avoid the effects of the indicated `puts` in the `for` loop but still hit its ending statements. Your program should produce the output shown below, and should **not** use `goto`, `return`, `setjmp/longjmp`, `mprotect`, `exit` or `exec`-family system calls, but may read and write any place its process memory as needed and allowed by the memory protections.

```
#include <stdio.h>

int main()
{
   int i;
   puts("Starting program...\n");

   /*
    *  Add your code here.
    */

   for(i=0; i<=100; i++){
       puts("Moo!\n");   /* Avoid this */
   }
   printf("Finishing program, %d iterations.\n", i);  /* Hit this! */
   return 0;
}
```

Output:

```
Starting program...
Finishing program, 101 iterations.
```

Problem 4. *Dynamic symbol popularity contest.*

What dynamic symbol occurs in the largest number of processes running on your system? (Assume that `ps ax` gives you a representative view of your process load). More specifically:

1. Answer the above question for any symbols present in the program executable's dynamic symbol table, without regard for their type or whether or not they have been resolved by the dynamic linker by the time you look.

2. Answer the above question limited to symbols that refer to dynamic library functions, whether or not they have been resolved & linked.

3. Answer the above question limited to dynamic library functions that have been already resolved & linked.

Extra credit: Generate information to answer questions (1) and (2) with one-liner shell commands (pipes and any command-line `awk`, `sed`, `perl`, `python` or `ruby` are allowed.[3] For (3), I don't immediately see a one-liner solution, so all the more credit if you find one!

Problem 5. *Lock and load.*

Write a program that will exercise the Magazine and Depot layers of the Illumos Kmem/Vmem allocator as described in Bonwick's 2001 paper.[4] You program should trigger magazine allocations of an OS object, and then cause the Depot layer to provide new magazines. Show that this happens by setting appropriate DTrace probes and recording a trace. Point out the work of different allocator layers in the trace.

You program can be in any language, not necessarily C.

Note that this problem is OpenSolaris/Illumos-specific and doesn't easily adapt to Linux. If you would like a similar one on Linux, please talk to me.

[3] Google "X one-liners" where X is one of the above languages; the trick is to specify the full program on the command line and then pipe the output to another program, such as `uniq`, `sort`, `cut`, etc., with appropriate options. There is no limit on how many commands are piped.

[4] http://www.cs.dartmouth.edu/ sergey/cs258/bonwick01.pdf