Browser Security through Gazelle

Why are browser exploits a problem? None of us want our fb page looking into our boa page. One site accessing another.

But why all this complexity?

Originally browsers were supposed to render static pages, html. Now there is a reasonably argument that browsers are OS. eg. Chrome browser – It’s a strip down version of Linux that only draws a web browser IE Chrome on display. It simply renders the Chrome API. Like if you want to write a simple program, so make it into a webpage. It basically implements an OS on a browser.

The evolution from markup text to the web browser is driven by Web2.0. Browser have a runtime environment, js script, plugins like Flash interpreter, cascading style sheets, mashup page1, etc. So pulling in content from different sites introduces security concerns. Cookies store session information for stateless protocols – http, and programs which have authentication, need to be session oriented.

DOM presents an object oriented abstraction of a webpage. Its like the common APIs, classes, internal representation of major components of a webpage. Every HTML is an object on a DOM tree, like doc, frame, paragraph, etc, are all objects on a DOM tree. A browser basically parses a webpage HTML and create a DOM tree out of it.

So with this in mind we have gone from static content we have gone to fairly rich programmable environment having java, flash, javascript, etc, all running and interpreted in browser add space. But this poses many security concerns.

Security problems

Different tabs shouldn’t be able to access the other by going through the DOM tree. But preventing one app from another is tricky and even though info leaks are subtle issues, if I can read data, I can pass it in some other format without being noticed. So if I have plugin how can I prevent it to look into my home directory for my tax filer document?

There is another concern about native code plugin – if they r malicious or buggy, they can do just about anything and read any content. Adobe flash and java are patched very often. A buddy JRE plugin plugged into a browser can access virtually any content on the browser as well as on the system.

1. Mashups: where contents from different sites onto the same page. Eg. of mashup is yelp which uses google maps.
Another issue is clickjacking. In program model within window we can have different things being drawn and layered on each other. What we see is browser tab, and browser flattens it and renders, it after figuring out the top layer. If I am a malicious Jscript, I can make big mashup page which is transparent over the entire browser and capture all the password entered and mouse clicks, I can pass it to something else. So it's a pretty common thing to worry on a mashup because all clicks on a bank website are being clickjacked by a huge overlapping transparent window on a browser.

Solution

This paper observes the browser as a miniature OS and leverages the properties in this. What works for buggy code is virtual memory protection. Code should be in separate processes and not touch each other, just like processes in OS.

Principle

A protection entity (an entity that has to be protected): a cookie, a username, a window, a display, etc. Eg In Gazelle, stonybrook.edu is principal, google is a principle.

Typical Policies Implementation

1. Same Origin Policy, SOP, is the default policy across all browsers to protect content from one origin from other origin. So if I am an obj from DOM tree of google, I shouldn’t be able to access DOM of boa. But again these are not well defined.

   origin = (protocol, domain, port)

   a. If I am running a script on http://www.example.com/dir/page.html, I can access ..dir2/page2.html
   b. If we http://userpass.www.example.com/dir/page.html, they are from same origin because userpass is just a syntactic sugar.
   d. http://en.example.com : is different hosts and are isolated. This is good for security as we might be just using same host for hosting different things. But there are cases where we wouldn’t like this – like we want all google products to play nicely together.

   In Gazelle, we have restrictive policy where a, b are from different domains.

2. Cookies: they have secure flags

3. Plugins are trusted to implement policies themselves. So by default they are trusted to treat all these objects (cookies, DOM) ethically.

Weakness:

1. Problem with SOP is if we want to protect content from iframe, we have to force it to separate all components.
2. If I am google.com and pull script from foo.com, it runs with in google’s origin and it would be able to alter the DOM.
3. SOP have heavy reliance on DNS. Default dns isn’t most secured. E.g Starbucks wifi throws a popup when you look into your google calendar. So any wifi can have a bugged DNS implementation and can break in. Also this is not browser specific.

4. One can include scripts from other origins which they don’t understand.

5. Cross site scripting attack: SOP is exploited where scripted posted as one origin actually belongs to someone else. If I have static html to include js from another page, it runs as mine. XSS is to trick webpage to run script from another page.

**Chrome’s innovations**

Every tab is a different process, like OS level memory protection. If we get doggy content only that tab dies. JScript running on one tab doesn’t get anything from other tab. But this doesn’t do anything about mashups.

**What Gazelle actually do?**

Gazelle helps this by treating every origin in a different process. Each site from each iframe is a different process. As a result JS script can’t be accidently trick into violating SOP. So how does it display from different process when they are to be displayed onto a same window? It use a browser kernel for that job.

Landlord-tenant abstraction model is used to implement offloading the display from a different process (tab, iframe) has the browser OS (rather the landlord) delegating real estate bitmaps to other origins to display their content into the assigned space.

This basically is like a window manager that can display content from other sources (processes). Say mashup.com is landlord
a.com, b.com are tenants (tenants are black box in mashup.com’s DOM)
So the landlord can
- resize its tenant window sizes,
- delete a tenant,
- change the origin of a tenant from a.com to c.com.
The transparent overlay problem is still a problem with Gazelle?

They only allow opaque overlay. The landlord tenant is fine but they allow overlays because of displaying menus. Landlord can temporarily invade the tenant space. But wont let do anything transparent.

What does Gazelle do about plugins?

Plugins are also made to obey the SOP in that they are forced to have process level instances. But they haven’t explicitly solved it. They haven’t prevented plugins from messing up with the host OS.

**Drawbacks**

Too much overhead.