Technology | DOI:10.1145/1538788.1538795

Kirk L. Kroeker

Toward Native Web Execution

Several software projects are narrowing the performance gap between browser-based applications and their desktop counterparts. In the process, they're creating new ways to improve the security of Web-based computing.

MOST INTERNET USERS do not expect the performance of browser-based applications to be the same as desktop applications, which are driven by code created from high-quality compilers and designed to run natively at the operating system (OS) level. However, several ongoing projects at Google, Microsoft, and other companies aim not only to close that performance gap, but also to eliminate some of the security weaknesses that have plagued Web browsers since the early days of the Internet.

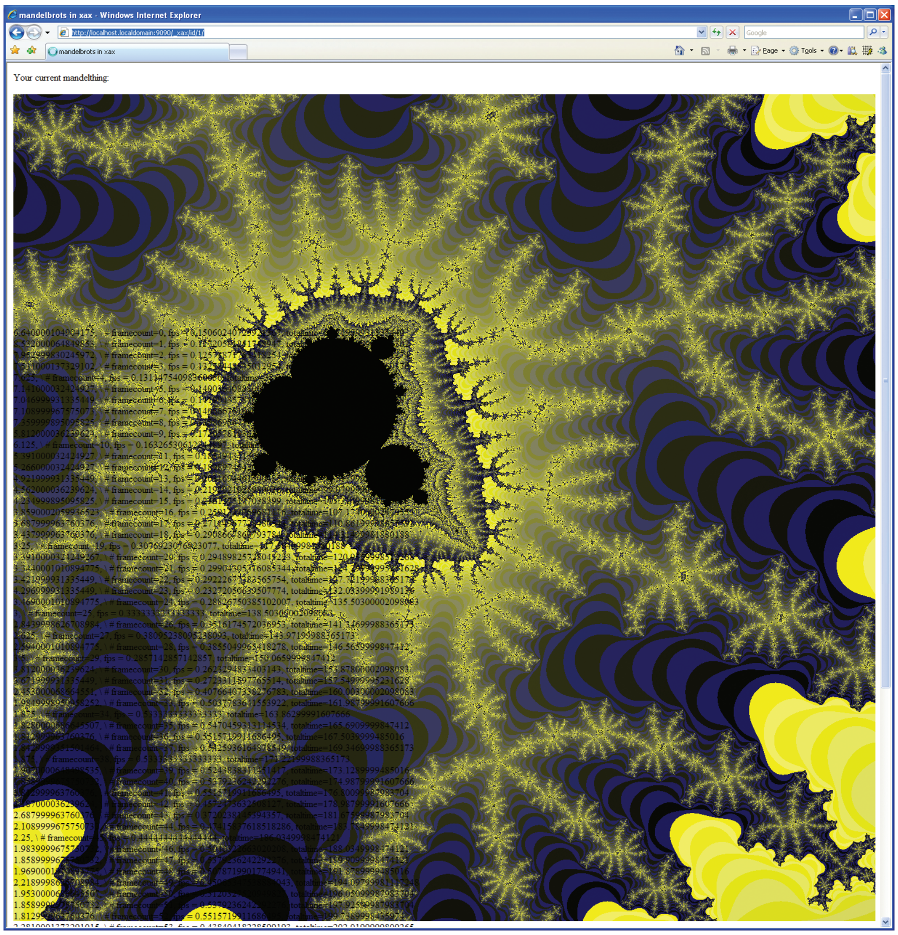
For years, the Netscape plug-in API and Microsoft's ActiveX have provided a way to use native code modules as part of a Web application. Along with enhanced browser functionality, these extension technologies provide full access to the OS's file and networking interfaces. But by relying on trust rather than strong technical measures for safety, these extension technologies are vulnerable to social-engineering attacks in which users are tricked into permitting malicious operations.

One software project that challenges this trust model yet still offers native performance is Xax, developed at Microsoft Research. Xax separates native instruction execution from native OS access, leveraging legacy code to deliver desktop applications on the Web. The project's goal is to incorporate legacy code into browser-based applications, which then run at native performance levels and rely on a security mechanism that is more flexible than language isolation.

"Rather than use a language-based isolation mechanism, why not instead use the well-evolved and ubiquitous memory management unit?" asks researcher Jon Howell, who developed Xax at Microsoft Research.

Howell and his colleagues at Microsoft Research are currently exploring how a Xax interface can be used to deliver not just Web extensions, but all of a Web application's content, including the rendering functions normally provided by a browser. Realigning the client's role in this way, according to Howell, will help make browsers more secure and lead to more flexible applications that can use new rendering frameworks without forcing developers to wait for widespread client adoption.

In theory, it is possible to deliver a new codec or a variant of an HTML renderer in Flash or JavaScript. However, the new code, including all of its libraries, would need to compile to the special language and tolerate performance penalties. "Being able to deliver native code to the client loosens the constraints," says Howell.


mandelbrots in xax - Windows Internet Explorer
http://localhost.localdomain:9090/_xax/ld/1/
mandelbrots in xax
Your current mandelthing:

Xax running a Mandelbrot set explorer to measure performance overhead. This CPU-bound benchmark runs as quickly inside the Xax container as when hosted in a native OS process, nearly 30 times faster than the fastest JavaScript implementations, according to Microsoft.

Different Approaches

In contrast to Xax, which relies on the memory management unit for memory isolation and a kernel system-call patch to prevent OS access, Google's Native Client takes a different approach. Using an OS-portable sandbox, Native Client relies on x86 segmentation hardware to enforce memory isolation and on a binary validator to isolate the OS interface, preventing direct access to the OS and resources such as the file system and the network.

Despite the different implementation techniques, the idea behind Xax and Native Client is similar, according to Howell. "Let the software use the processor however it likes," he says, "and rely for isolation on a simple bit of hardware designed to do just that."

Xax and Native Client are but two of the software technologies designed to close the performance gap and strengthen the security of Web browsers. Sun's Java, Microsoft's Silverlight, and Adobe's AIR represent another approach to isolating untrusted modules from OS interfaces while narrowing the performance gap with native execution. Of course, unlike Xax and Native Client, these application frameworks tend to be used mainly as replacements for the browser-based application environment.

Another alternative approach that is gaining popularity is full virtualization. Systems such as Xen or VMware aren't commonly used to deploy Web-based applications, but that might change soon. Because virtualization systems use code-distribution formats based on native code, they avoid the performance obstacles of JavaScript and other similar languages. And to protect native OS interfaces, they wrap untrusted code in an entire instance of the OS and run that on top of simulated hardware.

"The desire is to have some kind of strong isolation barrier that an attack will not be able to penetrate," says Mendel Rosenblum, cofounder of VMware and a computer science professor at Stanford University. "Hardware-level virtual machines provide precisely that high-assurance barrier yet can run existing browsers at near-native speeds."

Rosenblum says the computer industry's focus on low-level isolation mechanisms is missing the larger point about what virtualization layers can do for performance and security, especially as the Web evolves from a document-delivery mechanism into an ecosystem of interactive applications. "The ability to run sophisticated code safely, and with high performance on the clients, will allow the new applications running in the cloud to support the richer, highly interactive interfaces users are accustomed to," he says.

Full virtualization is an alternative approach that is gaining popularity.

In the meantime, despite the proliferation of technologies that aim to sidestep the performance issues associated with running single-threaded scripts in browsers, JavaScript remains indisputably popular among developers as the only viable choice for programming browsers today. While most believe it is unlikely that JavaScript performance will catch up to the speed of native code execution, both Firefox's TraceMonkey and Google's V8, the JavaScript rendering engine in the Chrome browser, have received industrywide praise for narrowing the performance gap.

"One thing we should never lose sight of is the fact that language virtual machines are not all about straight-line speed of code and that there are many moving parts in the system that need to be balanced against each other," says Ivan Posva, a Google software engineer who developed the V8 JavaScript implementation for Chrome. Still, he says, V8 has narrowed the gap.

In terms of the next speed increase that users can expect from JavaScript rendering engines, Posva says he remains skeptical about the ability of application-specific or language-specific hardware to offer significant improvement. "Currently in V8 there are still many more optimizations that can be applied on general-purpose CPUs," he says. "I do not think that JavaScript-oriented hardware support would be a silver bullet."

In addition to the performance issue, there remains the matter of security. JavaScript running in a browser opens up the possibility for local security attacks in which a malicious application tries to elevate its privileges. "Browser designers need to be aware that the more control we give the third-party programmers via JavaScript, the more control somebody malicious could potentially have," Posva says. "This is not a security issue on its own, but there is a lot more potential control in modern, high-performance virtual machines that can be used to exploit an independent security bug."

To mitigate these risks, V8 uses a layered approach with a sandboxed renderer. "V8 tries to minimize the attack surface by not giving total control over the generated code for a piece of JavaScript and by following common practices such as marking all data non-executable," says Posva. "V8 has to ensure that the policies set by the binding layer are followed properly."

Posva says the performance of V8 will improve regardless of whether it is embedded in a sandboxed environment. "We had to make some design decisions in V8 to allow it being embedded in the sandboxed renderer process within Google Chrome," he says. "But none of these decisions prevent a nonsandboxed use of V8, and none of these decisions had an impact on the real-world performance of V8."

That performance versatility might become increasingly important as browsers evolve, perhaps even to the point where they are no longer distinguishable from the applications they run. "In a few years," says Microsoft's Howell, "I don't think we'll mean the same thing by 'browser' that we mean today; we'll mean much less." Howell predicts that most of the functions of the traditional browser will be rendered moot, replaced by flexible code linked directly into the Web sites users visit.

Howell's prediction amounts to saying that the browser itself will become the sandbox, more or less a simple isolation framework. "Because Xax has such a narrow interface, and because we can compile the browser itself for the Xax container, you can think of Xax as a way to virtualize the browser," says Howell, who maintains that treating the host OS as something special is a short-lived phenomenon.

"As Web applications get richer, they're just as important to protect as the host OS," he says. "If Web applications are sandboxed, users can try one with no risk of exposing everything on their computer."

Kirk L. Kroeker works in communications and has written extensively about the impact of emerging technologies.