GOOGLE'S MOTION TO COMPEL COMPLIANCE WITH PATENT RULE 3-1

Exhibit 3

Google Chrome, Sandboxing, and Mac OS X

Monday, June 01, 2009

Sandboxing is a technique that Google Chrome employs to help make the browser more secure, and was discussed in a previous blog post. On Windows, getting a process sandboxed in a way that's useful to us is a pretty complicated affair. The relevant source code consists of over 100 files and is located under the sandbox/ directory in Chromium's Open Source repository. But for our Mac and Linux ports, sandboxing is a very different story. On Linux there are a number of different sandboxing mechanisms available. Different Linux distributions ship with different (or no) sandboxing APIs, and finding a mechanism that is guaranteed to work on end-user's machines is a challenge. Fortunately, on Mac OS X, the OS APIs for sandboxing a process are easy to use and straightforward.

Sandboxing on the Mac

Starting a sandbox involves a single call to sandbox_init() specifying which resources to block for a specific process. In our case we lock down the process pretty tightly. That means no network access, and very limited or no access to files and Mach ports.

When Chromium starts a renderer process, we open an IPC channel (a UNIX socketpair) back to the browser process before turning on the sandbox. Any resources a process owns before turning on the sandbox stay with the process, so this channel can still be used after the sandbox is enabled. When we want to pass a shared memory area between processes, we send it over as an mmaped file handle using the sendmsg() API. We don't need to do anything else, as Apple's sandbox API is smart enough to allow access to file descriptors passed between processes in this manner even if the receiving process itself is forbidden from calling open().

One sticky point we run into is that the sandboxed process calls through to OS X system APIs. There is no documentation available about which privileges each API needs, such as whether they need access to on-disk files, or call other APIs to which the sandbox restricts access. Our approach to date has been to "warm up" any problematic API calls before turning the sandbox on. This means that we call through to the API, to allow it to cache whatever resource it needs. For example, color profiles and shared libraries can be loaded from disk before we "lock down" the process. To get a more complete understanding of our use of the sandbox in OSX, you can read the OSX sandboxing design doc.

As we continue the porting efforts for Chromium on the Mac, it's very satisfying to see the puzzle pieces fit into place alongside the native system APIs. It's important to us that the Mac port of Chromium feels and performs like a native Mac application, and that it provides the kind of high-quality experience Mac users expect.

Posted by Jeremy Moskovich, Software Engineer

25 comments:

Juvenn said...

So, what about Linux?

June 1, 2009 11:48 PM