

 dropbox.tech/infrastructure/going-deeper-with-project-infinite

1. Starting from first principles
2. Performance
3. Security
4. So Instead...
5. But wait! There's more!

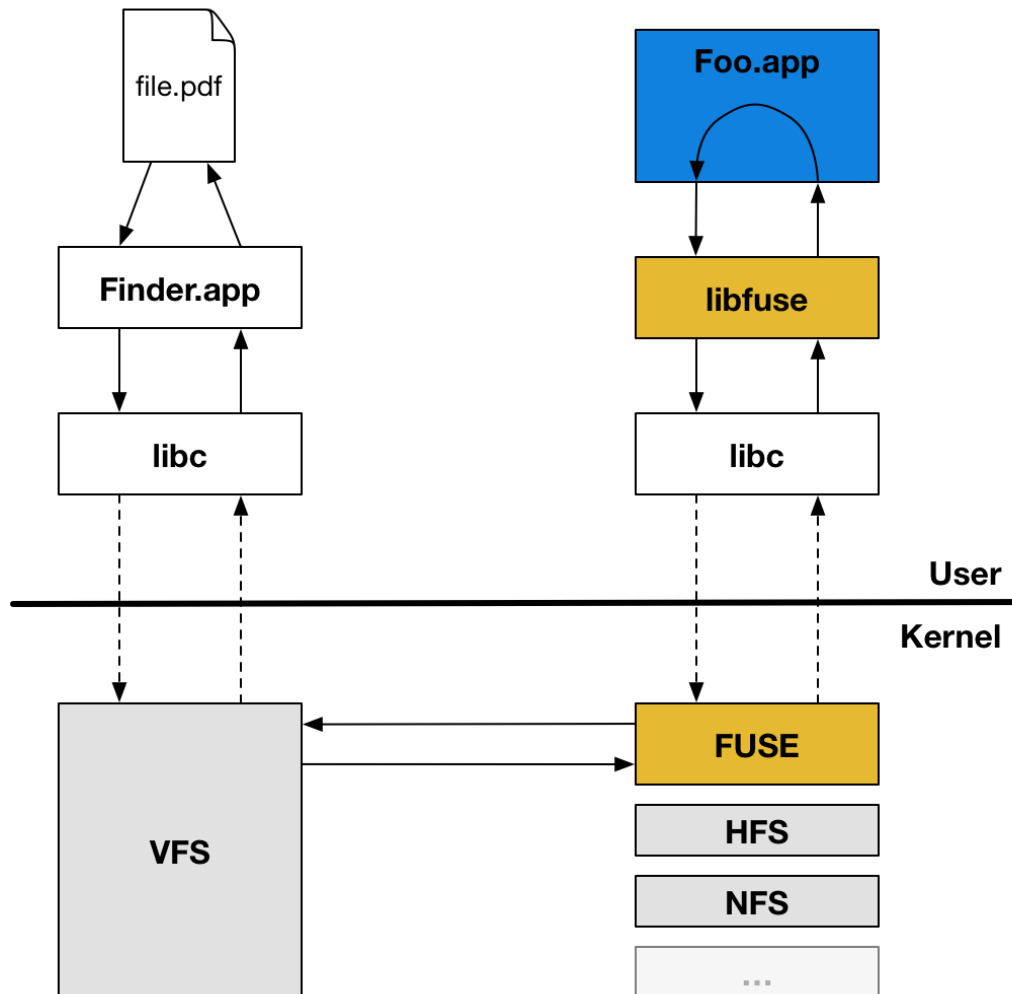
Traditionally, Dropbox operated entirely in user space as a program just like any other on your machine. With Dropbox Infinite, we're going deeper: into the kernel—the core of the operating system. With Project Infinite, Dropbox is evolving from a process that passively watches what happens on your local disk to one that actively plays a role in your filesystem. We have invested the better part of two years making all the pieces fit together seamlessly. This post is a glimpse into our journey.

Our earlier prototypes around solving the “limited disk-space problem” used something called FUSE or Filesystems in Userspace. FUSE is a software interface that lets non-privileged users create their own filesystems without needing to write a kernel extension. It is part of the kernel itself on some Unix-like operating systems and OS X has a port that is available as a dedicated kernel extension and a libfuse library that needs to be linked by a program in user space.

Motion Offense Exhibit 2018
1/6

Performance

Since FUSE filesystems are implemented in large part in user space, any file operation usually requires an extra user-kernel mode switch (one context switch between the application issuing the system call and the VFS in the kernel and an extra one between the FUSE kernel extension and the libfuse user space library). There's quite a lot going on, as you can see in the illustration below.



While context switches are usually quite inexpensive, this extra overhead for every file operation unfortunately leads to a degraded performance that we didn't want our users to experience when interacting with their files in Dropbox.

Security

We take security seriously. We do everything we can to protect our users and their data. This includes having internal Red Teams, running a bug-bounty program, and hiring external pen-testers on a regular basis to help us discover vulnerabilities in our products.

The various FUSE libraries on OS X are implemented as kernel extensions and introduce too much complexity and risk for us to feel comfortable with distributing as part of our Desktop client.

So Instead...

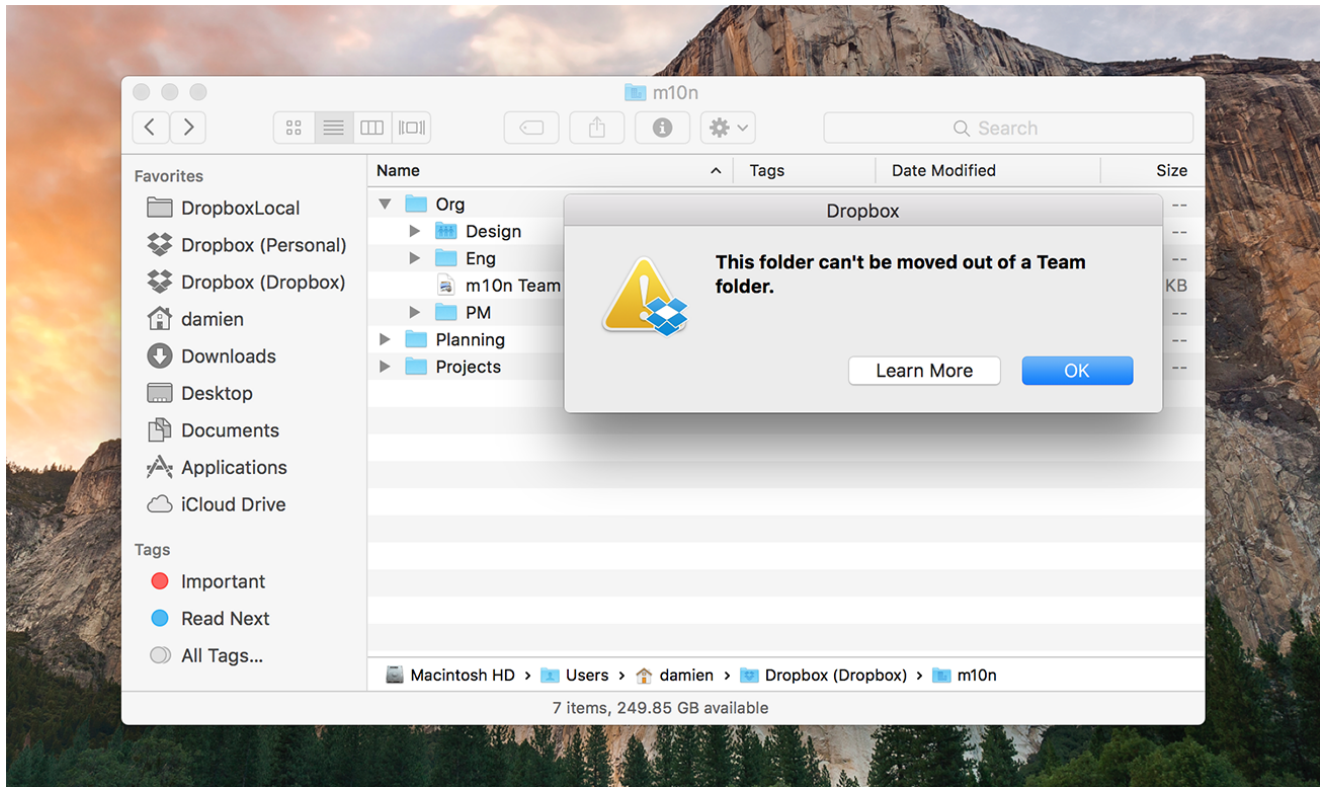
After exploring the option of using FUSE, we realized that there are many benefits to writing our own custom kernel extension: we are able to achieve minimal performance overhead while also ensuring that we understand 100% of what we're serving to our users. And when we control the interface boundary, we can do our best to push as much non-performance critical machinery up into user space, further improving security.

But wait! There's more!

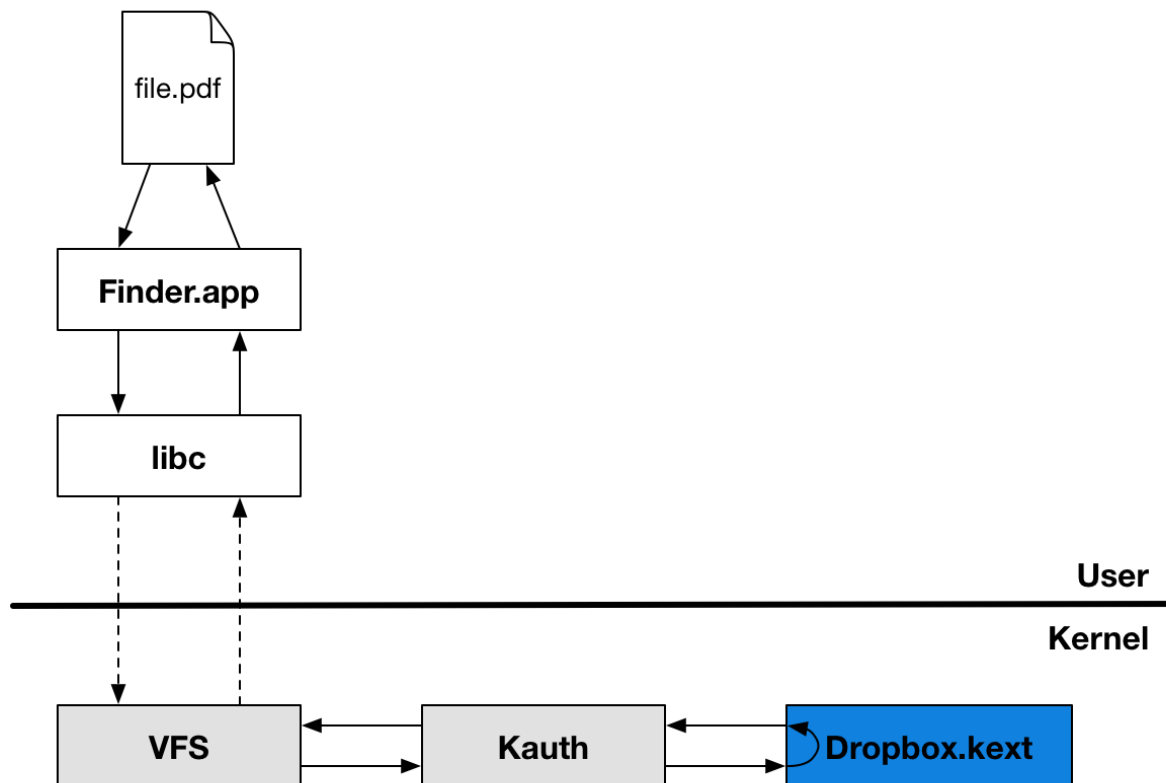
As we've been building out our kernel extension, we have also begun to look at what other long-standing user problems we can solve. It turns out there's a lot we can do.

We've seen the number of companies that rely on Dropbox Business soar past 150,000 since we launched it just three years ago. With so many teams on Dropbox, we increasingly hear about a scenario we call the "untrained intern problem." Imagine you are working with a bunch of other people on a project and collaborating through a Team folder on Dropbox. Summer is quickly approaching and you've brought on an intern. The intern, never having used Dropbox before, moves a folder from inside their Team folder to their Desktop, not realizing that they've simultaneously removed access to this folder for everyone else in the company. Now of course this folder could be restored, but don't you wish there was a better way so this could have been prevented from even happening?

Rolling out today, starting with Dropbox Enterprise customers, is a *better way*. Now, in order to protect the organization and shared content, when someone performs such an operation, they will be warned with a dialog that looks like this:



How does this work? On Windows, we use Copy Hooks, but on Mac we had to dig a little deeper. We use the Kernel Authorization (or Kauth for short) kernel subsystem in our kernel extension to manage file authorizations within the BSD portion of the kernel. By listening to actions on the KAUTH_SCOPE_VNODE scope, we can detect and deny actions that happen in the Dropbox folder. In the examples cited above, for example, we are interested in the KAUTH_VNODE_DELETE and KAUTH_VNODE_ADD_FILE actions since they allow us to check whether a file or folder in a user's shared folder is being deleted or moved. From there, it's just a matter of checking with the user whether the operation was in fact intended and inform them of the consequences of the operations for other members of the folder. As you can see below, this solution is much simpler than a FUSE implementation would have been, and involves no third-party dependencies.



So if you're someone who compulsively monitors the list of loaded kernel extensions on your system (there are dozens of us, dozens!) and you see `com.getdropbox.dropbox.kext` you now know why!

Stay tuned for more about Project Infinite as we continue to test and ultimately roll it out to a broader set of users.

UPDATE (5/25/15):

We wanted to address some comments about Project Infinite and the kernel. It's important to understand that many pieces of everyday software load components in the kernel, from simple device drivers for your mouse to highly complex anti-virus programs. We approach the kernel with extreme caution and respect. Because the kernel connects applications to the physical memory, CPU, and external devices, any bug introduced to the kernel can adversely affect the whole machine. We've been running this kernel extension internally at Dropbox for almost a year and have battle-tested its stability and integrity.

File systems exist in the kernel, so if you are going to extend the file system itself, you need to interface with the kernel. In order to innovate on the user's experience of the file system, as we are with Project Infinite, we need to catch file operation events on Dropbox files before other applications try to act on those files. After careful design and consideration, we

concluded that this kernel extension is the smallest and therefore most secure surface through which we can deliver Project Infinite. By focusing exclusively on Dropbox file actions in the kernel, we can ensure the best combination of privacy and usability.

We understand the concerns around this type of implementation, and our solution takes into consideration the security and stability of our users' experience, while providing what we believe will be a really useful feature.

// Copy link

Link copied