CSE 509 Lecture 15

Prof. Don Porter, Scribe:Arun Rathakrishnan

October 24, 2013

1 Introduction

Early browsers had to render only html pages and security was not a major goal for browser developers. Modern browsers support javascript, plugins, CSS, and web services are more powerful. That Chrome OS exposes the OS API through browser explains the evolution of browsers. Websites co-exist in a common browsing environment and browser's responsibility is to protect one site from accessing another site's data as a way of ensuring confidentiality. Browsers must also protect themselves and other web pages displayed from being compromised when displaying a malicious website.

2 Current Secuirty Trends

Browsers like Google Chrome, allows each tab (browsing instance) to execute in its own address space as a separate process. As a result, a malicious web page in a tab, can not bring down other processes. But with the prevalence of mashups where a part of browser estate(in a iframe), can be delegated to a different web site (like the ads area in a page). For example, a google calendar widget may be embedded in a travel planer website. In Chrome, both the entities will share the same process. It is important to ensure that one entity does not access or control the other, for the sake of security. Also, browsers entrust the security of plugin contents to native applications like JVM, Flash Content and PDF readers. This is known to cause several security vulnerabilities.

3 Same Origin Policy

3.1 Origin

A resource is identified by URL or URI. The origin of a resource with respect to Same Origin Policy is represented by the triple {Protocol, Domain, Port}. In a nutshell two resources having the same origin can have access to each other, while all other accesses between resources in general are disallowed.

URL 1	URL 2	Same/Different Origin
http://www.eg.com/dir1/pg2	http://www.eg.com/dir2/pg1	Same Origin
http://www.secure.eg.com/pg	http://www.eg.com/pg	Different Origins (Hostname differs)
http://www.user@eg.com/pg	http://www.eg.com/pg	Same Origin (Hostname is still the same)
http://eg.com	http://www.eg.com	Different Origins (Hostname is different)
http://www.eg.com	http://v2.www.eg.com	Different Origins (Hostname is different)
https://www.eg.com	http://www.eg.com	Different Origins (Protocol is different)
http://www.eg.com:80	http://www.eg.com:81	Different Origins (Port is different)

3.2 Examples

3.3 DOM

Document Object Model provides an API for accessing the browser's internal representation of the web page being displayed. It allows scripts to modify the DOM elements in response to events. Same Origin Policy dictates that a DOM element can be accessed by only a script with the same origin. If not for the policy script from a malicious site can steal data by looking at the DOM of another site. Usually the SOP is implemented by the browser from the origin information of HTML elements and scripts.

The HTML elements in DOM have a heirarchical relationship - an enclosing tag is a parent of all inner tags. Thus the entire document can be modelled as a tree. A mashup page has several elements with different origins in the same tree, where elements with disparate origins must not access each other.

3.4 Cookies

Cookies are used by webservers to track users. Since HTTP is a stateless protocol browser resident cookies play an important role in tracking a user session. A variant of SOP model governs Cookie acces, where origin is identified by path and host name. For example cookies set by http://www.example.com/service can be accessed by any descendant of the path like http://www.example.com/service/first/test while any non-descendant like http://www.example.com/home.html can not access the cookie set by the service URL. Similarly marking a cookie as secure means that only a secure protocol can access its value.

3.5 Exceptions

Some times it makes sense for two different origins resources from the same super-domain to have access to each other. For example, it's reasonable for mail.google.com to access DOM element of ads.google.com and viceversa. The web developers circumvent SOP by setting the same document.domain value for the two pages. In the case of domains hosted as a part of hosting service, elements hacker.domain.com should not accessed by attacker.domain.com.

Javascript source imported through the script tag from different origins, assume the origin of the importer. So a script imported from google.com by mypage.com can modify DOM elements of mypage.com as it has the origin reset to mypage.com. But as result, the script can not access elements from google.com.

3.6 Cross-site Scripting Attack

Cross site scripting attempts to bypass SOP policy, to enable a script from a different domain access DOM elements of a page. For example, in a Youtube comment, a user can leave a malicious script as a comment, that may be wrongly considered a plain text comment. Now the script is listed with other plain text comments to any other visitor. As it now shares the same origin as Youtube, it can access any DOM element in the page.

4 Gazelle

4.1 Process Organization

Every browser instance is in a separate process like Chrome. In addition, within an instance, each origin has a separate process. The cohesive display in a tab is achieved by a browser kernel which interacts with disparate processes (up calls). Plugins in an instance execute as processes separate from browser instances. Thus failures in plugins does not affect the host page or the browser. These processes interact with browser kernel through system calls and can do IPC by following the SOP.

4.2 Landlord Tenant model

Like many Window managers, plugins use the abstraction of bitmaps to display content from a web page in a browser instance. Some portions of a page can be delegated by a landlord to a tenant. When a page displays an advertisement at the side pane, that portion has been delegated to a tenant page. Landlord can not access DOM tree of the tenant or the browing history. But it can change the size, position of the tenant display, replace the URL at the tenant.

4.3 Display Protection

Modern browsers allow landlords to overlay a transparent window over a portion of the browser display. While this has genuine uses, it can be used for click jacking (tricking the user to click a button on the transparent victim page, when the user tries to click a visible button the underneath page). Gazelle prevents click jacking to an extent, by disallowing different origin windows to be transparently overlaid. This is called the opaque overlay policy. Thus a malicious webpage can not transparently draw a victim page from a different origin over it.