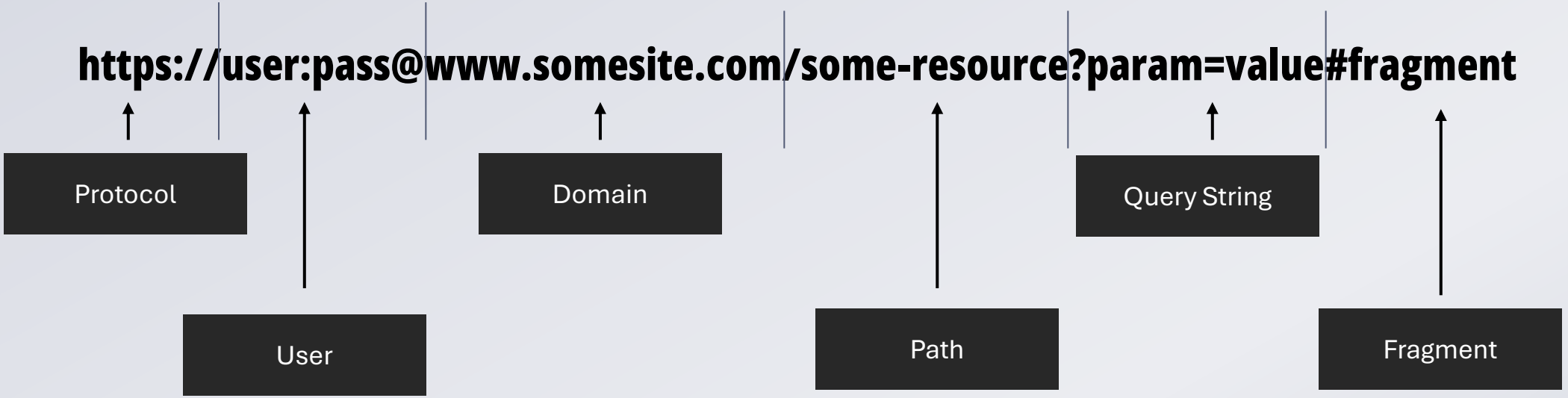
Cross-App Security I

Introduction to Same-Origin Policy

Key Concepts

- **URL: Uniform Resource Locator**
  A type of URI that specifies how and where to access a resource
- **URI: Uniform Resource Identifier**
  String identifying a resource (often used interchangeably with URL)
- **Domain**
  A hierarchical name within the Domain Name System (DNS) namespace
- **Hostname**
  Fully qualified name (often FQDN) to identify a host/endpoint
- **eTLD+1**
  - Registerable domain (*google.com*)

Key Concepts: URL



Key Concepts: Origin and Site

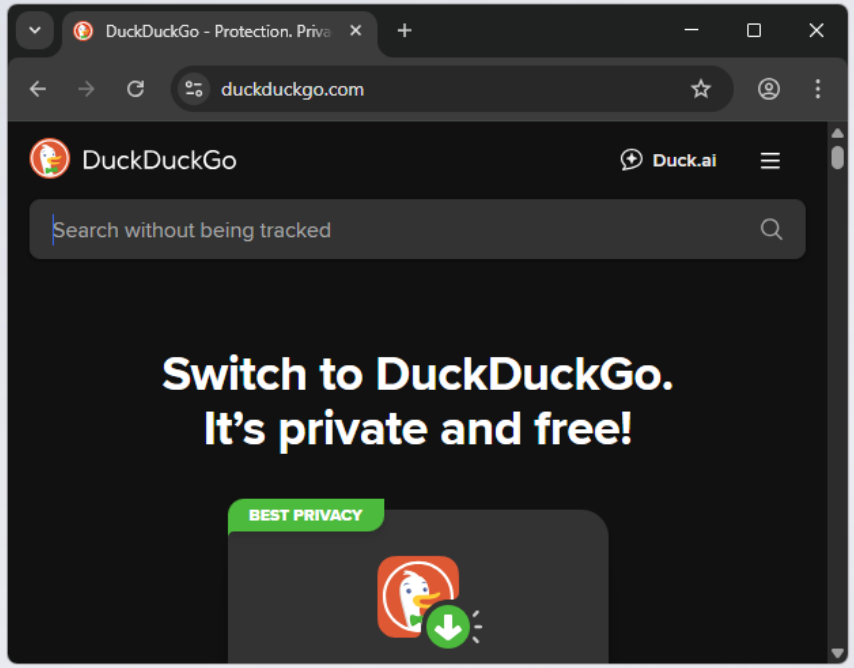
Aspect	Origin	Site
Definition	Scheme + Host + Port	Scheme + Registrable domain (~eTLD+1)
Example	https://app.example.com:443	https://example.com
Used By	Same-Origin Policy, JS Execution, Storage, DOM access, CORS, COOP/COEP	SameSite Cookie Attribute

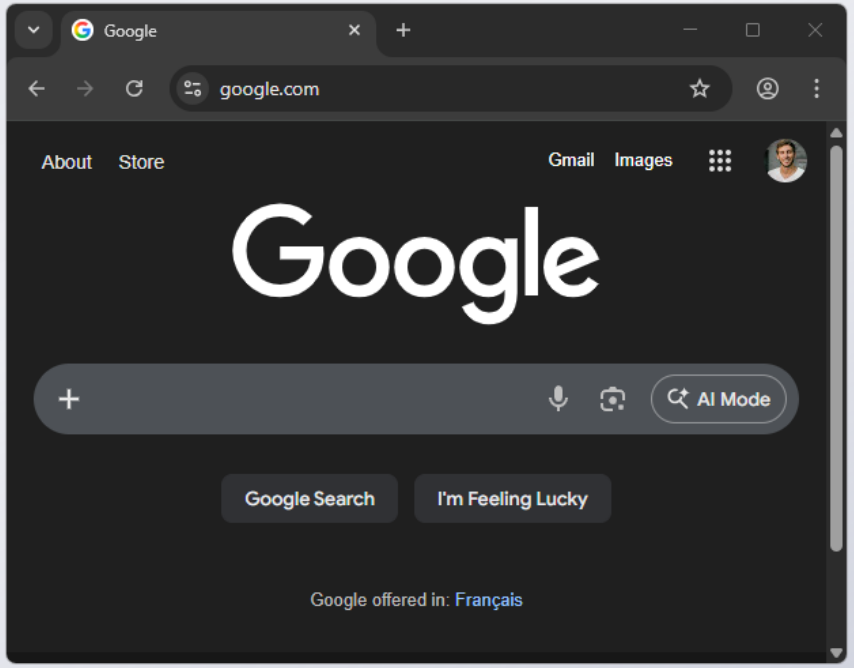
Same-Origin Policy

The core web security model that attempts to restrict different web apps from unintended violations of the confidentiality or integrity of other web apps

Key Concepts: Same-Origin Policy (SOP)

- The security model protecting distinct web applications
- Logical boundary between apps from different origins
- Does not prevent ALL interaction (it's messy)
- Enforced by the browser





What Does SOP Allow and Deny?

Permitted: Cross-Origin Writes

```
<form action="https://example.com/submit" method="post">
  <input  name="name" value="admin">
  <button>Send</button>
</form>
```

Blocked: *Some* Cross-Origin Writes

Permitted HTTP Request Methods (all others blocked):

- **GET**
- **POST**
- **Head**
- **OPTIONS (as a side effect)**

Permitted HTTP Request Headers (all others blocked):

- **Accept**
- **Accept-Language**
- **Content-Language**
- **Content-Type, limited to:**
  - **application/x-www-form-urlencoded**
  - **multipart/form-data**
  - **text/plain**

Note: some exceptions to the above

Blocked: *Some* Cross-Origin Writes

Initiation Mechanism	Methods Allowed	CORS Required?
`<img>`, `<script>`, `<link>`	GET	❌
`<form>`	GET, POST	❌
Navigation	GET	❌
`fetch()` / XHR	GET, POST, HEAD	❌ (simple only)
`fetch()` / XHR	PUT, PATCH, DELETE, ...	✅
OPTIONS	Browser-initiated only	✅

NOTE: this table is not comprehensive (additional mechanisms exist)

Blocked: Cross-Origin Reads

```
<script>
  const w = window.open("https://example.com/submit");

  // Cross-origin access blocked
  console.log(w.document.body.innerText);
</script>
```

```
>> const w = window.open("https://example.com");
<- undefined
>> console.log(w.document.body.innerText);
Uncaught DOMException: Permission denied to access property "document" on cross-origin object
    <anonymous> debugger eval code:1
```

Permitted: Cross-Origin Embedding

Blocked: Inspecting Embeds

```
<canvas id="c"></canvas>

<script>
  const img = document.getElementById("img");
  const ctx = c.getContext("2d");

  img.onload = () => {
    ctx.drawImage(img, 0, 0);

    // Blocked
    ctx.getImageData(0, 0, 1, 1);
  };
</script>
```

Permitted: Cross-Origin Script Execution

Blocked: Reading Scripts

```
<script src="https://victim.com/secret.js"></script>

<script>
  // BLOCKED
  console.log(document.scripts[0].text);
</script>
```

Same-Origin Policy Workarounds and Abuses

Cross-Origin Resource Sharing (CORS)

An opt-in mechanism to relax SOP for cross-origin HTTP reads

JSONP

Old hacky technique for cross-origin HTTP reads prior to CORS

postMessage

A cross-origin communication (messaging) mechanism

Metadata Reads

Information exposed when origins interact may be abused (XS-Leaks)

Same-Origin Policy Workarounds and Abuses

Window References

- Window objects can share properties (XS-Leaks)

Cookies!

- Cookies do NOT respect SOP

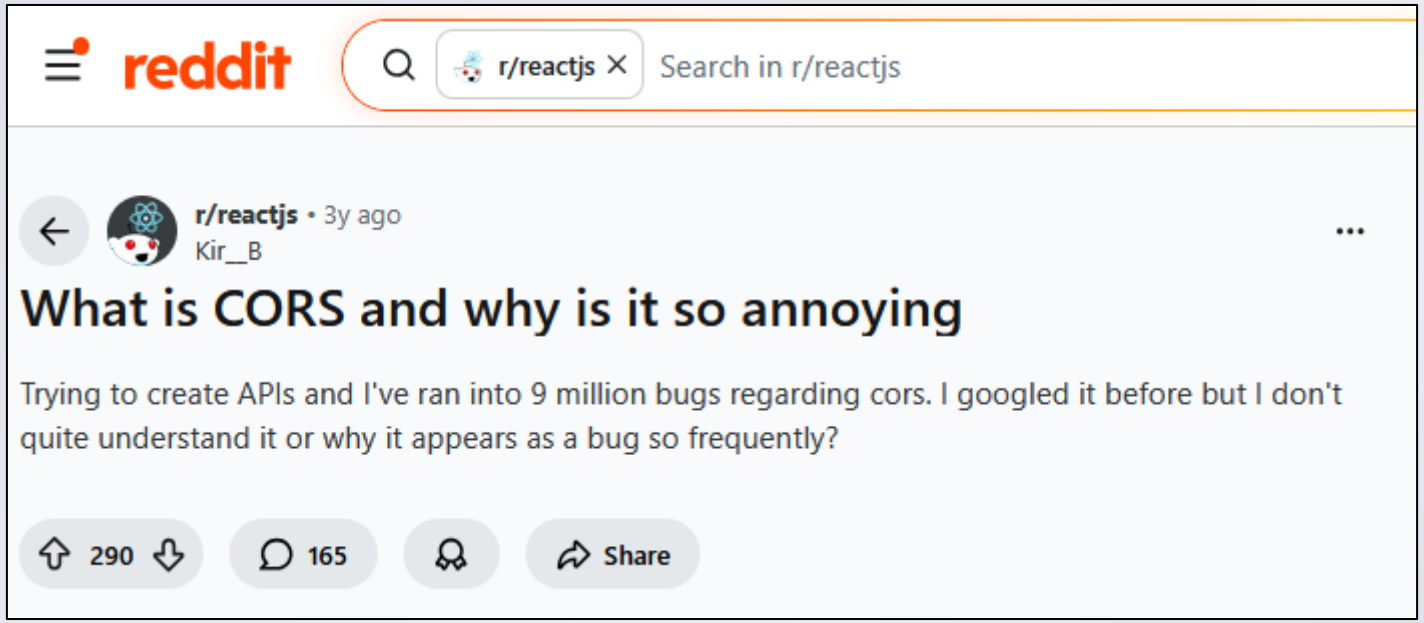
Side Channel Leaks

- Timing, errors, caching behavior and more! (XS-Leaks)

User Action

- Will relax SOP in some cases (XS-Leaks)

CORS: Cross-Origin Resource Sharing

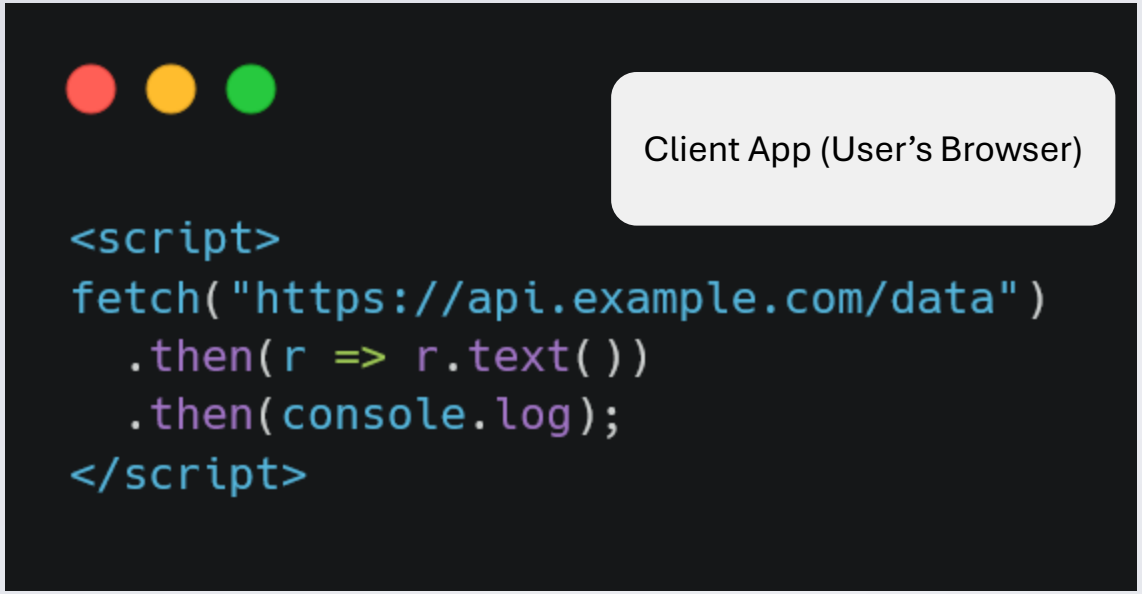


https://www.reddit.com/r/reactjs/comments/11cyejn/what_is_cors_and_why_is_it_so_annoying/

CORS: Cross-Origin Resource Sharing

- Relaxes SOP for sending requests and reading responses
- Servers explicitly opt in using *Access-Control-Allow-Origin*
- Browsers enforce CORS as servers do not block requests
- With *Access-Control-Allow-Credentials*, ambient authority type credentials (HTTP Cookies, HTTP Auth, mTLS) can be invoked for authorized requests
- CORS can permit HTTP methods/headers not typically permitted
- Fetch API is the standard consumer of CORS capabilities

CORS: Cross-Origin Resource Sharing



CORS Modes and Simple Requests

- ***Simple Requests*** **use "safelisted" methods, headers, and content types**
- ***Simple Requests*** **qualify for "no-cors" mode (no preflight needed)**
- ***Non-Simple Requests*** **are any that cannot be issued in *no-cors* mode**
- ***Non-Simple Requests*** **require a preflight (OPTIONS permission check) to be sent**
- ***"Simplicity"*** **is governed by request properties while CORS mode is determined by how the response is intended to be consumed (opaque vs readable)**
- ***Note: "Simple" no longer used in Fetch spec (https://fetch.spec.whatwg.org/)***

CORS Modes and Simple Requests

Request Type	Mode	Preflight?	Response Access
Simple cross-origin request	cors	No	*If server allows*
Simple cross-origin request	no-cors	No	Opaque
Non-simple cross-origin request	cors	Yes	*If server allows*
Non-simple cross-origin request	no-cors	Impossible	N/A

CORS Modes and Simple Requests

Simple Request

```
fetch("https://api.example", {
  method: "POST",
  headers: {
    "Content-Type": "text/plain"
  },
  body: "hello"
});
```

Non-Simple
(Requires server permission)

```
fetch("https://api.example", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ a: 1 })
});
```

JSONP: JSON with Padding

Client App (User's Browser)

```
<script src="https://api.example.com/data?callback=handle"></script>
<script>
  function handle(data) { console.log(data); }
</script>
```

Data-Sharing Server
(HTTP Response)

```
Content-Type: application/javascript
...
handle({ secret: "value" });
```

JSONP: JSON with Padding

```
<form action="https://example.com/submit" method="post">
  <input  name="name" value="admin">
  <button>Send</button>
</form>
```

postMessage

- *postMessage* intentionally allows cross-origin data exchange
- Messages flow between windows/iframes that have a shared window reference (via opening or embedding)
- Only open and related pages can communicate
- Every message includes event.origin, permitting source validation
- Senders can deliver data but cannot read DOM/state

postMessage

https://victim.com

```
<script>
window.addEventListener("message", e => {
  if (e.origin !== "https://attacker.com") return;
  console.log(e.data);
});
</script>
```

https://attacker.com

```
<script>
const w = window.open("https://victim.com");
w.postMessage("hello", "https://victim.com");
</script>
```

Metadata Reads (Image Properties)

```
<img id="x" src="https://example.com/image.jpg">
<script>
x.onload = () => {
  console.log(x.naturalWidth, x.naturalHeight);
};
</script>
```

Many such cases...

Window Reference (Frame Counting)

```
<iframe id="x" src="https://other-origin.example"></iframe>

<script>
const w = x.contentWindow;
let last = w.length;

setInterval(() => {
  if (w.length !== last) {
    console.log("frame count changed:", last, "→", w.length);
    last = w.length;
  }
}, 500);
</script>
```

Cookies (just a taste)

Side Channel Leaks

```
const t0 = performance.now();
const img = new Image();

img.onload = () => {
  console.log("load time:", performance.now() - t0);
};

img.src = "https://target.example/account/avatar";
```

User Interaction/Gesture

```
<button id="go">Open target</button>

<script>
let w;

go.onclick = () => {
  w = window.open("https://target.example");
};

setInterval(() => {
  if (w) {
    console.log("frame count:", w.length);
  }
}, 1000);
</script>
```

Considerations for Security Testing

- “Cross-App” security depends on security mechanism use and app-specific functionality
- Not all security boundaries are enforced by the Same-Origin Policy
- Same-site attacks can be more capable than cross-site attacks, even when applications are cross-origin
- SOP-allowed behaviors (embedding, navigation, messaging, metadata) can create meaningful attack surfaces
- Security testing must evaluate interaction between applications, not applications in isolation

Coming Sessions..

- HTTP Cookies
- Web Storage and Caching
- Cross-Origin Resource Sharing
- postMessage and iFrames
- Site Isolation
- Modern Cross-Site Request Forgery
- Cross-Site Leaks
- UI Redress Attacks
- Content Security Policy and Trusted Types