On the Design of a Web Browser: Lessons learned from Operating Systems

Kapil Singh Wenke Lee

College of Computing, Georgia Institute of Technology, Atlanta, USA
E-mail: {ksingh, wenke}@cc.gatech.edu

Abstract

The advent of "Web 2.0" applications has changed the requirements for a web browser: it has evolved from being an application for rendering static web pages to a host of a variety of applications, where each web page represents a distinct application, such as a news feed, an email client or a video application. This new role has led to the emergence of new security holes in the web browser. While research has focused on fixing these vulnerabilities by augmenting solutions to the current browsers, limited work has been done in designing a browser from scratch considering the current and future role of the browser.

The web browser is quickly evolving as a mini-OS running a huge variety of application code. Going by this thinking, we analyze the functional similarities between an operating system and a web browser and propose a new browser design based on a typical μ-kernel based OS. Our design is flexible and allows finer customization of the browser. The simplicity of the design shows promise in solving most of the security issues prevalent in the current generation of browsers; the feasibility of the design in terms of performance and acceptability is being studied along with the browser implementation.

1 Introduction

The web browser has become the centerfold application to the Internet. It has evolved from being an application for rendering static web pages to a host of a variety of applications, where each web page represents a distinct application, such as a news feed, an email client or a video application. The concept of mashups has taken this requirement to the next level with these different applications being aggregated into a single web page.

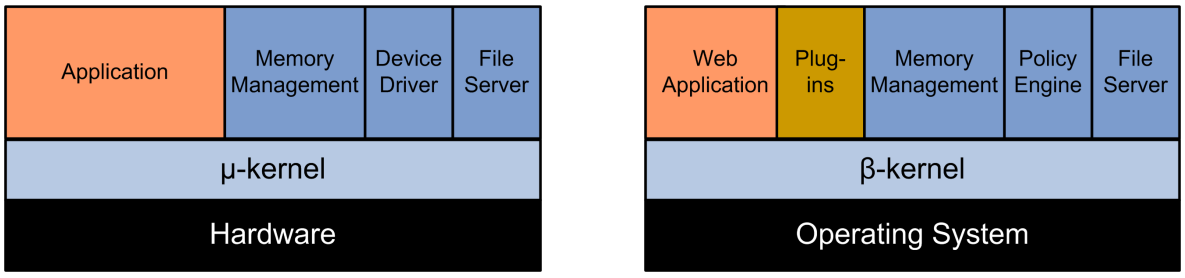
The growing requirements for a web browser has lead to code additions, without much change in its base design. The latest release of Firefox contains over 3.7 million lines of code [11]. The growing size of the browser code and the reluctance to change the design of the browser has left the browser open to a number of exploits. According to a recent report [12], Symantec documented 93 vulnerabilities in Internet Explorer, 74 in Mozilla browsers, 29 in Safari, and 11 in Opera. In addition to these browser vulnerabilities, Symantec also documented 301 bugs in browser plug-ins over the same period of time.

The monolithic model for the web browser provides little security and isolation to the distinct web applications. Once a part of the browser is exploited, it results in the total compromise of the applications and plug-ins running in the browser. In addition, crash of one component of the browser results in crashing of the complete browser.

Research has tried to fix the security problems related to the web browsers as they evolve. The most commonly used security policy for web browsers – the same origin policy – is found to be too restrictive for Web 2.0 applications. As a result, the application authors have developed ad-hoc solutions to work around the same origin policy, leaving the applications vulnerable to attacks. Other solutions have focused on fixing specific issues with the browser such as browser cache [7], plug-ins [11]; or to improve on the same origin policy [8].

With the ever increasing demand and development of new web applications, technologies such as AJAX and mashups have become more prominent in the new Web 2.0 revolution. The web browser is running more code now and has effectively evolved from a standalone application to a mini operating system (mini-OS) running a variety of web applications. Some efforts have been made to design next generation of browsers: while Tahoma [3] uses Virtual Machine Monitors (VMMs) to provide isolation to web applications, OP web browser [6] provides isolation by running each web application as a different process. MashupOS [13] provides new abstractions to enable secure communication among multiple principles in current browsers.

The paper analyzes requirements of an operating system (OS) design from the prospect of designing a new web browser. We study the characteristics of an μ-kernel based OS to determine how they can be perceived in a web browser, and propose a new browser design that attempts to overcome the shortcomings of current web browsers. Our design is flexible to include customized policies and provides mediation between several distinct



(a) μ-kernel architecture.

(b) β-kernel architecture.

Figure 1: Correspondence between μ-kernel based OS and β-kernel based browser.

and isolated browser components.

2 Design Principles

With the growing exposure of web browser to a variety of applications and the increasing number of new attacks against these web applications has highlighted security as one of the foremost parameters in next-generation browser design. We adhere to the following properties that should be desirable from a web browser design:

- **Principle 1: Isolation.** Strong isolation between browser components is required to prevent malicious or unauthorized access between components. It should be possible to implement an arbitrary component in a way that it cannot be disturbed or corrupted by other components. The property has an additional advantage in isolating the fault in a particular component from affecting other web components. The isolation can be at the level of web pages or at more fine-grained frame level.
- **Principle 2: Integrity.** A set of components should be able to establish a communication channel among themselves which can neither be corrupted nor eavesdropped.
- **Principle 3: Separation between policy and mechanism.** The user should be given the freedom to choose its own security policies. These policies should be easily defined and the user should be oblivious to the underlying mechanisms that implement the policies.
- **Principle 4: Customization and Flexibility.** The browser should allow customization based on the requirements of individuals or corporations. This customization should not only include addition of specific functionality to the browser via plug-ins, but also cover the interfaces to provide access control between the browser components. For a mashup page, the user of the browser should be able to control the communication between the web pages running on his browser. The browser should provide explicit interfaces for addition of new browser components.

3 Current Browser Design

The design of current web browsers can be compared to the design of a monolithic kernel where every basic system service like memory management, I/O communication, file system etc. runs in kernel space. The current browsers run as single application with minimum or no isolation between various browser components such as rendering engine, script engine, etc. Once the plug-ins are installed, they run as part of the same browser space with easy access to the user's private information and other browser resources [11].

The inclusion of all basic services in the browser has three big drawbacks: increased code size, lack of flexibility and bad maintainability. Bug-fixing or the addition of new features means restarting the whole browser. An issue in one browser component or extension leads to a complete browser crash. Similar to a monolithic kernel, a much larger code base is difficulty to maintain resulting in browser vulnerabilities.

4 Redesigning the Browser

At a high level, there are many similarities in the functional requirements of an OS and a web browser. Figure 1 gives a high level view of these similarities; we leverage these similarities to propose a new browser design.

We propose a layered architecture for the browser with a browser kernel mode and a user mode. The web applications run in the user mode with their communication being mediated by the browser kernel (β-kernel). Since the browser runs as a single process with its own address space, we need flexible and easy-to-use protection mechanism to isolate these protection layers in the same address space. For our design, we develop techniques in the lines of the intra-address space protection

mechanisms based on segmentation and paging hardware for Intel x86 architecture proposed in [2].

Our layered architecture within the same address space allows our browser to provide memory isolation for various browser components as the memory management has a single view of the memory available to the browser. This is an advantage over the process-based approach used by OP web browser [6].

4.1 Browser Kernel

The β-kernel is the central component of the web browser. Its responsibilities include managing the browser's resources (the communication between operating system and browser components). As a basic component of the web browser, a β-kernel provides the lowest-level abstraction layer for the resources (especially memory, processors and I/O devices) that web application must control to perform its function. It will make these facilities available to applications through intra-process communication mechanisms.

There are many OS designs, such as Exokernel [4], SpinOS [1] and Singularity [5], that might provide interesting prospectives in designing a new web browser and need to be evaluated in detail. With the objective of keeping the kernel as small as possible, we designed our β-kernel based on the concepts of a L4 μ-kernel design [10]. In this section, we anticipate the minimum "primitives" that a β-kernel should implement. A primitive is tolerated inside the kernel if moving it outside the kernel would prevent implementation of browser's required functionality.

4.1.1 Address Spaces

The state of the browser has been a target of exploits to track users against their wishes. This tracking is possible because persistent browser state is not properly partitioned on per site-basis. The same origin policy has been refined to address this problem [7], but the solution is not flexible to include any general user-defined policy. Standing by our principles 3 and 4, we include the concept of address spaces in the browser kernel to isolate memory space for different browser components and to implement customized access control policies.

The basic idea is similar to the μ-kernel concept to support recursive construction of address spaces outside the kernel. At browser startup, there is only one address space controlled by the first memory management component. For constructing and maintaining further address spaces, the β-kernel provides three operations:
Grant. The owner of an address space can grant any of its own memory space to another browser component. This operation is used by memory management server to allocate memory to new web application.
Map. Mapping is used to share address space between various components of the browser.
Flush. Flushing operation is used to revoke sharing of an address space.

Moving the address space concept into the β-kernel allows memory management policies to be specified outside the kernel. The model can be easily combined with the access right policies on the browser persistent state. Mapping and granting copy the source page's access right or a subset of them, i.e., can restrict the access but not widen it. Special flushing operations can be used to remove specified access rights.

The complete memory available to the browser is in the control of the memory management module, hence the installation of drive-by plug-in installations can be detected. The browser has installation directories in the file system to install the plug-ins; memory concepts can be extended to the file system to prevent unauthorized plug-in installations: only the file server can *grant* permission to write to the file system and hence all plug-in installations are mediated by the file server in compliance with the security engine policies.

4.1.2 Communication between Browser Components

With the advent of new web applications, there is a growing need for communication across domain boundaries. This communication includes communication between different web domains in a mashup, between different plug-ins, or between web pages and plug-ins.

From a security point of view, this communication needs to be verified and checked for malicious intent. The β-kernel provides the complete mediation and hence assure integrity (Principle 2); the functionality to provide the message passing interface for inter-domain communication resides in the kernel. This communication model is analogous to the IPC in μ-kernel design.

Liedtke et. al also proposed supervised IPC channels or Clans [9] for providing access control for the IPCs between various subsystems. Access control for intra-browser communication can be implemented along the same lines: the policies are specified at the user-level and allows for browser customization (Principle 4).

4.1.3 Identity of Components

The β-kernel assigns identifiers to the browser components for reliable and efficient communication between them. The identifiers should be unique both in time and space. The identifiers are useful to force component-specific policies in the kernel effectively. Since the β-kernel keeps track of the identifiers and provides complete isolation between components, it is difficult for a

malicious component (such as plug-in) to masquerade another component's identity and as a result, support runtime monitoring solutions for plug-ins [11].

4.2 Browser Extensions

A variety of extensions can be developed on top of the β-kernel showing the flexibility of our design. The user has the flexibility to develop their own memory management, access control system, file server, etc. as a layer on top of the kernel.

The browser plug-ins are installed in the user mode on top of the β-kernel as extensions. The installation process is mediated by the kernel based on the user-specified policies for security and memory management. The communication interfaces published by the plug-ins are cross-checked against the user policies before making them available to the other browser components.

4.3 Web applications

The rendering of web pages in a browser is equivalent to running of user applications in an operating system. A web page could be a standalone page running in the browser or could be a component of a mashup page. These web pages communicate with each other similar to the inter-process communication between OS applications. The β-kernel is their gateway to other web-pages and to the resources available in the operating system (Figure 1(b)).

5 Policies and Browser Customization

We propose a layered approach for specifying security policies: policies are defined by the web page authors, by system administrators and by the user of the browser. In case of conflicts, the administrator policies take precedence over user and the web page provider: the user can only make the policies more restrictive. The user policies have higher authority compared to web page author's policies.

This layered policy model allows development of customized browsers with controlled intra-browser communication. For example, the system administrator of a corporation provides a custom-made browser with selected plug-ins, putting restrictions on the visit of certain web-pages, and policies to define what interfaces are available to web pages in order to interact with plug-ins or other web pages. The web page author can specify the interfaces accessible to other web components. The system administrator can only allow access to a subset of these interfaces; the user can, in turn, restrict these policies further.

6 Discussion

We presented a new browser design leveraging the learnings from a μ-kernel design. Our design is flexible providing customized browser policies and mediated intra-browser communication. We believe that the flexibility and simplicity of the design could solve most of the security problems of the current browsers.

Our design shows potential, but whether it is really feasible in practice depends on the achieved performance of the β-kernel. The success of μ-kernel based OS shows promise that such designs are practical; we are currently implementing the design to test its feasibility.

In this paper, we tried to bridge the gap between the web browser and the OS. This might be useful in utilizing the experiences in the field of OS to provide defensive solutions for the web browser. One argument that supports this thinking is that web attacks also show correspondence with the OS level attacks. For example, a typical cross-site scripting (XSS) attack is basically a code injection attack for an operating system, the difference being the injection is done in the network outside the browser domain. Considering correspondences, it would be interesting to see if the host-based solutions can be utilized to develop defenses against the web attacks. Our β-kernel based design provides flexibility to include such future security solutions.

References

[1] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. J. Eggers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, Dec. 1995.

[2] T. Chiueh, G. Venkitachalam, and P. Pradhan. Integrating segmentation and paging protection for safe, efficient and transparent software extensions. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99)*, Dec. 1999.

[3] R. S. Cox, S. D. Gribble, H. M. Levy, and J. G. Hansen. A safety-oriented platform for web applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, Oakland, CA, May 2006.

[4] D. R. Engler, M. F. Kaashoek, and J. J. O'Toole. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, Dec. 1995.

[5] M. Fähndrich, M. Aiken, C. Hawblitzel, O. Hodson, G. C. Hunt, J. R. Larus, and S. Levi. Language support for fast and reliable message-based communication in singularity os. In *Proceedings of EuroSys2006*, Apr. 2006.

[6] C. Grier, S. Tang, and S. T. King. Secure web browsing with the op web browser. In *Proceedings of the*

2008 IEEE Symposium on Security and Privacy, Oakland, CA, May 2008.

[7] C. Jackson, A. Bortz, D. Boneh, and J. C. Mitchell. Protecting browser state from web privacy attacks. In *WWW '06: Proceedings of the 15th International Conference on World Wide Web*, pages 737–744, Edinburgh, Scotland, May 2006.

[8] C. Karlof, U. Shankar, J. D. Tygar, and D. Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *CCS '07: Proceedings of the 14th ACM conference on Computer and Communications Security*, pages 58–71, Alexandria, Virginia, USA, Oct. 2007.

[9] J. Liedtke. Clans & chiefs. In *Architektur von Rechensystemen, 12. GI/ITG-Fachtagung*, pages 294–305, London, UK, 1992. Springer-Verlag.

[10] J. Liedtke. On micro-kernel construction. In *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP)*, pages 237–250, Copper Mountain Resort, Colorado, December 1995.

[11] M. T. Louw, J. S. Lim, and V. N. Venkatakrishnan. Extensible web browser security. In *DIMVA*, pages 1–19, Lucerne, Switzerland, July 2007.

[12] D. Turner. Symantec internet security threat report. Technical report, Symantec, Sept. 2007.

[13] H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and communication abstractions for web browsers in MashupOS. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP'07)*, Stevenson, WA, Oct. 2007.