Today’s lecture is about web security in general and what’s going on with browser security. It is based on the paper: “The Multi-Principal OS Construction of the Gazelle Web Browser”

What problem is gazelle trying to solve – Browser Exploits.

Why so worried about the browser? Two security issues with the Browser:

1. Browser gets tricked into accessing system resources which is not allowed to. (for eg. files in the home directory)
2. Unauthorized data access amongst different origins. For eg. You don’t want FB accessing Bofa cookies

Back in the day (When dinosaurs roamed the earth) browsers handled very simple static HTML Pages. Tasks included formatting HTML, displaying images etc.

These days – Reasonable argument that web browsers are OS.

ChromeOS – It’s a stripped down version of Linux that draws a web browser on the display. Surrendered OS API to Chrome. If you want to run an app – Make a web page.

What are the things that browser have now – JavaScript(which is atrocious!), Flash interpreter, Java, CSS – Dynamically modify content.

Mashup Page: Page that pulls in content from different sources. For eg. Yelp – Google map data, comments from Facebook etc. But we have security implications.

Cookies – Store sessions – Why? – Principles of Distributed computing state: Stateless protocols are required for efficient working. Therefore HTTP is stateless. Not good for authenticating users. Cookies are basically a hack to enable authentication over a stateless protocol.

JavaScript and CSS are programmed on DOM. What is DOM – A common set of API’s (classes) that represent browser’s internal implementations of objects. Can walk through the DOM to get from header to paragraph.

In short, DOM is OO abstract of a WebPage

Embedding Java/Flash objects in a webpage lets you have Binary blobs in browsers address space.
What are the security concerns with browsers?

Preventing one app from getting others’ data is tricky. Info leaks are subtle issues. Preventing info leaks once you have access to the info is difficult.

You don’t want yelp getting your banking information.

Plugin is native code -> which implies it has access to system calls -> which in turn implies that you can read/write file. JavaScript by itself can’t do this. It has to go through interpreter.

A Plugin is like any piece of code running on system.

A Buggy JVM if exploited by a Website can access the whole system. (For eg. Contents of home directory)

Flash/Java flashed on a weekly basis.

Clickjacking attack

You have a bunch of different things drawn on a web page. Some may be on top of each other. If you have intersection between two windows, the click goes to the window on top.

Let’s take the Mashup example again. This mashup website has a number of JavaScripts -> Every JS draws whatever it wants. Some of the object drawn are on top of others -> On Click, the browser’s kernel figures out what comes on top layer. A Malicious JS may make a transparent canvas and access everything Even passwords! This JS sees all keystrokes and mouse clicks. This is the motivation for the problem.

Browsers weren’t designed with security in mind. As they evolved they have gotten better, adding access control in DOM the tree is one example of how they’ve become better.

What this paper observes is that at this point the browser is a miniature OS.

Lessons with OS -> What works with buggy code is hardware memory protection. Lot of people have lost faith that even OS can enforce security policy (15 million lines of code!) -> Therefore use VMs -> smaller hypervisor helps here. Keeps everything in check and is easier to maintain.

Multi principled OS construction. In an OS, the use is the Principal. In Gazelle, different origins are principals -> stonybrook.edu is a principal and google.com is one.

Actual Policies

Origin = <protocol, domain, port>

Policies are highly inconsistent from browser to browser.

SOP -> Applies to browser side languages. If running JavaScript -> Should be able to access elements from DOM tree only from places the script itself came from.
This does not prohibit “a.com” to statically include object from “b.com” through frames.

Starting out from and running a script on -
http://www.example.com/dir/page.html

Can you access DOM tree at

1. http://www.example.com/dir/page2.html -> YES!
2. http://www.example.com/dir2/* -> YES!
3. user.pass@www.example.com -> YES! (Same Origin)
6. Example.com -> Violate SOP (no www)
7. www.example.com:80 -> Depends on browser. Browsers are dumb. Some take the port into account, some don’t
8. Accessing local files -> Behaviour not well defined

As an example, on a hosting service, you do not want scripts from don@homelinux.org accessing objects on Kevin@homelinux.org

This is good! But there is a major hindrance -> drive.google.com, mail.google.com

We can trust google to not bring up problems, but now these cannot cooperate with each other.

But there are programmatic interfaces which let two webpages to talk to each other

Cookies work basically the same way -> Use hostname and path

Cookies have secure flag which specify if they should be given to http or https

Plugins are supposed to enforce these policies themselves. So SOP is very weakly enforced here

If you want to protect iframe from same source -> Find some way of changing hostname for script.

Integrating Scripts from different sources on one page: Script loaded from foo.com on Google’s page runs with google origin -> Can access Google’s DOM

SOP has a very heavy reliance on DNS for assigning origins. Default DNS is not a super secure protocol. Wi-Fi access at Starbucks redirects to a terms and conditions page. You can’t trust the integrity of hostname lookups and yet you entire browser’s security is based on it!
Cross Site Scripting

*Script posted as text*

Leverage exception -> if I’m statically including content in HTML -> runs with origin of page that included it.

CSS attack -> Trick a web page to load content from a place where you didn’t anticipate.

Reflected attack -> Server takes input of some form (Comment etc.). Attacker inserts JavaScript as comment, the website sucks in the info and spits it out when anyone else requests this content which runs with full faith of this hosting website.

The worst part: This doesn’t violate SOP.

What gazelle actually does?

Chrome -> Every tab is a separate process

Component crashes -> only one tab dies, also good for security as JavaScript on one tab can’t get access into the other tab. Doesn’t do anything about Mashup pages.

What does gazelle do differently-> Every origin within the same page is in a separate process!!!

Now we are confident that JavaScript interpreter cannot be tricked into violating SOP

How does everything get together -> Browser Kernel
How does a.com get output to the page? (See figure above)

The only output you get from a.com is a bitmap. The browser kernel takes bitmap as input and composites it on the screen. Window manager does essentially the same thing.

If you click on some pixel, kernel figures out where you clicked and redirects to the process that is responsible for that object.

What is landlord able to do -

1. Landlord can change the size of tenants
2. Can replace / Get rid of the tenant
   (By changing the URL to say c.com)

Transparent Overlay in Gazelle - > Not allowed, only opaque overlays

What functionality do you use - > Menus (google docs similar to word)

How Gazelle handles Plugins

Plugins are subject to SOP. There are multiples instances of the plugin, each in its separate process. The plugin interacts with the Browser Kernel. The problem with this approach is that the plugin has to be modified to support Gazelle. This is a non-goal of the paper as plugin rewrite is required.

Google Native Client

Downloads binaries and runs them in a highly restricted environment.