The Multi-Principal OS Construction of the Gazelle Web Browser by Helen J. Wang, et al. (*USENIX Security Symposium*, 2009)

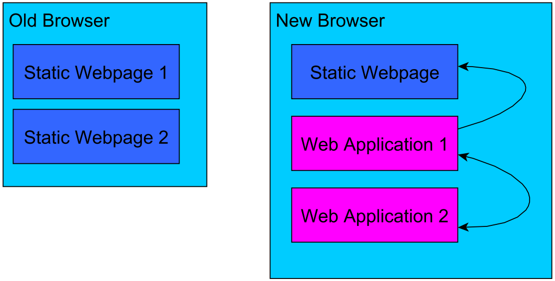
presented by
Jedidiah R. McClurg

Northwestern University

April 16, 2012

Background

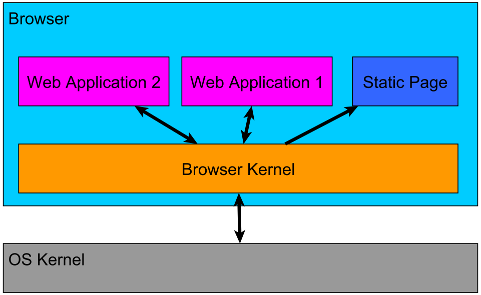
- The nature of the web is changing
  - Originally, web pages featured static content
  - Increasingly, web pages are dynamic *applications*
- Since the *browser* is the environment which loads/executes web pages, it needs to acommodate these changes



- This new browser structure should look familiar...

Motivation

- An *operating system*!
  - Multitasking
  - Inter-process communication
  - Window management



- A browser OS structure has several major advantages
  - Site (process) memory isolation
  - Error recovery
  - Centralized policy enforcement (in the browser kernel)

Motivation (Cont.)

- The Gazelle web browser [1] is based on this *browser OS* approach.
- The browser kernel is the *sole entity* in charge of...
  - Fair sharing of system resources
  - Cross-site resource protection (addressed in this paper)
- This main concern regarding resource protection is the SOP (same-origin policy)
  - An *origin* or (*principal*) is defined as `<protocol, domain-name, port>`
  - Different origins should be in different browser OS "processes"
  - Note that `news.google.com` is a different origin than `google.com`

Related Work

- Unfortunately, the popular browsers don't quite work this way
- Example: the Google Chrome browser
    - Its origin policy is more lax, i.e. an origin is defined in terms of the top-level domain
    - It has a per-site-instance process model, i.e. each site being browsed corresponds to exactly one process, regardless of embedded cross-site `iframes`
    - All plugins (regardless of origin) are in a single process
    - The rendering process (rather than a centralized kernel) is responsible for some of the origin policy enforcement

Related Work (Cont.)

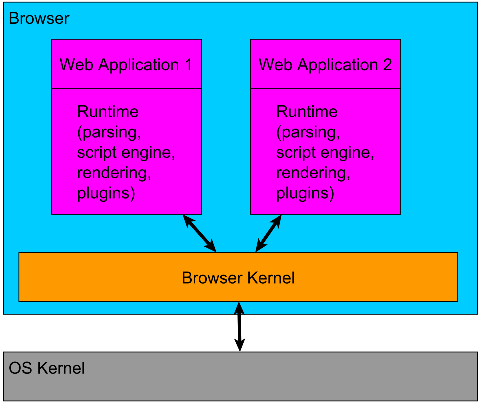
- Example: the Internet Explorer (IE 8) browser
    - Each *tab* corresponds to a different process, even if different sites are browsed in that tab
    - The goal here is not so much security as *error recovery*
- Example: the experimental OP browser
    - Uses a process for each browser component
    - Kernel delegates some of its protection responsibility
    - This doesn't really address the cross-site protection issue

Security Model

- Security models of popular browsers
    - The SOP is usually enforced in regards to scripts
        - `XMLHttpRequest` can only communicate with origin
        - Scripts in an `iframe` cannot access the DOM for cross-origin `iframes`
        - Descendant navigation is a potentially problematic exception (more about this later)
    - Cookies have a path-based security policy, but scripts can supersede this and access all cookies for a given domain
    - Plugins are not usually regulated by any browser security policies
- Gazelle security model
    - Enforce the SOP in all cases (scripts, cookies, plugins)
    - A more flexible definition of “origin” could be adopted (e.g. to include *paths*), but this could cause compatibility problems with existing websites

Gazelle System Architecture

- Web applications communicate with each other (and/or the system) *only* through "system calls" to the browser kernel

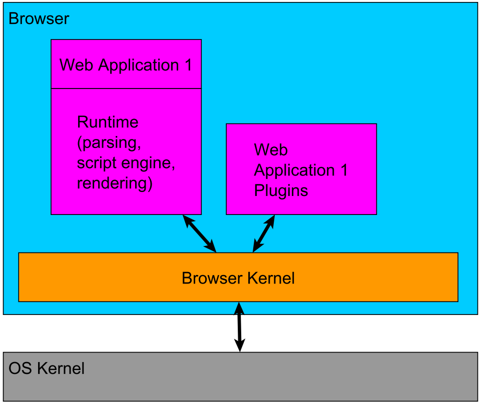

Browser
Web Application 1
Runtime (parsing, script engine, rendering, plugins)
Web Application 2
Runtime (parsing, script engine, rendering, plugins)
Browser Kernel
OS Kernel

Gazelle System Architecture (Cont.)

- This architecture provides a good combination of features...
  - Efficiency – a dedicated runtime instance (with parsing, rendering, etc.) is spawned for each web application, which is faster than approaches having separate modules for these functions (such as the OP browser)
  - Security – since the web applications exist in separate processes owned by the browser kernel, they are prohibited (via the OS kernel's process management) from communicating with each other, except through the browser kernel
  - Robustness – if one web application (or runtime instance) encounters an error, that process can be safely terminated without effecting the rest of the browser

Gazelle System Architecture (Cont.)

- One slight modification, for the sake of legacy script protection...


Browser
Web Application 1
Runtime (parsing, script engine, rendering)
Web Application 1 Plugins
Browser Kernel
OS Kernel

Cross-Origin Protection

- The web application's runtime is responsible for rendering of content, but the browser kernel is responsible for the content positioning
- A *landlord* web application can "rent out" space to a cross-origin *tenant*
- **Display protection** then becomes an important issue
  - Tenant should not be able to interfere with the landlord's display
  - Only the tenant can draw within its rented space
  - Landlord and tenant can write each other's URL location, but only tenant can read

Cross-Origin Protection (Cont.)

- The browser kernel is responsible for intercepting events, processing them, and dispatching them to the correct web application.
- **Event protection** becomes important here, since things like stacked transparent windows can be used to trick the user into interacting with the wrong site
    - Instead of arbitrary z-position for windows, Gazelle only allows for two-dimensional placement (landlords can rent out a part of their space to a tenant)
    - One exception is *opaque* overlays, which are allowed

Security Analysis

- Gazelle addresses cross-origin and memory vulnerabilities
    - This is done via isolation of web applications (via processes)
    - The cited vulnerability study found 6 origin-related errors in IE, and 11 in Firefox
    - It also found 38 memory-related errors in IE, and 25 in Firefox
    - Total of 80 security issues potentially addressed by Gazelle

Security Analysis (Cont.)

- Gazelle addresses display vulnerabilities
  - This is done via allocating exclusive control of window layout to the browser kernel
  - The vulnerability study found 3 display-related errors in IE, and 13 in Firefox
- Gazelle addresses plugin vulnerabilities
  - This is done by placing plugins in the same type of sandbox configuration as the web applications themselves

Implementation

Gazelle prototype runs on Windows Vista with the .NET Framework

- Browser kernel
  - Approximately 5K lines of C# code
  - Communicates with web application instances via XML-based messaging over pipes
  - Web application instance provides rendered bitmaps for each window (generated with the .NET graphics libraries), and the browser kernel composes them with respect to the proper layout

Implementation (Cont.)

- Web application instance
  - Rendering is done via Internet Explorer's Trident renderer component
  - Site events such as frame creation and user events such as mouse clicks are captured and re-routed to the browser kernel
  - Network requests are captured by connecting through a local (virtual) proxy server, which re-routes the requests to the browser kernel

Evaluation

- The prototype Gazelle browser was able to successfully browse 19 of the top 20 Alexa web sites
- Typical page load latency was found to be comparable to that of other browsers

	Task	Gazelle	IE 7	Chrome
1	Browser startup (no page)	668 ms	635 ms	500 ms
2	New tab (blank page)	602 ms	115 ms	230 ms
3	New tab (google.com)	939 ms	499 ms	480 ms
4	Navigate from google.com to google.com/ads	955 ms	1139 ms	1020 ms
5	Navigate to nytimes.com (with a cross-origin frame)	5773 ms	3213 ms	3520 ms

Evaluation (Cont.)

- Gazelle's baseline memory footprint is around 9 MB
- Typical memory usage was found to be comparable to that of other browsers

	Task	Gazelle	IE 7	Chrome
1	Browser startup (no page)	9 MB	14 MB	25 MB
2	New tab (blank page)	14 MB	0.7 MB	1.8 MB
3	New tab (google.com)	16 MB	1.4 MB	7.6 MB
4	Navigate from google.com to google.com/ads	6 MB	3.1 MB	1.4 MB
5	Navigate to nytimes.com (with a cross-origin frame)	88 MB	53 MB	19.4 MB

Evaluation (Cont.)

- Responsiveness is acceptible
  - A user event (e.g. mouse click) has a delay of
    upcall time to web application +
    Trident delay +
    display update if necessary =
    (2 ms + 1 ms + 77 ms) = 80 ms
- Process creation is acceptible
  - The top 100 Alexa sites were visited with Gazelle to obtain statistics regarding process creation
  - The median number of processes was 4, the minimum was 1, and the maximum was 28

Compatibility Issues

- As we have said, many browsers do not implement the strong security policies implemented in Gazelle
- Thus, it is important to see how these strong policies affect everyday browsing
- Again, the top 100 Alexa sites were browsed using Gazelle
- Subdomain behavior
    - Gazelle does not allow pages to change their domain at all, but some sites use subdomain changes (e.g. `news.google.com` to `google.com`) to accomplish cross-origin communication
    - It was found that 6 out of 100 of the sites try to do this

Compatibility Issues (Cont.)

- Mixed HTTPS and HTTP content
    - Allowing HTTP content to be embedded in an HTTPS page is potentially dangerous, so Gazelle doesn't allow it
    - Several popular account-based sites (e.g. Amazon, Blogger, etc.) were accessed and browsed, showing no use of HTTP content within HTTPS
    - A log of 5500 unique SSL URLs were obtained from a user's browsing history, and less than 2% were found to have included non-secure (HTTP) content

Compatibility Issues (Cont.)

- Layout policies
  - Allowing transparent overlays is potentially dangerous, so Gazelle does not allow this behavior
  - Only 2 out of the 100 top Alexa sites were found to attempt this behavior
  - This behavior can be easily avoided by simply making transparent windows opaque
- Plugins
  - In order to work properly with Gazelle, plugins must be changed to use browser kernel system calls
  - Flash is the only plugin used in the top 100 Alexa sites, appearing in 34 of them

Future Work

What things are important in regards to future browser research?

- Straightforward methods for converting plugins to work with the browser system call interface
- Browser implementation which does not rely on expensive Trident/proxy hooks
- Fair resource sharing in the browser OS

Conclusion

- Many security issues can arise from cross-site interactions
- Gazelle is an experimental web browser designed to address these issues
- Gazelle is structured as an OS whose “applications” are the web sites being browsed
- Separating the web applications in this way provides powerful benefits in terms of security and robustness

Thanks!

Helen J. Wang, Chris Grier, Alexander Moshchuk, Samuel T. King, Piali Choudhury, and Herman Venter.
The multi-principal os construction of the gazelle web browser.
In *Proceedings of the 18th conference on USENIX security symposium*, SSYM'09, pages 417–432, Berkeley, CA, USA, 2009. USENIX Association.