A Secure Web Browser Without Isolation Kernels

Eugene Vasserman and Keith McVey
Kansas State University

Abstract (poster)

Web browsers have come to embody *the* interface to the Internet, and securing those interactions is more important than ever. Highly secure browsers have generally taken one of two approaches: use advanced operating system security features (e.g. Chromium and SELinux) or design an isolation kernel below the various browser components (e.g. OP). We present the design of a secure web browser, enforcing strong website isolation and cross-site scripting protection while also protecting the user in case of total browser compromise, all without using only light-weight OS-provided mechanisms.

Our "ultra-thin" browser design delegates security enforcement to the underlying operating system, using existing security mechanisms like user and process isolation, fine-grained object access permissions, and resource limits to prevent insecure interaction between different websites and between websites and the user's system. We treat individual websites as "pseudo-users" of the operating system, with a subset of regular user permissions. Each website's browser instance runs in a user context created only for that website (connected via a persistent socket to that website only), so even bugs in the browser will not allow malicious websites to take control of the user's account or the underlying system, nor connect to third-party sites unless explicitly permitted. The proposed design offers many benefits over current browsers, including increased performance through lower security overhead, support for multiple client-side languages (not just JavaScript), a richer client-side storage infrastructure compatible with HTML5, and even the ability to run native code on the client (without static analysis), making full use of local features such as 3D acceleration (without the need for WebGL).

Our browser is made up of three logical components: a rendering engine to display test and graphics in a browser-like window, a language support module to execute untrusted code downloaded from websites, and a client-side storage modules to store website state (e.g. cookies). Each presents unique challenges in terms of design, implementation, and backward compatibility. The renderer must display websites in separate processes, but modern web pages are composed of many elements, some from different domains than the one hosting the page. For backward compatibility, all these elements *must* be displayed in the same window, but for security, they *should* run in different memory spaces. The language support module (LSM) accepts untrusted website code and runs it on the local machine, inheriting the permissions of the sending website's pseudo-user. It must allow interaction between downloaded code and the displayed browser window, since a large number of websites modify their display elements dynamically, so the rendering engine exports hooks for inter-process communication-based interaction with the webpage DOM. Based on the OS process and user model, untrusted code is already prevented from interacting with other processes' memory or files, and cannot access the network except to communicate with the sending site only, through the use of socket and file descriptor limits.