**Day 2: Cross Window Communication**

Two URLs are said to have the "same origin" if they share the following three components:

**1. Protocol**: The protocol part of the URL specifies how the browser should communicate with the server. Common protocols include HTTP, HTTPS, FTP, and others. URLs with the same protocol are considered to have the same origin.

2. **Domain:** The domain, also known as the hostname, represents the web server's address. It is the human-readable name used to identify a specific server on the Internet. URLs with the same domain are considered to have the same origin. For example, "example.com" and "subdomain.example.com" have the same origin if the protocol and port also match.

3. **Port:** The port number is a numerical identifier that specifies a specific endpoint on the server. If a port is not explicitly specified in the URL, the default port for the given protocol is assumed (e.g., 80 for HTTP, 443 for HTTPS). URLs with the same protocol and domain (including subdomains) but different port numbers are considered to have different origins. For example, "example.com:8080" and "example.com" have different origins.

The same-origin policy is a security feature implemented by web browsers to restrict web pages from making requests to a different origin for security reasons. This policy helps prevent cross-site request forgery (CSRF) attacks, cross-site scripting (XSS) attacks, and other security vulnerabilities.

Web pages from one origin are generally allowed to interact with each other through JavaScript and access each other's resources (e.g., read cookies or make AJAX requests) if they share the same origin. However, pages from different origins are subject to security restrictions, and JavaScript running on one page typically cannot access or manipulate the content of another page from a different origin.

Understanding the same-origin policy is essential for web developers to ensure the security and proper functioning of web applications, as it governs how different web resources can interact with each other in the context of a web browser.

**In Action: iFrame**

An <iframe> tag hosts a separate embedded window, with its own separate document and window objects.

We can access them using properties:

* + iframe.contentWindow to get the window inside the <iframe>.
  + iframe.contentDocument to get the document inside the <iframe>, iframe.contentWindow.document.

When we access something inside the embedded window, the browser checks if the iframe has the same origin. If that’s not so then the access is denied (writing to location is an exception, it’s still permitted).

**Windows on Subdomains:**

The **document.domain** property is a way to relax the same-origin policy when working with subdomains within the same parent domain. It allows web pages served from different subdomains to communicate with each other using JavaScript, even though they would normally be subject to the same-origin policy's restrictions.

But if windows share the same second-level domain, for instance, john.site.com, peter.site.com, and site.com (so that their common second-level domain is site.com), we can make the browser ignore that difference, so that they can be treated as coming from the “same origin” for the purposes of cross-window communication.

To make it work, each such window should run the code:

document.domain = 'site.com';

**Collection: Window.frames**

An alternative way to get a window object for <iframe>– is to get it from the named collectionwindow.frames:

* By number: window.frames[0] – the window object for the first frame in the document.
* By name: window.frames.iframeName – the window object for the frame withname="iframeName".

An iframe may have other iframes inside. The corresponding window objects form a hierarchy.

Navigation links are:

* + window.frames – the collection of “children” windows (for nested frames).
  + window.parent – the reference to the “parent” (outer) window.
  + window.top – the reference to the topmost parent window.

**The “sandbox” iframe attribute**

The sandbox attribute allows for the exclusion of certain actions inside an <iframe> to prevent it from executing untrusted code. It “sandboxes” the iframe by treating it as coming from another origin and/or applying other limitations.

There’s a “default set” of restrictions applied for <iframe sandbox src="...">. But it can be relaxed if we provide a space-separated list of restrictions that should not be applied as a value of the attribute, like this: <iframe sandbox="allow-forms allow-popups">.

In other words, an empty "sandbox" attribute puts the strictest limitations possible, but we can put a space-delimited list of those that we want to lift.

Here’s a list of limitations:

**allow-same-origin**

By default "sandbox" forces the “different origin” policy for the iframe. In other words, it makes the browser to treat the iframe as coming from another origin, even if its src points to the same site. With all implied restrictions for scripts. This option removes that feature.

**allow-top-navigation**

Allows the iframe to change parent.location.

**allow-forms**

Allows to submit forms from iframe.

**allow-scripts**

Allows to run scripts from the iframe.

**allow-popups**

**Open Window Messaging**

Allows to window.open popups from the iframe.

The postMessage interface allows windows to talk to each other no matter which origin they are from.

So, it’s a way around the “Same Origin” policy. It allows a window from john-smith.com to talk to gmail.comand exchange information, but only if they both agree and call corresponding JavaScript functions. That makes it safe for users.

The interface has two parts.

**postMessage**

The window that wants to send a message calls postMessage method of the receiving window. In other words, if we want to send the message to win, we should call win.postMessage(data, targetOrigin).

**Arguments:**

**data**

The data to send. Can be any object, the data is cloned using the “structured cloning algorithm”. IE supports only strings, so we should JSON.stringify complex objects to support that browser.

**targetOrigin**

Specifies the origin for the target window, so that only a window from the given origin will get the message.

The targetOrigin is a safety measure. Remember, if the target window comes from another origin, we can’t read it’s location in the sender window. So we can’t be sure which site is open in the intended window right now: the user could navigate away, and the sender window has no idea about it.

Specifying targetOrigin ensures that the window only receives the data if it’s still at the right site. Important when the data is sensitive.