6.828 Project Writeup

Patrick Hulin Steven Valdez

December 12, 2012

1 Goal

The goal of this project was to implement a capability model similar to the one in the Capsicum paper in xv6. In order to show capabilities in action, we also ported a networking stack, lwip, into xv6.

2 Networking

To implement networking, we had to port both a ethernet driver and a TCP/IP network stack. For the driver, we ended up modifying a non-working port of the NE2K_PCI driver by Shintaro Sheki, which we ended up having to fix to work with the memory model in the version of xv6 we were using. In addition, we ended up porting over lwIP, a lightweight TCP/IP stack commonly used for embedded systems. As a starting point, we used the sys_arch bridge by Henry Hu. Unfortunately, due to various inconsistencies with the semantics used by lwIP and the version of xv6 we were developing on, a number of reduncancies were created that we didn't have time to solve, such as the thread and process relationship, and the sharing of page tables. Unlike JOS, which has a builtin system for IPC, we had to use a semaphore/mailbox system for communication between the networking code and other user code. In addition, we had to make sure that the network code, which was running at the user level, could receive memory from other processes, and as such ended up using a modified page table to give it access to some of the required memory.

Due to the lack of a physical environment, we ended up using QEMU's builtin features, such as network redirection to provide the bridge between the host machine, and the guest network adapter and system.

3 Security model

Processes enter capability mode by calling cap_enter(), a syscall which sets a flag in the proc struct. After calling cap_enter, the process should lose access to all global namespaces: the filesystem, network access, process information, etc. This information is all encapsulated in file descriptors created before cap_enter, which hold rights flags which allow only certain kinds of access. For example, sh holds a file descriptor for /bin which only permits reading and executing files, and a descriptor for /tmp which permits all types of access. Thus, users in a capabilitied shell cannot modify the executables in /bin, and cannot run executables elsewhere, as sh only searches /bin for executables.

4 Challenges

Our most crucial challenge was porting the network stack, which took a lot of time, research, and debugging. Luckily, lwip is relatively easy (as network stacks go) to port to new platforms. Doing so required implementing a few operating system primitives which stock xv6 does not have, but this was not terribly difficult. After that, we had to implement the capability system. In capability mode, processes are forced to use only relative paths or to base every filesystem access on a directory file descriptor, as sh does with /bin. We implemented these via the openat syscall. Next, we implemented a networked shell; in our security model, there should be no way for users to escape to other directories on the system, assuming correctness of the capability implementation. For example, we placed a secret file in /secret/password; no remote user should be able to read it.

5 Results and further work

Our system is a fairly rough prototype; the security model and its implementation have not been rigorously tested and evaluated. Given further time, that would be our first priority. Extending it into a more complete system would require better restrictions on network access, etc, and would need to be implemented on a more full-featured operating system: xv6 simply does not have enough kinds of information to restrict.