# Embassies: Radically Refactoring the Web

Jon Howell, Bryan Parno, John R. Douceur, *Microsoft Research*
{*howell,parno,johndo*}*@microsoft.com*

This work is accepted for presentation at NSDI; we propose to present a poster and demo of the system. None of the authors are students.

A defining feature of the web application model is its ostensibly strong notion of isolation. On the desktop, a user must accept responsibility for installing apps, and if an app misbehaves, the consequences are unbounded. On the web, if the user clicks on a link and doesn't like what she sees, she clicks the 'close' button, and web app isolation promises that the closed app has no lasting effect on the user's experience.

Sadly, the promise of isolation is routinely broken, and so in practice, we caution users to avoid clicking on "dangerous links". Isolation fails because the web's API, responsible for application isolation, has simultaneously pursued application richness, accreting HTTP, MIME, HTML, DOM, CSS, JavaScript, JPG, PNG, Java, Flash, Silverlight, SVG, Canvas, and more. This richness introduces so much complexity that any precise specification of the web API is virtually impossible. Yet we can't hope for correct application isolation until we can specify the API's semantics. Thus, the current web API is a battle between isolation and richness, and isolation is losing.

The same battle was fought—and lost—on the desktop. The initially-simple conventional OS evolved into a rich, complex desktop API, an unmanageable disaster of complexity. Is there hope? Or do isolation (via simple specification) and richness inevitably conflict?

There is, in fact, a context in which mutually-untrusting participants interact in near-perfect autonomy, maintaining arbitrarily strong isolation in the face of evolving complexity. On the Internet, application providers, or *vendors*, run server-side applications over which they exercise total control, from the app down to the network stack, firewall, and OS. Even when vendors are tenants of a shared datacenter, each tenant autonomously controls its software stack down to the machine code, and each tenant is accessible only via IP. The strong isolation among virtualized Infrastructure-as-a-Service datacenter tenants derives not from physical separation but from the simplicity of the execution interface.

This paper extends the semantics of datacenter relationships to the client's web experience. Suspending disbelief momentarily, suppose every client had ubiquitous high-performance Internet connectivity. In such a world, exploiting datacenter semantics is easy: The client is merely a *screencast* (VNC) viewer; every app runs on its vendor's servers and streams a video of its display to the client. The client bears only a few responsibilities, primarily around providing a *trusted path*, i.e., enabling the user to select which vendor to interact with and providing user input authenticity and privacy.

We can restore reality by moving the vendors' code down to the client, with the client acting as a notional *pico-datacenter*. On the client, apps enjoy fast, reliable access to the display, but the semantics of isolation remain identical to the server model: Each vendor has autonomous control over its software stack, and each vendor interacts with other vendors (remote *and* local) only through opt-in network protocols.

The pico-datacenter abstraction offers an escape from the battle between isolation and richness, by deconflating the goals into two levels of interface. The client implements the *client execution interface* (CEI), which is dedicated to isolating applications and defines how a vendor's bag of bits is interpreted by the client. Different vendors may employ, inside their isolated containers, different *developer programming interfaces* (DPIs). Today's web API is stuck in a painful battle because it conflates these goals into a single interface: The API is simultaneously a collection of rich, expressive DPI functions, and also a CEI that separates vendors. The conflated result is a poor CEI: neither simple nor well-defined. Indeed, this conflation explains why it took a decade to prevent text coloring from leaking privacy information, and why today's web allows cross-site fetches of JPGs or JavaScript but not XML. The semantics of web app isolation wind through a teetering stack of rich software layers.

We deconflate the CEI and DPI by following the pico-datacenter analogy, arriving at a concrete client architecture called Embassies. We pare the web CEI down to isolated native code picoprocesses, IP for communication beyond the process, and minimal low-level UI primitives to support the new display responsibilities identified above. The rich DPI, on the other hand, becomes part of the web app itself, giving developers unparalleled freedom.