ChatGPT

Iframes and Sandbox Basics

An `<iframe>` embeds another HTML page within a page, but this "invisible" window can pose security risks if not handled carefully. By default (no `sandbox` attribute), an iframe can run scripts, navigate the parent, submit forms, etc. If it's same-origin with its parent, it even shares cookies and storage, giving it full privileges. Cross-origin frames are still dangerous: they can *perform* authenticated requests (cookies are sent by default unless flagged) and they can also facilitate *clickjacking* or *Cross-Frame Scripting* (XFS). For example, OWASP warns that a malicious script can load a legitimate site in an iframe and steal credentials ("Cross-Frame Scripting") [1]. Likewise, hidden iframes can trigger **CSRF** actions on a target site (auto-submitting a form under the user's session) [2]. Importantly, the same-origin policy prevents an attacker's page from reading the iframe's content if origins differ [3], but it does *not* stop the iframe from sending state-changing requests using the user's cookies. Thus in an attack, an invisible iframe can quietly POST to a banking site using the user's session (CSRF), even though the attacker cannot read the response [2] [3].

To prevent unauthorized framing (e.g. clickjacking or XFS), modern sites should send anti-framing headers. The CSP `frame-ancestors` directive (or older `X-Frame-Options`) tells browsers which origins may embed the page. For example, setting `Content-Security-Policy: frame-ancestors 'none'` is equivalent to `X-Frame-Options: DENY` [4], completely disallowing framing. In general, sanitizing all untrusted iframe content and using CSP/frame-ancestors is strongly recommended.

The `sandbox` Attribute

The HTML5 `sandbox` attribute on an iframe enforces *least privilege* on the framed content [5] [6]. Without any flags (e.g. `<iframe sandbox>`), **all** of the following are blocked [7]: - **No scripts**: All JavaScript is disabled (inline and external) [7].
- **Unique Origin**: The framed document is given a new opaque origin, so it cannot access any cookies, localStorage, or sessionStorage from any origin (not even itself) [7].
- **No top-level navigation**: Scripts can only navigate *within* the iframe, not break out to `window.top` [8] [3].
- **No forms or plugins**: Form submissions, plugins and pop-up windows are blocked by default [8].

These restrictions can be *partially lifted* by including space-separated keywords in the `sandbox` value. Key tokens include [6] [9]:

- `allow-scripts`: Permits the iframe to run JavaScript. Without this, *no* script or event handler will execute [7] [6].
- `allow-same-origin`: Restores the original origin for the iframe. With this, a page from `https://example.com/` keeps that origin (and its cookies/storage) instead of being forced to "null" [6] [9]. (Using `allow-scripts` **and** `allow-same-origin` together essentially gives the iframe full original privileges.)
- `allow-forms`: Allows form submissions and navigation triggered by forms. Without it, clicking a `<form>` does nothing [8].

- `allow-popups`: Allows pop-ups (e.g. via `window.open` or `target="_blank"`). Otherwise new windows silently fail [8].
- `allow-popups-to-escape-sandbox`: If *not* set, any popup opened by the iframe inherits the same sandbox restrictions. Including this token lets the popup run without those sandbox flags.
- `allow-top-navigation` (or `allow-top-navigation-by-user-activation`): Lets the iframe navigate the top-level window (`window.top.location`). Without it, any attempt to change the parent page is blocked [6] [10].
- Additional flags like `allow-modals`, `allow-orientation-lock`, `allow-pointer-lock` each re-enable specific APIs (e.g. `alert()` or Pointer Lock) that are otherwise disabled [11] [12].

As MDN explains, `sandbox=""` (empty) applies **all** restrictions, and each keyword *adds back* one capability [7] [6]. For example, embedding Twitter's button only needs scripts, popups, forms and same-origin, so Twitter suggests:

```
<iframe sandbox="allow-same-origin allow-scripts allow-popups allow-forms"
src="https://twitter.com/widgets/tweet_button.html"></iframe>
```

This precisely gives the iframe only the rights it needs (script and cookies access) and disallows everything else [13] [6]. In general, the recommendation is *start with full sandbox* (`sandbox` with no value) and explicitly enable only what you trust.

`src` vs. `srcdoc`

An iframe normally loads content via its `src` URL. The newer `srcdoc` attribute instead embeds literal HTML content (like inline HTML) inside the frame. This is equivalent to having an `about:srcdoc` location. Notably, **sandboxed** `srcdoc` frames inherit the parent document's *base URL* and *Content Security Policy*, even though the iframe itself has an opaque origin [14] [15]. In practice, this means: - Relative links in `srcdoc` content resolve against the parent's URL, not a separate one [15]. - `document.baseURI` inside the iframe returns the parent's URL (which may include secrets in its path or hash) [15]. - The iframe's loaded HTML still obeys the parent's CSP (unless the parent has none) [14].

This behavior has been exploited in CTFs. For example, *Johann Carlsson (May 2024)* showed that a sandboxed iframe with `srcdoc` could leak the parent's location: a script inside fetched an empty string (triggering a request to the parent URL) and the ensuing CORS error or baseURI gave away the parent's address [16] [15]. In short, `srcdoc` frames are very constrained in origin but not completely blind: they see the parent's URL (via `baseURI`) and CSP [14] [15].

Cross-Domain and Communication

Even with sandboxing, iframes are still subject to the browser's same-origin policy. A script in an iframe **cannot** directly read or manipulate the parent window if origins differ [3]. Cross-origin communication must use `window.postMessage()`, carefully checking the message's `origin` field [17] [3]. For instance, one writeup advises always verifying `if (event.origin !== "https://trusted.com") return;` before trusting posted messages [17].

If an iframe's origin is "null" (as happens by default in a sandbox without `allow-same-origin`), even its `postMessage` events report `event.origin === "null"` [18] [3], which is a signal to validate carefully. (Chrome's newer "credentialless" iframes use a similar empty-origin model for COEP, explicitly loading without any cookies or storage [19].)

CSRF and CORS Considerations

Because iframes send stored credentials by default (unless cookies have `SameSite=Strict`), they can facilitate CSRF if not defended. In a CSRF scenario, a malicious site embeds a hidden iframe pointing to a bank or API, and triggers a form submission or XHR inside it. The browser includes the user's session cookie, so the request is authenticated as the user. As one blog notes, "A hidden iframe loads in the background and performs actions without user consent," all under the user's session [2]. However, the attacker's page can't read the result due to SOP [3] – it can only cause side effects. (Modern defenses like `SameSite` cookies can block this; `SameSite=Strict` will *not* send cookies to cross-site iframes [20].)

Cross-Origin Resource Sharing (CORS) comes into play if an iframe's script tries to fetch data from another domain. By default, a fetch from within an iframe to a different origin will be blocked unless that origin allows it via CORS headers. For example, in Johann's sandbox-iframe challenge, a `fetch("")` from origin `null` to the parent's URL was blocked by CORS as expected [16]. In general, a sandboxed iframe without `allow-same-origin` cannot bypass CORS/SOP to read parent data. If allow-same-origin is granted, the iframe's scripts run with the embedded page's normal origin (and can access its cookies/storage), but cross-site data access still requires proper CORS.

Sandbox Escape Examples (CTFs and Research)

Attackers have found creative ways to weaken or bypass sandbox restrictions. Many CTF write-ups illustrate these tactics:

- **Srcdoc Base-URL Leak (Golf Jail CTF 2023)**: A sandboxed iframe with `allow-scripts` only was given a short XSS payload. Because it was `srcdoc`, the attack used the `baseURI` property of an element (`<svg>`) to retrieve the parent page's full URL, which contained the secret flag in a comment. The write-up explains: *"When you're within an iframe, you can't use* `location` *or* `top.location` *due to sandbox, but you can leverage the* `baseURI` *property… This property provides the absolute base URL of the document"* [21]. In practice, the payload did `eval(baseURI)` to get the flag.

- **Sandbox + Popup Inheritance (soXSS 2022)**: A "same-origin XSS" challenge exploited sandbox inheritance rules. The iframe was sandboxed with `allow-scripts allow-popups allow-top-navigation`. The attacker noted that a popup opened from this iframe inherits the sandbox flags by default [22]. By opening a popup from a `null`-origin iframe, the popup stayed sandboxed (origin `null`), allowing the attacker to inject a message there and then break out into the real origin. In short, using `allow-popups` without `allow-popups-to-escape-sandbox` let them chain through a `null` origin to steal data [22].

- **History Navigation Reversal (idekCTF 2024)**: Another trick is to change the sandbox state via navigation. In one example, a page loaded `test1` in a sandboxed iframe (so its script didn't run). A user action then *removed* the `sandbox` attribute and set `iframe.src = 'data:...test2...'`, causing the script in `test2` to execute. If the user then clicks "back", the iframe returns to `test1`, but crucially *the sandbox attribute stays removed*, so the previously disabled script in `test1` now executes [23]. The write-up clarifies: *"Pressing back, the sandbox is gone, and the script in test1 can now execute"* [23].

- **Parent DOM Access via null-origin (sandbox-iframe XSS, 2024)**: Carlsson's write-up describes using a sandboxed `srcdoc` iframe to break out. The payload (via `srcdoc`) runs scripts under `allow-scripts` but *no* `allow-same-origin`, so its `window.origin` is `null` [24]. However, because it loaded through `srcdoc`, it saw the parent's CSP and base URI. The iframe script used `fetch("")` which triggered a CORS request to the parent URL; the resulting CORS error leaked the full parent location [16]. The payload then did `window.top.location = "/"` (allowed by the challenge) to send that information out. This clever chain shows that even a "null"-origin iframe can sometimes communicate critical context to the attacker [16].

These examples highlight that browser behaviors (history, `baseURI`, window opening) can sometimes *leak* or *change* the security context despite the sandbox. Many such issues have been discussed in security forums (e.g. CVE-2017-7788 was a CSP bypass via sandboxed iframe srcdoc) and continue to be actively researched. In response, browsers have tightened rules (new CSP handling for `srcdoc`, disallowing some postMessage uses of `"null"`, etc.), but defenders should assume complex interactions and keep software up to date.

Mitigations and Best Practices

Given these subtleties, best practices include:

- **Minimal sandbox flags:** Only use the absolute minimum (`allow-*`) tokens needed. For fully untrusted content, use a strict sandbox (no flags). Only grant `allow-same-origin` if the content *must* retain its origin (and even then, be cautious, since `allow-same-origin + allow-scripts` on same-origin content essentially disables the sandbox) [10].
- **Use CSP and X-Frame-Options/frame-ancestors:** Explicitly forbid unwanted framing. Setting `frame-ancestors 'none'` (or `X-Frame-Options: DENY`) ensures your site cannot be iframed by others [4].
- **Cookie and CORS hygiene:** Mark cookies with `Secure; HttpOnly; SameSite=Strict` unless cross-site access is needed [20]. This prevents cookies (and thus sessions) from being sent inside cross-origin iframes, mitigating CSRF.
- **Careful messaging:** If you must communicate with a cross-origin iframe via `postMessage`, always validate `event.origin`. As one guide warns, *"verify every message's origin"* to avoid accepting malicious commands from a rogue iframe [17].
- **Stay updated:** Many sandbox escape techniques rely on specific browser behaviors. Keep browsers updated, and follow security advisories (e.g. Chrome's "credentialless" iframes to avoid cookie leaks [19]).

In short, iframes should be treated as untrusted by default. Use sandboxing and headers to quarantine them. When using sandbox flags like `allow-same-origin` or `allow-top-navigation`, do so only when absolutely needed and understand the trust implications [6] [10]. By applying the principle of least privilege and monitoring current research (CTF writeups, vendor blogs), developers can avoid most iframe-related attacks.

Sources: Authoritative docs and security research, including MDN and web.dev guides on `<iframe sandbox>` [7] [6] [3], recent CTF writeups on sandboxed XSS [16] [21] [22] [23], and security blogs on iframe risks [1] [17] [25] [4].

---

[1] [2] [17] [20] [25] 2025 Iframe Security Risks and 10 Ways to Secure Them
https://qrvey.com/blog/iframe-security/

3 9 10 11 12 : The Inline Frame element - HTML | MDN</span> </div> <div class="citation-url"> <a
href="https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/iframe">https://
developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/iframe</a> </div> </div> </div> <div
class="citation"> <div class="citation-body"> <div class="title_wrapper"> <a class="source" href="https://
developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy/frame-
ancestors#:~:text=Setting%20this%20directive%20to%20,also%20supported%20in%20older%20browsers">4</a>
<span class="title">Content-Security-Policy: frame-ancestors directive - HTTP | MDN</span> </div> <div
class="citation-url"> <a href="https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-
Security-Policy/frame-ancestors">https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/
Content-Security-Policy/frame-ancestors</a> </div> </div> </div> <div class="citation"> <div class="citation-
body"> <div class="title_wrapper"> <a class="source" href="https://web.dev/articles/sandboxed-
iframes#:~:text=Given%20an%20iframe%20with%20an,it%20to%20the%20following%20restrictions">5</a> <a
class="source" href="https://web.dev/articles/sandboxed-
iframes#:~:text=%2A%20%60allow,level%20window">6</a> <a class="source" href="https://web.dev/articles/
sandboxed-iframes#:~:text=,DOM%20storage%2C%20Indexed%20DB%2C%20etc">7</a> <a class="source"
href="https://web.dev/articles/sandboxed-iframes#:~:text=or%20%60target%3D,are%20blocked">8</a> <a
class="source" href="https://web.dev/articles/sandboxed-
iframes#:~:text=To%20figure%20out%20what%20we,by%20sandboxing%20the%20frame%27s%20content">13</
a> <a class="source" href="https://web.dev/articles/sandboxed-
iframes#:~:text=window.addEventListener%28%27message%27%2C%20function%20%28e%29%20,data">18</a>
<span class="title">Play safely in sandboxed IFrames | Articles | web.dev</span> </div> <div class="citation-
url"> <a href="https://web.dev/articles/sandboxed-iframes">https://web.dev/articles/sandboxed-iframes</a> </
div> </div> </div> <div class="citation"> <div class="citation-body"> <div class="title_wrapper"> <a class="source"
href="https://joaxcar.com/blog/2024/05/16/sandbox-iframe-xss-challenge-solution/
#:~:text=The%20setup%20allows%20us%20to,scripts%20that%20passes%20this%20CSP">14</a> <a
class="source" href="https://joaxcar.com/blog/2024/05/16/sandbox-iframe-xss-challenge-solution/
#:~:text=Doing%20a%20%60fetch%28,make%20a%20request%20like%20this">16</a> <a class="source"
href="https://joaxcar.com/blog/2024/05/16/sandbox-iframe-xss-challenge-solution/
#:~:text=The%20iframe%20sandbox%20contains%20the,null">24</a> <span class="title">Sandbox-iframe XSS
challenge solution - Johan Carlsson</span> </div> <div class="citation-url"> <a href="https://joaxcar.com/blog/
2024/05/16/sandbox-iframe-xss-challenge-solution/">https://joaxcar.com/blog/2024/05/16/sandbox-iframe-xss-
challenge-solution/</a> </div> </div> </div> <div class="citation"> <div class="citation-body"> <div
class="title_wrapper"> <a class="source" href="https://github.com/whatwg/html/issues/
8105#:~:text=A%20sandboxed%20iframe%20srcdoc%20inherits,document.baseURI">15</a> <span
class="title">Sandboxed iframe srcdoc inherits parent's document base URL · Issue #8105 · whatwg/html ·
GitHub</span> </div> <div class="citation-url"> <a href="https://github.com/whatwg/html/issues/8105">https://
github.com/whatwg/html/issues/8105</a> </div> </div> </div> <div class="citation"> <div class="citation-body">
<div class="title_wrapper"> <a class="source" href="https://developer.chrome.com/blog/iframe-
credentialless#:~:text=Example%3A">19</a> <span class="title">Iframe credentialless: Easily embed iframes in
COEP environments | Blog | Chrome for Developers</span> </div> <div class="citation-url"> <a href="https://
developer.chrome.com/blog/iframe-credentialless">https://developer.chrome.com/blog/iframe-credentialless</
a> </div> </div> </div> <div class="citation"> <div class="citation-body"> <div class="title_wrapper"> <a
class="source" href="https://www.offensiveweb.com/docs/writeup/sekai-ctf-2023-golf-jail/
#:~:text=When%20you%E2%80%99re%20within%20an%20,the%20document%20housing%20the%20node">21</
a> <span class="title">SEKAI CTF 2023 - Golf Jail | OffensiveWeb</span> </div> <div class="citation-url"> <a
href="https://www.offensiveweb.com/docs/writeup/sekai-ctf-2023-golf-jail/">https://www.offensiveweb.com/
docs/writeup/sekai-ctf-2023-golf-jail/</a> </div> </div> </div> <div class="citation"> <div class="citation-body">
<div class="title_wrapper"> <a class="source" href="https://github.com/terjanq/same-origin-

xss#:~:text=will%20be%20,the%20solution%20to%20the%20challenge">22</a> <span class="title">GitHub - terjanq/same-origin-xss: Same Origin XSS challenge</span> </div> <div class="citation-url"> <a href="https://github.com/terjanq/same-origin-xss">https://github.com/terjanq/same-origin-xss</a> </div> </div> </div> <div class="citation"> <div class="citation-body"> <div class="title_wrapper"> <a class="source" href="https://blog.huli.tw/2024/09/07/en/idek-ctf-2024-iframe/#:~:text=1,so%20the%20script%20will%20execute">23</a> <span class="title">idekCTF 2024 Writeup - Advanced iframe Magic - Huli's blog</span> </div> <div class="citation-url"> <a href="https://blog.huli.tw/2024/09/07/en/idek-ctf-2024-iframe/">https://blog.huli.tw/2024/09/07/en/idek-ctf-2024-iframe/</a> </div> </div> </div> </div> </body> </html>