Same-origin policy

The **same-origin policy** is a critical security mechanism that restricts how a document or script loaded by one origin can interact with a resource from another origin.

It helps isolate potentially malicious documents, reducing possible attack vectors. For example, it prevents a malicious website on the Internet from running JS in a browser to read data from a third-party webmail service (which the user is signed into) or a company intranet (which is protected from direct access by the attacker by not having a public IP address) and relaying that data to the attacker.

Definition of an origin

Two URLs have the *same origin* if the protocol, port (if specified), and host are the same for both. You may see this referenced as the "scheme/host/port tuple", or just "tuple". (A "tuple" is a set of items that together comprise a whole — a generic form for double/triple/quadruple/quintuple/etc.)

The following table gives examples of origin comparisons with the URL `http://store.company.com/dir/page.html`:

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

Inherited origins

Scripts executed from pages with an `about:blank` or `javascript:` URL inherit the origin of the document containing that URL, since these types of URLs do not contain information about an origin server.

For example, `about:blank` is often used as a URL of new, empty popup windows into which the parent script writes content (e.g. via the `Window.open()` mechanism). If this popup also contains JavaScript, that script would inherit the same origin as the script that created it.

`data:` URLs get a new, empty, security context.

Exceptions in Internet Explorer

Internet Explorer has two major exceptions to the same-origin policy:

Trust Zones

If both domains are in the *highly trusted zone* (e.g. corporate intranet domains), then the same-origin limitations are not applied.

Port

IE doesn't include port into same-origin checks. Therefore, `https://company.com:81/index.html` and `https://company.com/index.html` are considered the same origin and no restrictions are applied.

These exceptions are nonstandard and unsupported in any other browser.

File origins

Modern browsers usually treat the origin of files loaded using the `file:///` schema as *opaque origins*. What this means is that if a file includes other files from the same folder (say), they are not assumed to come from the same origin, and may trigger CORS errors.

Note that the URL specification states that the origin of files is implementation-dependent, and some browsers may treat files in the same directory or subdirectory as same-origin even though this has security implications .

Changing origin

> **Warning:** The approach described here (using the `document.domain` setter) is deprecated because it undermines the security protections provided by the same origin policy, and complicates the origin model in browsers, leading to interoperability problems and security bugs.

A page may change its own origin, with some limitations. A script can set the value of `document.domain` to its current domain or a superdomain of its current domain. If set to a superdomain of the current domain, the shorter superdomain is used for same-origin checks.

For example, assume a script from the document at `http://store.company.com/dir/other.html` executes the following:

```
document.domain = "company.com";
```

Afterward, the page can pass the same-origin check with `http://company.com/dir/page.html` (assuming `http://company.com/dir/page.html` sets its `document.domain` to "`company.com`" to indicate that it wishes to allow that - see `document.domain` for more). However, `company.com` could **not** set `document.domain` to `othercompany.com`, since that is not a superdomain of `company.com`.

The port number is checked separately by the browser. Any call to `document.domain`, including `document.domain = document.domain`, causes the port number to be overwritten with `null`. Therefore, one **cannot** make `company.com:8080` talk to `company.com` by only setting `document.domain = "company.com"` in the first. It has to be set in both so their port numbers are both `null`.

The mechanism has some limitations. For example, it will throw a "`SecurityError`" `DOMException` if the `document-domain` `Feature-Policy` is enabled or the document is in a sandboxed `<iframe>`, and changing the origin in this way does not affect the origin checks used by many Web APIs (e.g. `localStorage`, `indexedDB`, `BroadcastChannel`, `SharedWorker`). A more exhaustive list of failure cases can be found in Document.domain > Failures.

> **Note:** When using `document.domain` to allow a subdomain to access its parent, you need to set `document.domain` to the *same value* in both the parent domain and the subdomain. This is necessary even if doing so is setting the parent domain back to its original value. Failure to do this may result in permission errors.

Cross-origin network access

The same-origin policy controls interactions between two different origins, such as when you use `XMLHttpRequest` or an `<img>` element. These interactions are typically placed into three categories:

- Cross-origin *writes* are typically allowed. Examples are links, redirects, and form submissions. Some HTTP requests require preflight.
- Cross-origin *embedding* is typically allowed. (Examples are listed below.)
- Cross-origin *reads* are typically disallowed, but read access is often leaked by embedding. For example, you can read the dimensions of an embedded image, the actions of an embedded script, or the availability of an embedded resource .

Here are some examples of resources which may be embedded cross-origin:

- JavaScript with `<script src="…"></script>`. Error details for syntax errors are only available for same-origin scripts.
- CSS applied with `<link rel="stylesheet" href="…">`. Due to the relaxed syntax rules of CSS, cross-origin CSS requires a correct `Content-Type` header. Restrictions vary by browser: Internet Explorer , Firefox , Chrome , Safari (scroll down to CVE-2010-0051) and Opera .
- Images displayed by `<img>`.
- Media played by `<video>` and `<audio>`.
- External resources embedded with `<object>` and `<embed>`.
- Fonts applied with `@font-face`. Some browsers allow cross-origin fonts, others require same-origin.
- Anything embedded by `<iframe>`. Sites can use the `X-Frame-Options` header to prevent cross-origin framing.

How to allow cross-origin access

Use CORS to allow cross-origin access. CORS is a part of HTTP that lets servers specify any other hosts from which a browser should permit loading of content.

How to block cross-origin access

- To prevent cross-origin writes, check an unguessable token in the request — known as a Cross-Site Request Forgery (CSRF) token. You must prevent cross-origin reads of pages that require this token.
- To prevent cross-origin reads of a resource, ensure that it is not embeddable. It is often necessary to prevent embedding because embedding a resource always leaks some information

about it.

- To prevent cross-origin embeds, ensure that your resource cannot be interpreted as one of the embeddable formats listed above. Browsers may not respect the `Content-Type` header. For example, if you point a `<script>` tag at an HTML document, the browser will try to parse the HTML as JavaScript. When your resource is not an entry point to your site, you can also use a CSRF token to prevent embedding.

Cross-origin script API access

JavaScript APIs like `iframe.contentWindow`, `window.parent`, `window.open`, and `window.opener` allow documents to directly reference each other. When two documents do not have the same origin, these references provide very limited access to `Window` and `Location` objects, as described in the next two sections.

To communicate between documents from different origins, use `window.postMessage`.

Specification: HTML Living Standard § Cross-origin objects .

Window

The following cross-origin access to these `Window` properties is allowed:

Methods

`window.blur`

`window.close`

`window.focus`

`window.postMessage`

Attributes

`window.closed`	Read only.
`window.frames`	Read only.
`window.length`	Read only.
`window.location`	Read/Write.
`window.opener`	Read only.

Attributes

`window.parent`	Read only.
`window.self`	Read only.
`window.top`	Read only.
`window.window`	Read only.

Some browsers allow access to more properties than the above.

Location

The following cross-origin access to `Location` properties is allowed:

Methods

`location.replace`

Attributes

`URLUtils.href`	Write-only.

Some browsers allow access to more properties than the above.

Cross-origin data storage access

Access to data stored in the browser such as Web Storage and IndexedDB are separated by origin. Each origin gets its own separate storage, and JavaScript in one origin cannot read from or write to the storage belonging to another origin.

Cookies use a separate definition of origins. A page can set a cookie for its own domain or any parent domain, as long as the parent domain is not a public suffix. Firefox and Chrome use the Public Suffix List to determine if a domain is a public suffix. Internet Explorer uses its own internal method to determine if a domain is a public suffix. The browser will make a cookie available to the given domain including any sub-domains, no matter which protocol (HTTP/HTTPS) or port is used. When you set a cookie, you can limit its availability using the `Domain`, `Path`, `Secure`, and `HttpOnly` flags. When you read a cookie, you cannot see from where it was set. Even if you use only secure https connections, any cookie you see may have been set using an insecure connection.

See also

- Same Origin Policy at W3C
- Same-origin policy at web.dev
- `Cross-Origin-Resource-Policy`
- `Cross-Origin-Embedder-Policy`

Last modified: Apr 22, 2022, by MDN contributors