BROWSER SECURITY

Discussion on one paper: -

- The Multi-Principal OS Construction of the Gazelle Web Browser by Helen J. Wang, Chris Griery, Alexander Moshchukz, Samuel T. Kingy, Piali Choudhury, Herman Venter.

Class discussion involved around Gazelle web browser and web security in general then taking forward about the happenings in the browser field.

Problems that Gazelle addresses, before getting into that let’s see what the actual problem with the browser are: -

1) Nowadays we see browser gets tricked by the malicious web pages to access the things on my system which it shouldn’t.
2) One site accessing the data of another site via communication creating a window for the vulnerability in the browsers. For example, Facebook accessing the details for bank of America for the matter of fact, basically we deal with the communication or sharing of data between the applications.

Why it is that browser does a very bad job in preventing these possible accesses?

- To understand this, let’s go back in time and delineate some principle first and then see the answer to this question.
- Way back in early 90’s when computing was different, what browser used to do at that time? Like how they handled data. For example www? They handled static data. Reading the html, formatting it may be bold, italic etc displaying images on screen and all.
- Now, Web browser have turned into OS itself. This is a reasonable argument. Here, we talked about Chrome OS. What actually is chrome OS?
- It is strip down version of Linux. It has surrendered the entire OS API to chrome and provided a feature like if we have to run program on it, we do it in the form of web pages.
How we have transformed that gif image and static page era to the environment of running web browser as an OS?

Technology driving this case?

- Now, not only static content is handled by the browser, but also the runtime environment for javascript. We also believe that javascript is notorious. It was designed as type safety language or anything of that sort.
- Flash- sort of fancier plugin. Eg: Java.
- Cascading style sheet- that dynamically modify content.
- Mash up page: -When we call the content from the bunch of different resources into a single web page, it is termed as mash up page. eg: yelp website is a day to day life example. As a matter of fact it includes advertisements from another source or the google maps, it shows the mash up contents. Obviously it has security concern.
- Cookies: - Store session information. Because stateless protocol requires them. Cookies are basically the hack to layer on authentication type abstractions that are over otherwise stateless protocol.

So, Javascript and CSS are programmed in such way to provide a DOM elements. Document object model is DOM. We can think of it as common set of API’s or roughly like classes that handles major elements of the browser.

Like an html object handles a DOM tree. We can walk down this tree to get from documents to frame, header to paragraph etc. As a result from the browser perspective, let it implements a javascript interpreter is basically sucking in the web page, turning it into some DOM type object and then running the javascript interpreter to interact with these objects in browser address space.

Let us discuss the Motivation behind the browser security which is a Concern:

- 2 websites can exchange information. For example, yelp and bank of America are open in the same browser but in different browser. So I want to make sure that my yelp javascript doesn’t somehow figure out how to get access to my bank of America account using Dom elements of my Bank tab. So, preventing one app to access the data of the another app. It is a subtle issue.
- Accessing the system is another concern. It is like running some plugin type code like I can download the plugin from adobe, that’s a native code that can access the system calls on my host address, also can
read write the files etc. So, javascript is constrained enough that it can’t go and start driving the files around, doing system calls etc. So if I download some malicious plugin, that plugin can run the native code and act as just like program running on my system. So it is a concern about native code running on my system. It can be either malicious or vulnerable.

- Paper also mentions about the clip jacket. What is it? It is nothing but the programming model within a browser, is kind of like window manager, where we sort of have bunch of different things being drawn and then layered at top of each other.

- It is just like we might have two windows on the desktop, what we see on the screen in is sort of flattening of these two windows (see fig below). And if I click on the intersection, however brief the intersection exist, what happens in desktop environment? Control goes to window on top. Browser have the same idea, if we have mash up of two different things, every javascript is executed and it is displayed on the screen.

So, Concern here is, if we have a malicious javascript, what it can do is make a big clear canvas that goes over everything else, which means we can intercept all the incoming request or keystrokes including the passwords. So first it coming to this canvas, who is reading it first logging it password characters and then sending the next level down. So if we are able to create a whole transparent canvas over the web page, we get to see all key strokes and all. It is a major security issue.
**Paper Observation:**

At this point, it considers browser as miniature OS and we should leverage some design principle of OS and apply them to browser to enforce some security. That’s what gazelle do.

Argument in the paper is, Code should not be touching each other. It should be in different address spaces as it happens in OS process. Multiprincipled OS here means they are talking about different websites for one user. Eg: Gazelle says stonybrook.edu is a principle and google is a principle.

Actual Policy to be enforced:-

- Same Origin Policy is a default policy. Rest of the policy are highly inconsistent from browser to browser.
- SOP: it applies to browser side languages, if I am running javascript in the browser, I should be able to access the elements on the DOM tree that also came from site, where I came from. Eg: If I am script from google, I am not allowed to access the DOM elements from bank of America.
- Let’s see the example from Wikipedia for SOP. For that we consider ORIGIN to be as (Protocol(http vs https), Domain, Port number(443,800)) :-
  
  1) [http://www.example.com/dir/page.html](http://www.example.com/dir/page.html). If I am running script on this site, Can I access the DOM tree elements which came from below link?
  [http://www.example.com/dir/page2.html](http://www.example.com/dir/page2.html) Yes, we can.

  [http://www.example.com/dir2/page2.html](http://www.example.com/dir2/page2.html) Yes again we can.

  [http://user:pass.www.example.com/dir2/page.html](http://user:pass.www.example.com/dir2/page.html) Yes it again is within the same origin as host is still www.example.com and we consider that browser are intelligent enough to gauge that.

  [http://www.example.com:81/dir/page.html](http://www.example.com:81/dir/page.html) as it is running on different port, it is not the same origin policy.

  [http://en.example.com/dir/page.html](http://en.example.com/dir/page.html) Here Domain name is different so not SOP.

  [https://www.example.com/dir/page.html](https://www.example.com/dir/page.html) Different protocol so not SOP.
So, does these restriction make sense? Yes it does make sense, suppose we have different subdomains. Like we have don.homelinux.org and may be kevin.homelinux.org, so we don’t want the javascript from each of the page to hinder in other one’s page because we are using the same hosting service. Obviously NO. So it is a good restriction/Policy.

Now, why it is too restrictive? Let us see the counter example? For example, like google have docs.google.com, calendar.google.com so we trust google more than anything else. So we can assume that they won’t do anything with our credentials. So in case of google we want this to be handle nicely together which it does.

So, instead of www.example.com if I go http://v2.www.example.com, it violates SOP. Even http://v2.www.example.com :80 will consider based on which browser we are running. Whether it consider it violation or not.

- Cookies Work pretty much the same way. It’s only that they don’t have the protocol and they use the same host name as path.
- Other thing to consider is plugins are considered to enforce these policy themselves.

**High Level Policy: -**

- Weakness policy.
- Note: if we want to save same origin policy for iframes, we would have to make a different level of access control for them.
- Same origin policy has a heavy reliance on DNS. It is a different thing altogether. Hijacking of DNS request can be done.
- Another weakness for this policy is that we can still include scripts from the site which we do not understand. This bring us to the attack called Cross site scripting attack.
- What is XSS? -> Script stored in database of authentic website, but when rendered to the user, it would act as a script and it would then send the malicious content. XSS is a attack in which you trick a web page in loading a content that includes a script from other place where you wouldn’t expect it to be. It is a kind of very tricky as it is within the same origin policy but its obviously something bad.

What Chrome had that other browser didn’t have?

**Every tab has different process altogether.** So, if one of the tab is handling the dodgy content and it crashes then only that particular tab would crash. From security perspective it is very nice.
In chrome we haven’t eradicated the entire problem as in whole. So WHAT DOES GAZELLE DO IN ORDER TO TAKE THIS IDEA TO LOGICAL CONCLUSION?

So, in this every origin gives its own process. Let us understand it using diagram:-

They talk about landlord and tenant model which we see below.

1) If I am mash up.com, I don’t know what’s going on in tenant. It is a black box. What we can do is change the surface or make a layer as shown in blue above. Landlord can make the tenant smaller or bigger depending on how it fits.

2) We can change the URL of the tenant in order to have some privilege.

3) Some idea of IPC mechanism is also provided but it is not done in general in the paper and we don’t think it is needed other than communication.

4) Transparent overlay problem. I am mash up and I am attacker and creates an overlay, transparent box on a.com above in light blue. So in paper they state that they only allow OPAQUE OVERLAY.

5) So, why do they allow overlay at all? Model breaks down when there would be menu, which will pop down. Eg: google docs behave same as word it’s just it is browser. So everyone wants that, so we can’t keep that functionality away. So, we don’t let them do anything transparent as it might affect security in the end.
What does gazelle do regarding plugins?

- Plugins are subjected to same origin policy but basically, user runs plugin they run it in different class altogether. It’s just that we need to run the multiple instance of this plugin. So efficiency comes into the picture, but security is the major concern which is taken care of.
- Gazelle need to re write the plugins, so they do not solve the problem regarding this.
- They are not trying to protect the host kernel from malicious plugin.

Class Concluded.