Ibssa, Girum
Cpe 453, Section 02
Assignment 4
May 24, 2013

Assignment 4: Minix Secret Driver

This is the template I passed out in the lab (well, slightly modified). It works reasonably well to have each step of the lab be a \section and each component below it a \subsection. For the "problem set" labs an `enumerate` environment is probably more appropriate.

2 Overall Architecture of the Driver

Minix uses a non-monolithic, message-based, callback framework for its kernel. A mouthful, but this just means that Minix is structured to be a very modular, framework-y codebase. This is great for me, since all I really have to do is write callback functions for `open()`, `close()` and `transfer()` to finish the assignment. To me, this means that all my Secret has to do is define handlers for:

- Parse the incoming `message *` for its sender's credentials in the callback for `open()`.
- Persist who said owner is globally.
- Bounce users who try and access my Secret if they ask for credentials that don't agree with my business logic. E.g., no read/write access (why would you write a secret and then immediately read it again?) and no reading a secret you're not the owner of.
- Lastly, allow processes to transfer ownership of my Secret to each other via a custom SSGRANT flag that can be retrieved programmatically.
- Of course, I need to register my driver as a legitimate Minix driver in its kernel, so I have to modify a few Minix files before everything comes together to work.

3 Description of Driver Implementation

3.1 Development Environment

- Deployed my driver to a Minix 3.2.1 installation.
- Installed Minix on a VM: VMware Fusion 5.0.0, hosted on my Mac.
- Wrote driver code in Sublime Text 2 (boo) and read up on the Minix source in Eclipse CDT.
- This report is written in TeXShop for Mac.

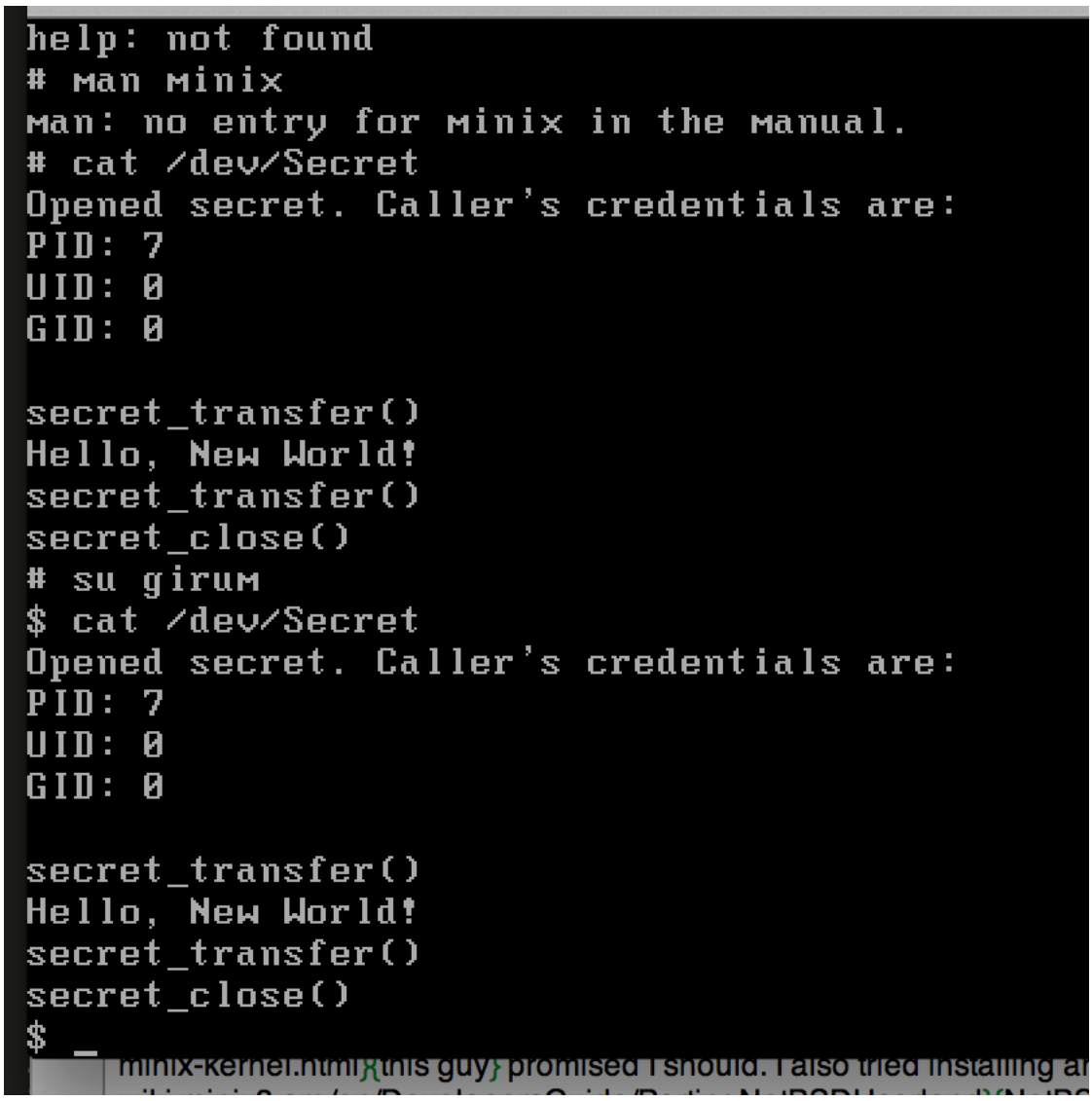
3.2 Files modified

- `/etc/system.conf` to register my service in the OS.
- `/usr/src/include/sys/ioctl.h` to register the SSGRANT flag for ownership between processes transfers later.

3.3 The code

Live version available here.

4 Driver Behavior



5 Results

5.1 Problems encountered

- I couldn't compile Minix locally. Eclipse thus wouldn't work for this project. This was because Minix doesn't use standard `gmake` for its Makefiles. Instead, Minix uses some unknown custom `make` utility that includes a ton of conditional logic used to auto generate parts of the Makefiles. In the end, I couldn't manage to move Minix's version of `make` over to my local machine.

- Right as I began coding (after 6 hours of reading, config and more reading), my `getnucred(2)` call refused to work properly. I'm passing it the from `m_source` from the `message *` passed in via the framework, and fill up a `struct ucred` initialized on the runtime stack. The PID outputs "7" when it should be 697, and both the UID and GID output "0". What's worse, this output is consistent across both of my user accounts.

5.2 Solutions attempted

- In attempting to fix the compilation issue (really just a `make` issue, since Minix doesn't use `gmake` and instead relies on heavy conditionals to auto generate its Makefiles) I tried copying over the entire `/usr/src/tools` folder like this guy promised I should. I also tried installing and using NetBSD make to no avail. I gave up and switched to Sublime Text, but I realize now how dependent I am on my IDE. Without Content Assist, I code about as rapidly as a Galapagos Tortoise.

5.3 Results obtained

Just the modified hello world program so far. And a lot of new knowledge on Minix.

5.4 Lessions learned

- Don't write Makefiles in non-`gmake`. Not being able to compile Minix on my own machine was by far my biggest headache. But that's more for the writers of Minix to learn from.

6 Other things (as necessary)

Source code that I have submitted via handin to asgn4, and available online at my GitHub account.