Cross-App Security IV

Documents, Windows, and Frames

HTML

HTML: The Living Standard

“Is this HTML5? Yes.
In more length: the term "HTML5" is widely used as a buzzword to refer to modern web technologies, many of which (though by no means all) are developed at the WHATWG. “

https://html.spec.whatwg.org

Key Concepts via HTML: The Living Standard

Navigable

Something houses a Browsing Context and displays content (tabs, iframes)

Document

The actual content and DOM state loaded from a URL; a single entry in a session history that possesses an Origin

Browsing Context

The scripting environment for a Document (execution context)

Browsing Context Group

A set of contexts that can reach each other via properties like *window.opener* and *window.frame* (process isolation typically defined at BCG)

Component Lifecycle

Navigation (clicking a link, for example) has the following outcome:

- New Document is created
- New DOM is created
    - All JavaScript state is wiped out
- The Browsing Context is unchanged
- The *window* identity is unchanged
- The underlying Window instance is **changed**

Windows on the Web

Window

We could be talking about the OS UI window that runs your browser

Window

Or we could be using the Window JavaScript Interface

window

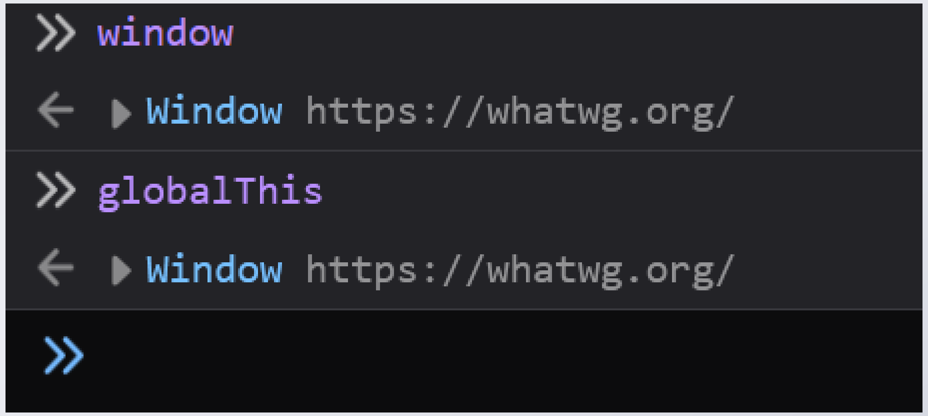
Or we could be referring to the JavaScript identifier *window*, the instance of the Window interface and global object for the current Browsing Context

WindowProxy

Or we may be discussing the WindowProxy, which the *window* identifier actually is (the proxy permits enforcement of SOP)

Working With *window*

- Global object for the current browsing context
  - Implementation of *globalThis*
- For top-level declarations in non-module scripts:
  - Variables declared with *var* (not *let*) becomes a *window* property
  - Functions declared globally becomes a method of *window*
- You don't have to type *window.* to access its properties
  - alert() is shorthand for window.alert()



```
>> window
← ▶ Window https://whatwg.org/
>> globalThis
← ▶ Window https://whatwg.org/
>>
```

Window and Its Interface

- Complex set of properties and methods
- *window* contains a *DOM* document: *window.document*
- Each browser tab/window has its own Window object, but some properties and methods are shared between tabs in a window
- Certain situations create relationships that allow cross-context (and Cross-Origin) *window* references, for example:

```
let winreference = window.open("https://example.com")
```

Cross-Origin Window References

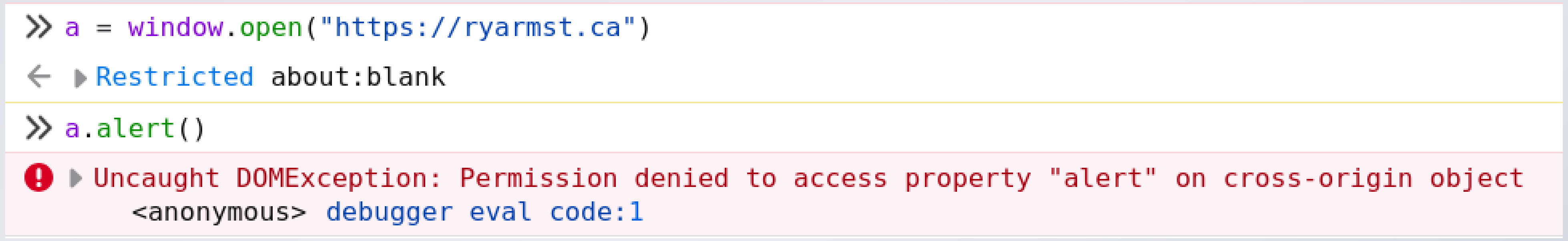
Cross-Origin applications can interact through Window interfaces where limited properties/methods are exposed

- **This includes the following situations**
  - **Forward open: *window.open()***
  - **Reverse reference: *window.opener***
  - **Framed content: *HTMLIFrameElement.contentWindow***
  - **Hierarchy references: *window.parent, window.top, window.frames[]***

Historical Capabilities

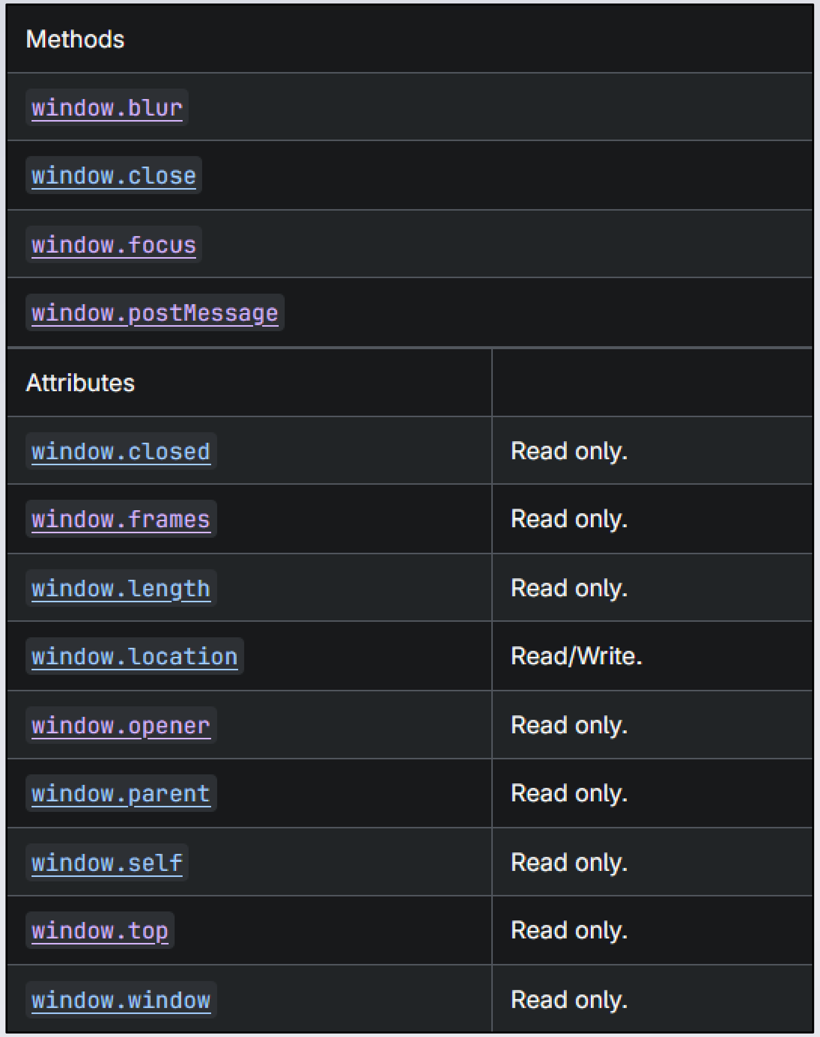
Due to Cross-Origin abuse, the following have been restricted:

- alert(), confirm()
- moveTo(), resizeTo()
- window.status
- window.print()
- window.name

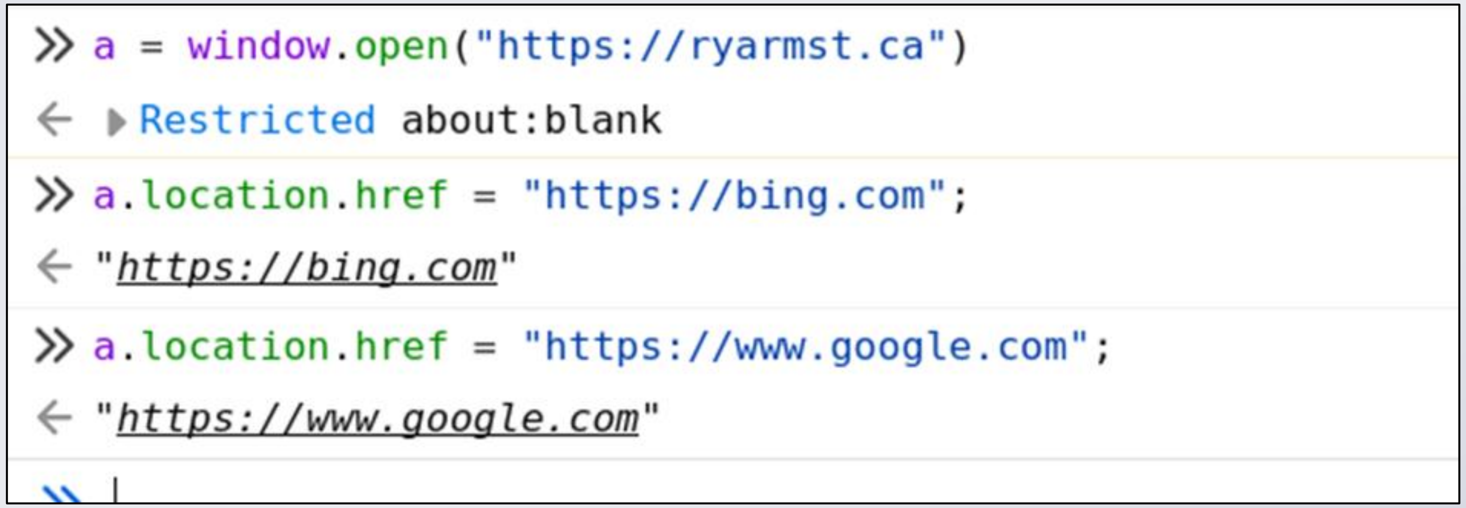


```
>> a = window.open("https://ryarmst.ca")
← ▶ Restricted about:blank
>> a.alert()
! ▶ Uncaught DOMException: Permission denied to access property "alert" on cross-origin object
      <anonymous> debugger eval code:1
```

Present Capabilities (Window)



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



```
>> a = window.open("https://ryarmst.ca")
← ▶ Restricted about:blank
>> a.location.href = "https://bing.com";
← "https://bing.com"
>> a.location.href = "https://www.google.com";
← "https://www.google.com"
>> |
```

https://developer.mozilla.org/en-US/docs/Web/Security/Defenses/Same-origin_policy

Tabnabbing Attacks

Tabnabbing attacks abuse shared Window properties through manipulation of *window.location* and often social engineering

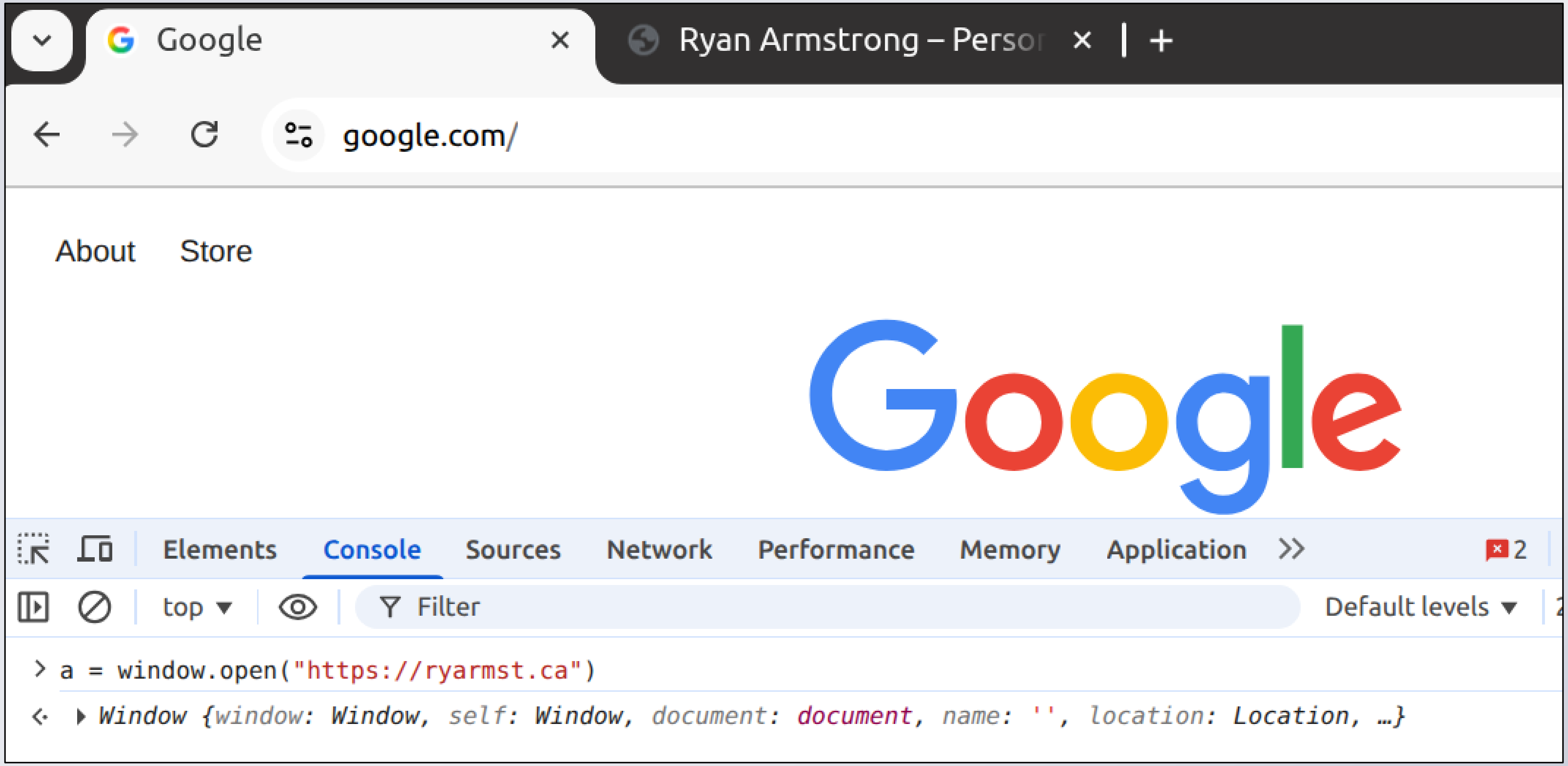
Reverse Tabnabbing:

> **A window opened by a trusted site uses *window.opener* to change the *location* property, navigating the trusted source site**

Forward Tabnabbing:

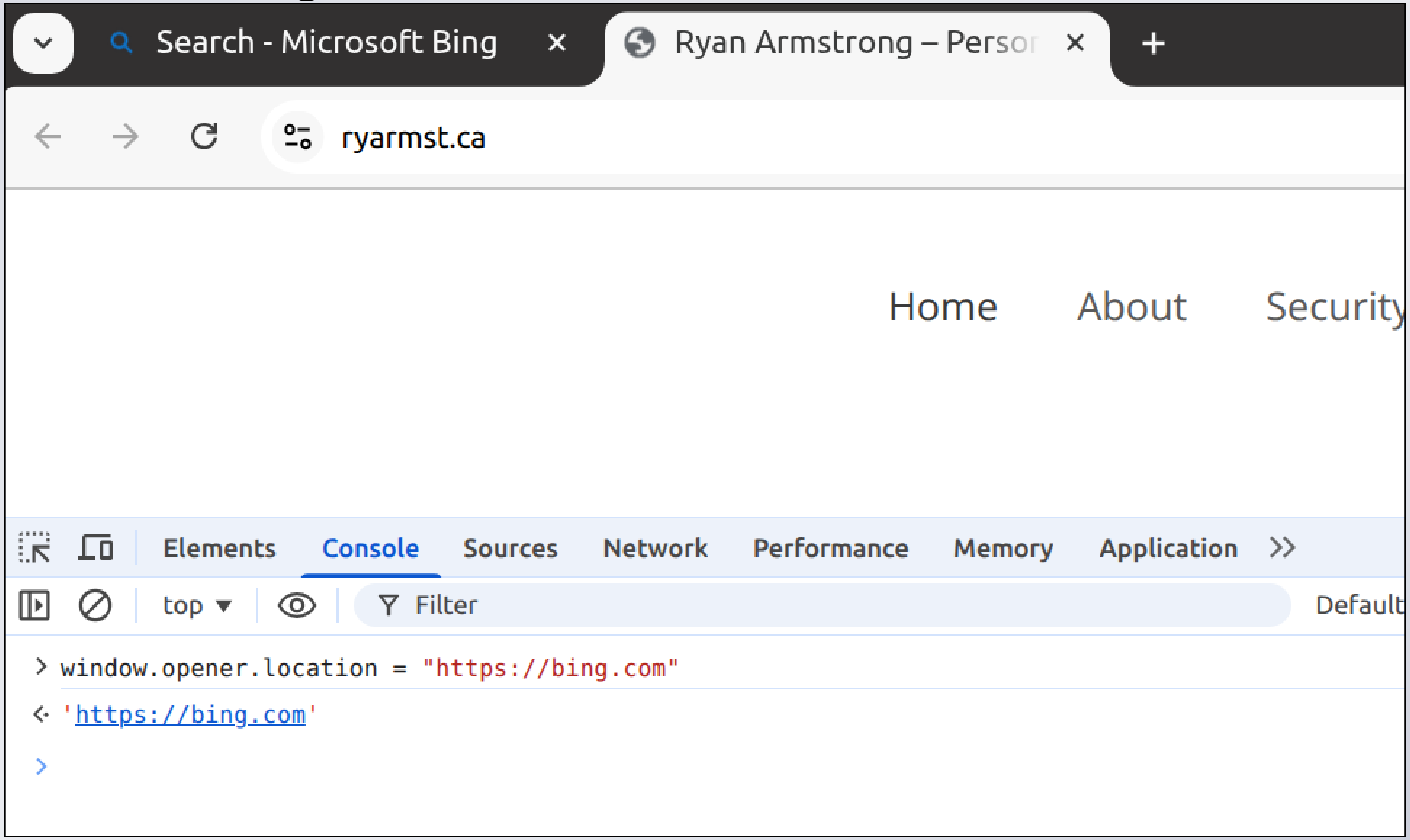
> **A site opens another, maintaining a *window* reference, which it uses to navigate away by changing *location***

Tabnabbing Attacks



```
> a = window.open("https://ryarmst.ca")
<· ▸ Window {window: Window, self: Window, document: document, name: '', location: Location, …}
```

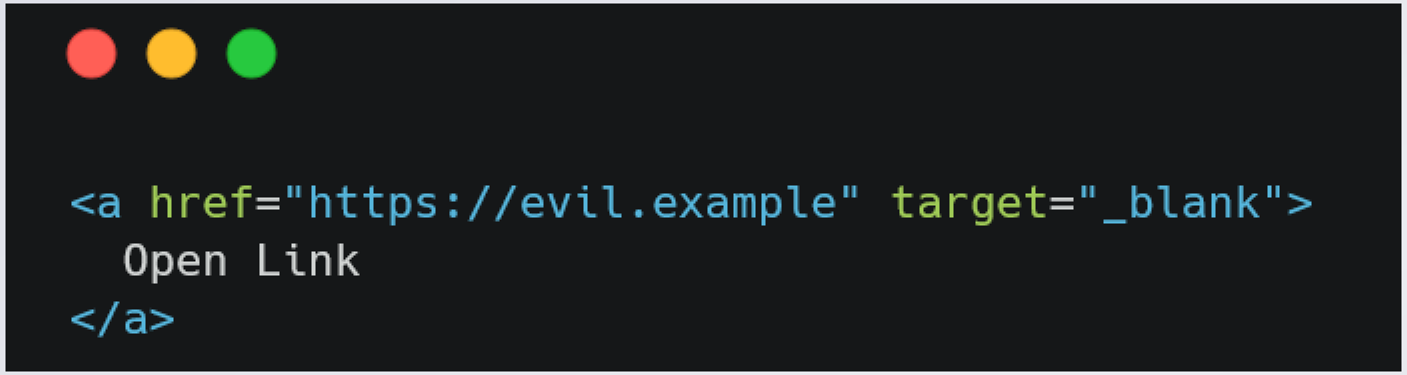
Tabnabbing Attacks



```
> window.opener.location = "https://bing.com"
<· 'https://bing.com'
>
```

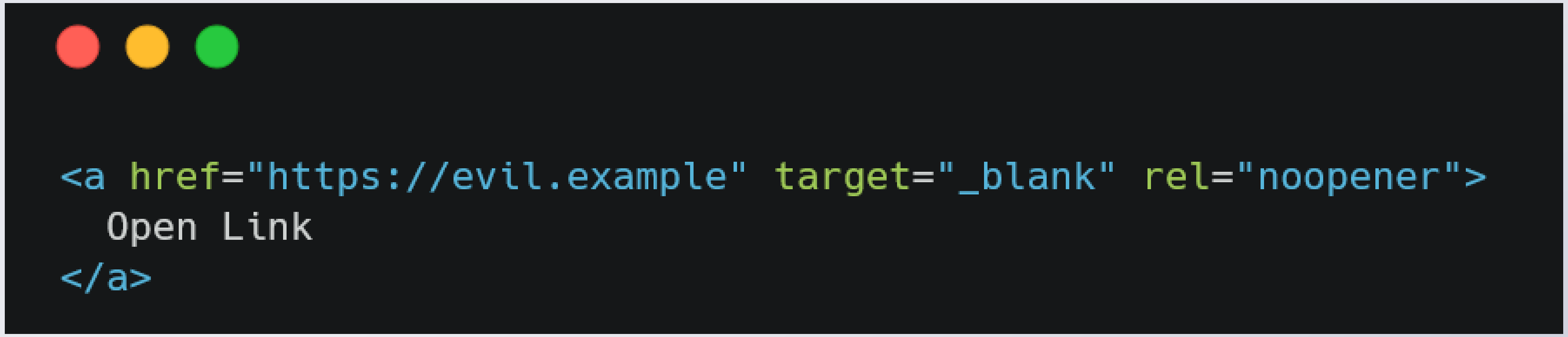
Modern Tabnabbing Attacks

Traditionally, tabnabbing impacted most navigation



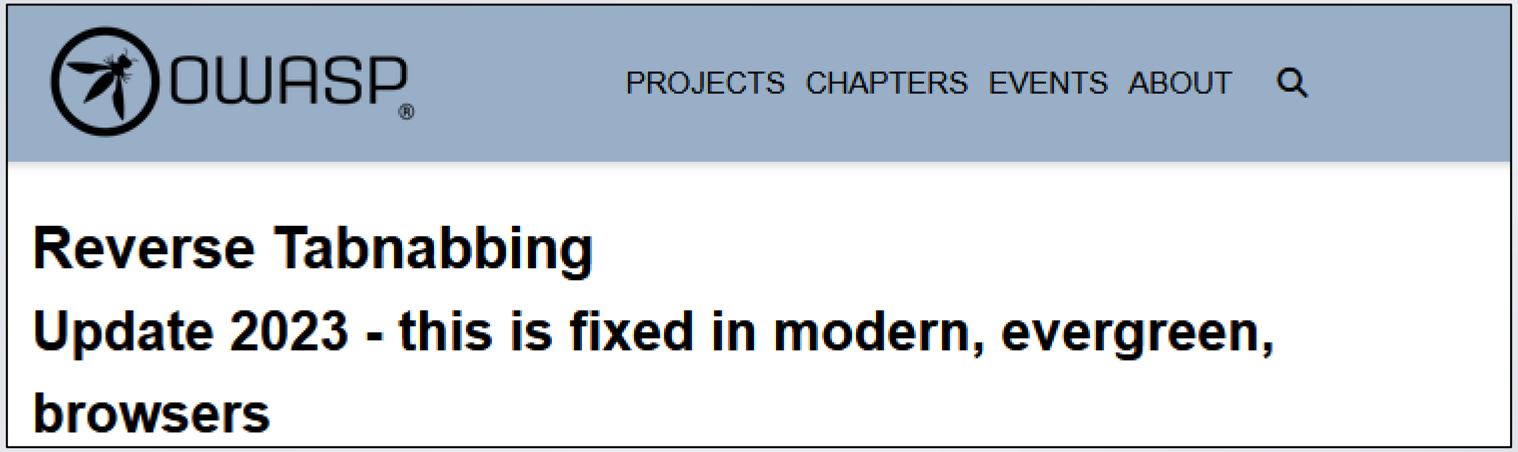
```html
<a href="https://evil.example" target="_blank">
  Open Link
</a>
```

Old recommendation (now defaulted by modern browsers):



```html
<a href="https://evil.example" target="_blank" rel="noopener">
  Open Link
</a>
```

Modern Tabnabbing Attacks: Fixed?



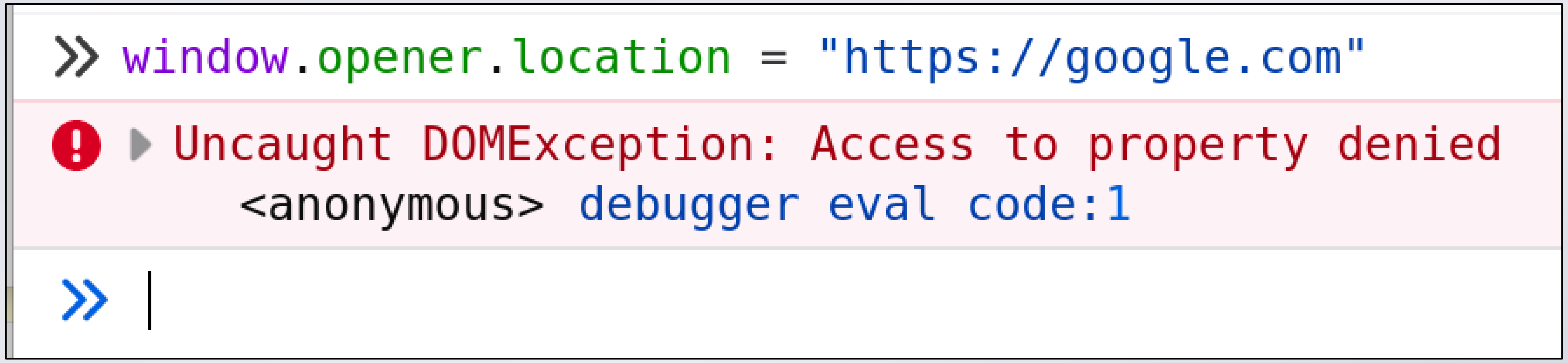
Reverse Tabnabbing

Update 2023 - this is fixed in modern, evergreen, browsers

https://owasp.org/www-community/attacks/Reverse_Tabnabbing

Modern Tabnabbing Attacks

- Still possible as a result of *window.open()* in Chromium-based browsers (previous example was in modern Chrome)
- Blocked in Firefox



```
>> window.opener.location = "https://google.com"
! ▶ Uncaught DOMException: Access to property denied
      <anonymous> debugger eval code:1
>> |
```

DBG App Test Finding
Cross-Origin Browser Tab Hijacking

Cross-Origin Opener Policy (COOP)

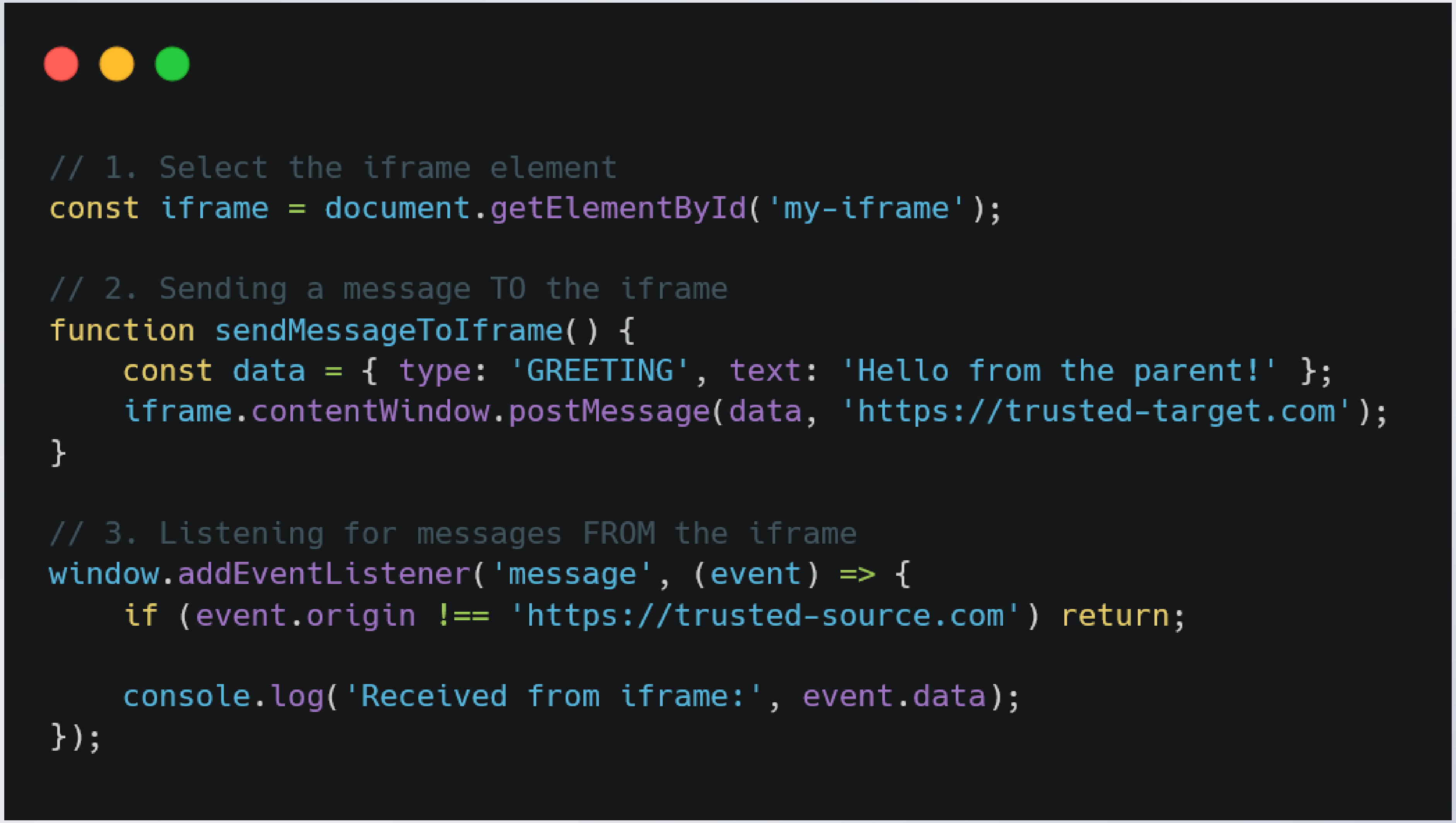
- Mitigation for tabnabbing attacks
- HTTP response header that forces a *Navigable* into a new BCG for top-level Browsing Contexts
- Values:
  - *unsafe-none*: permits sharing BCG with any document
  - *same-origin*: restricts BCG to same-origin and requires the same COOP setting
  - *same-origin-allow-popups*: Same as *same-origin*, but can open documents in the same BCG if they have a COOP value of *unsafe-none*
  - *noopener-allow-popups*: always open in new BCG except when opened by navigating from a document that also has *noopener-allow-popups*
- Result when applied: *window.opener* === *null*

DBG App Test Finding
Cross-Origin Isolation Not Implemented

Window.postMessage

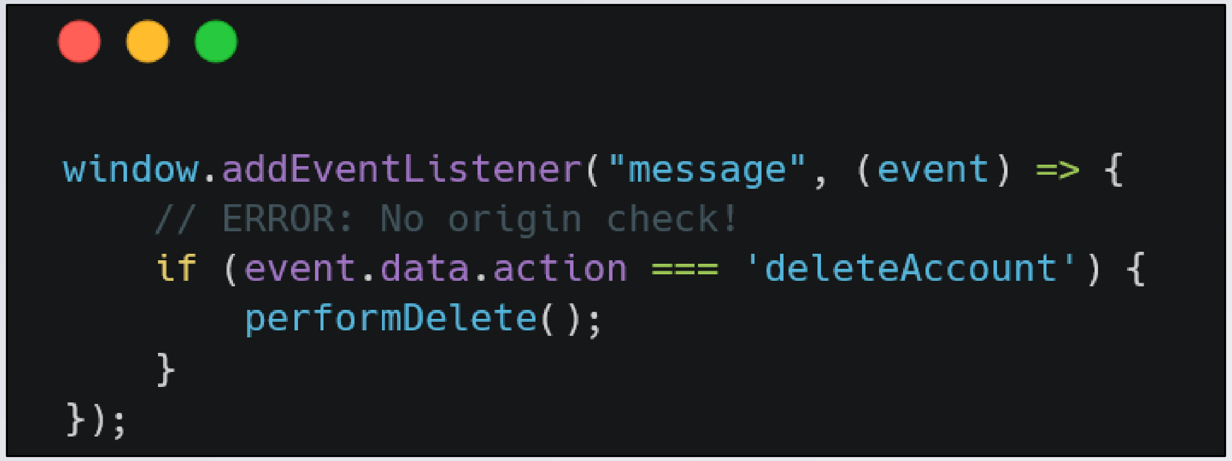
- The preferred method to enable Cross-Origin communication between Window objects
- Two pages must belong to the same Browsing Context Group:
  - Either one page opens another, or
  - One page embeds another (via <*iframe*>)
- Utilizes:
  - *Window.postMessage()* to send messages
  - *Window.addEventListener()* to receive messages

Window.postMessage

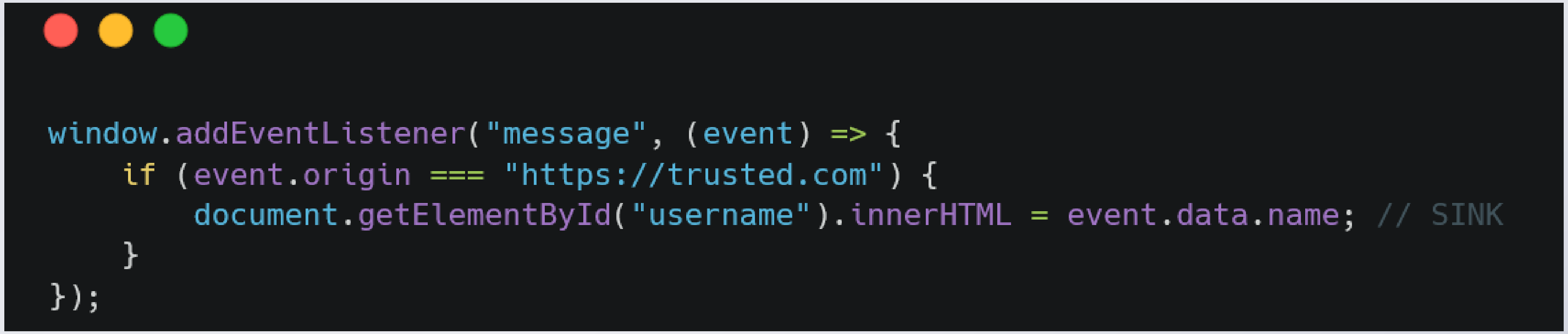


```javascript
// 1. Select the iframe element
const iframe = document.getElementById('my-iframe');

// 2. Sending a message TO the iframe
function sendMessageToIframe() {
    const data = { type: 'GREETING', text: 'Hello from the parent!' };
    iframe.contentWindow.postMessage(data, 'https://trusted-target.com');
}

// 3. Listening for messages FROM the iframe
window.addEventListener('message', (event) => {
    if (event.origin !== 'https://trusted-source.com') return;

    console.log('Received from iframe:', event.data);
});
```

postMessage: Lack of Origin Validation



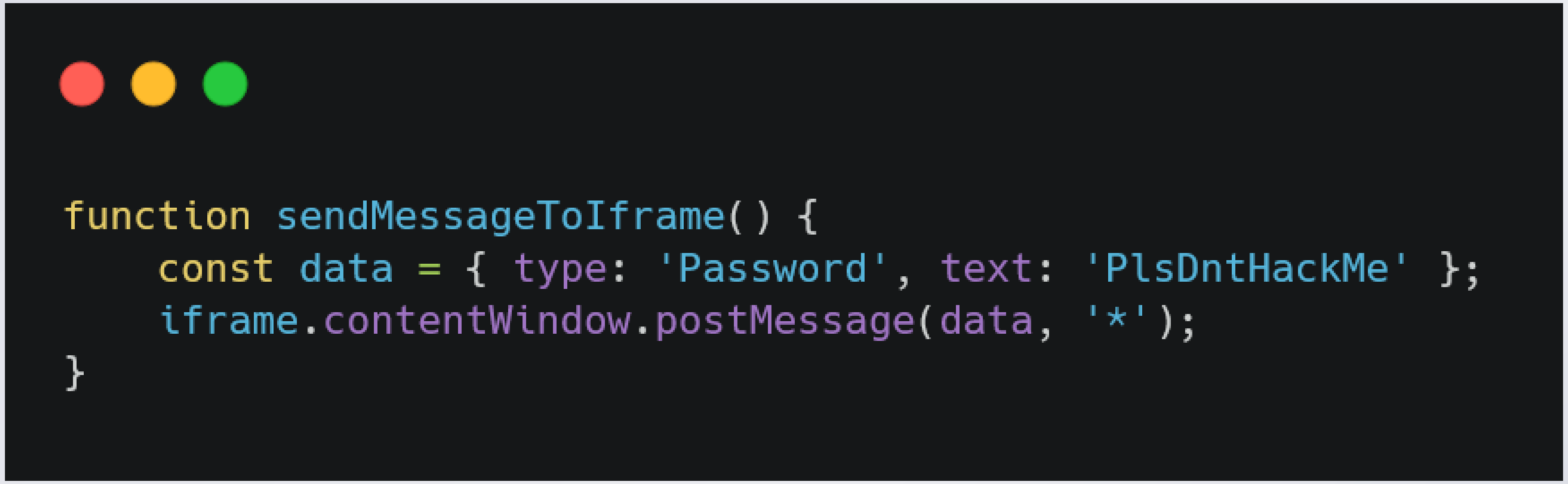
```
window.addEventListener("message", (event) => {
    // ERROR: No origin check!
    if (event.data.action === 'deleteAccount') {
        performDelete();
    }
});
```



```
window.addEventListener("message", (event) => {
    if (event.origin === "https://trusted.com") {
        document.getElementById("username").innerHTML = event.data.name; // SINK
    }
});
```

DBG App Test Finding
Web Messaging Handler Missing Origin Validation

postMessage: Sensitive Data Transmission



```
function sendMessageToIframe() {
    const data = { type: 'Password', text: 'PlsDntHackMe' };
    iframe.contentWindow.postMessage(data, '*');
}
```

DBG App Test Finding
Insecure Web Message Sender

COOP and postMessage

- When a cross-origin app opens another in a new window, strict COOP will sever *window* references
- *postMessage* requires a valid reference to a *window*
- Therefore, COOP breaks *postMessage* between windows, but..
- COOP works on top-level documents and does not impede parent-child relationships
- Therefore, a parent can embed an iframe and use *postMessage*

Embedded Contexts (iframes)

- An iframe (Inline Frame) is a nested browsing context
- Each iframe has:
  - Its own browsing context
  - Its own document
  - Its own DOM
  - Its own Origin
- An iframe may share a Browsing Context Group!
- Window references via:
  - *window.parent*, *window.top*, *window.frames[]*
- Side note: *window.length!?*

iframe Source Content

External URL (src) | Origin: Matches Source

```
<iframe src="https://example.com"></iframe>
```

Inline HTML (srcdoc) | Origin: Parent document

```
<iframe srcdoc="<h1>Hello</h1><p>Inline content.</p>"></iframe>
```

Data URI (src with Scheme) | Origin: Unique/Opaque

```
<iframe src="data:text/html,<h1>Data URI</h1>"></iframe>
```

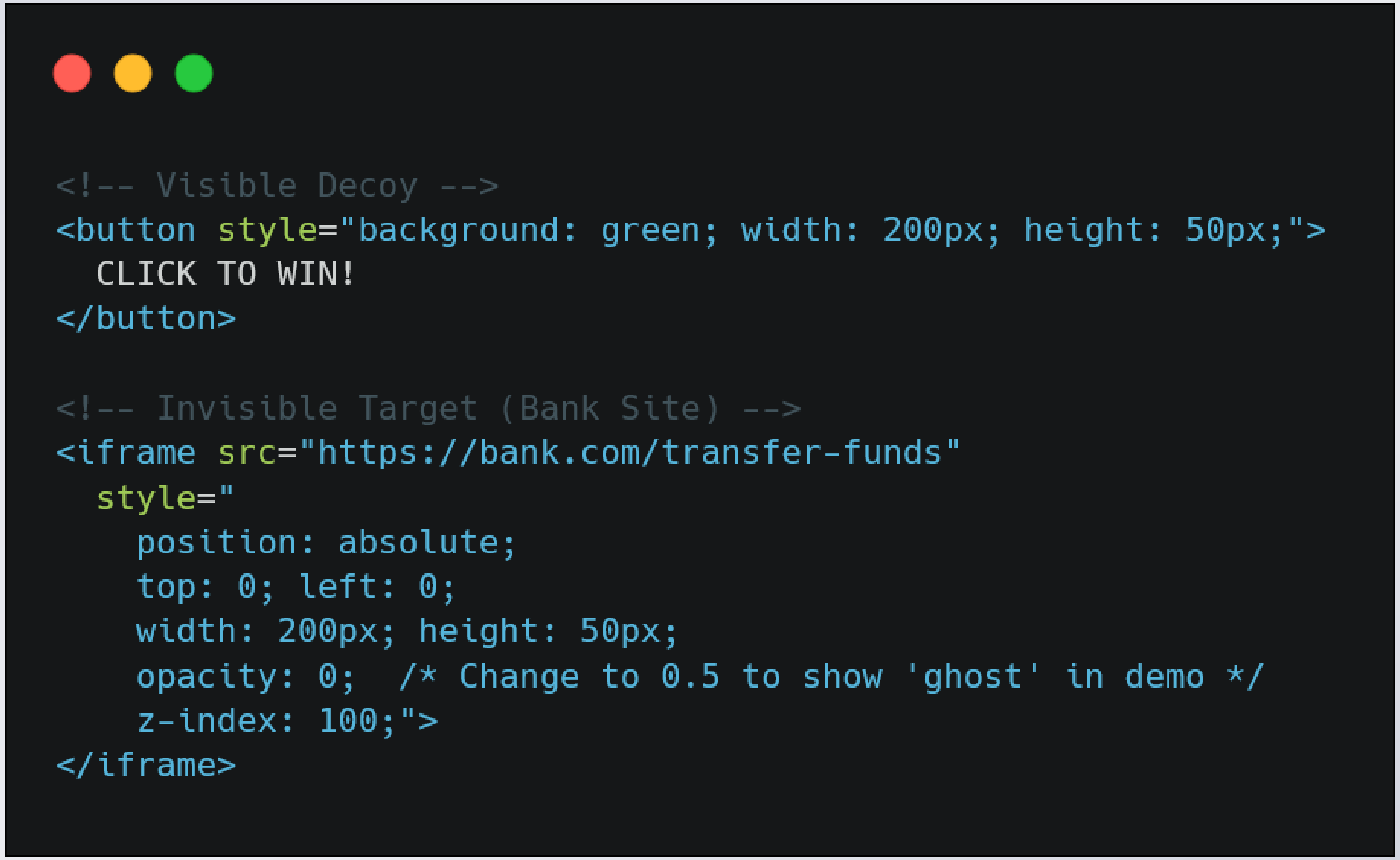
Blank / Scripted | Origin: Parent document

```
<iframe id="myFrame"></iframe>
<script>
  const doc = document.getElementById('myFrame').contentDocument;
  doc.body.innerHTML = "<h1>Injected via JS</h1>";
</script>
```

Clickjacking (UI Redressing)

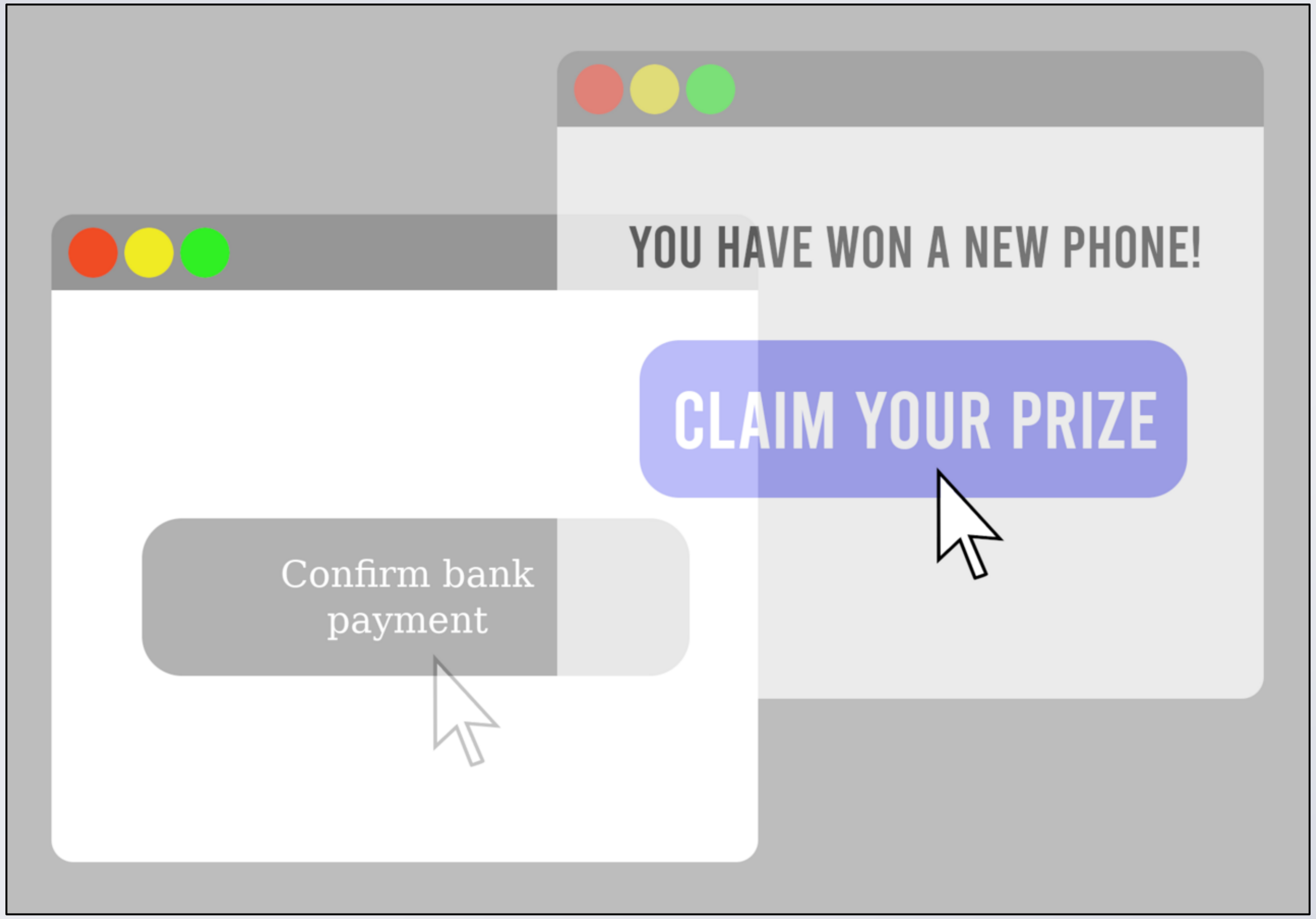
- Class of attacks that tricks users (deception) into interaction
- The UI is manipulated to hide a malicious action, typically against a trusted application
- Similar to CSRF, the target trusted application cannot identify the action as illegitimate (unintended)
- Historically, more methods existing permitting attackers to conduct effective Clickjacking attacks
- Such attacks often require embedded the target application in an <*iframe*>

Simple Clickjacking Example: Invisible Overlay



```html
<!-- Visible Decoy -->
<button style="background: green; width: 200px; height: 50px;">
  CLICK TO WIN!
</button>

<!-- Invisible Target (Bank Site) -->
<iframe src="https://bank.com/transfer-funds"
  style="
    position: absolute;
    top: 0; left: 0;
    width: 200px; height: 50px;
    opacity: 0;  /* Change to 0.5 to show 'ghost' in demo */
    z-index: 100;">
</iframe>
```

Simple Clickjacking Example: Invisible Overlay



By Lord Belbury - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=107229444

Clickjacking Mitigation

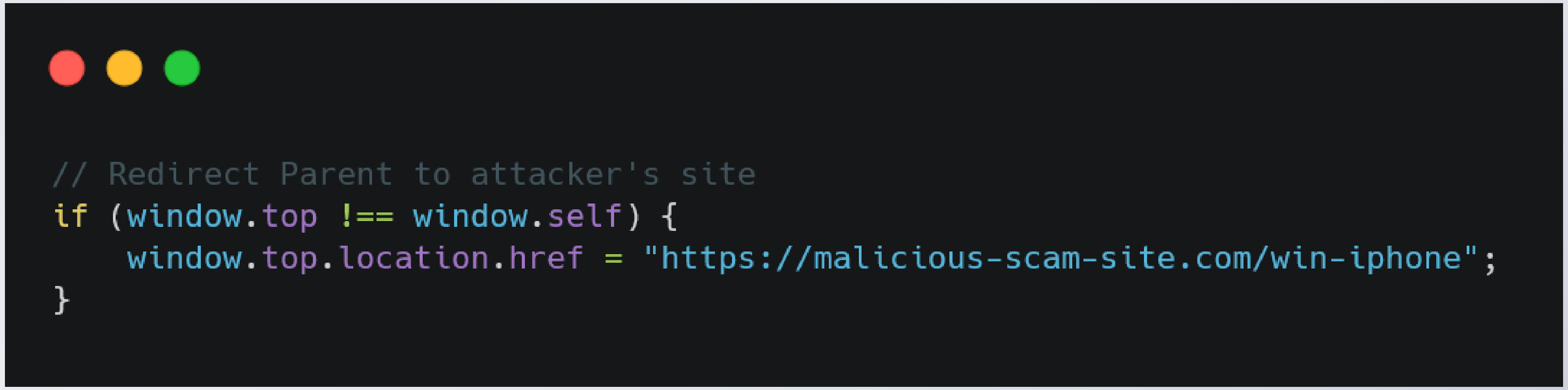
- The primary mitigation to clickjacking attacks using framing techniques is to restrict framing of your document
- HTTP response header *X-Frame-Options* set to *DENY* or *SAMEORIGIN*
  - This header is now deprecated but still widely supported by browsers
- Modern alternative: *Content-Security-Policy frame-ancestors* directive
  - We will discuss Content-Security-Policy (CSP) later in the series

DBG App Test Finding
Missing Protections Against Page Framing

DBG App Test Finding
Unrestricted Page Framing

Attacks From Within: Embedded iframes

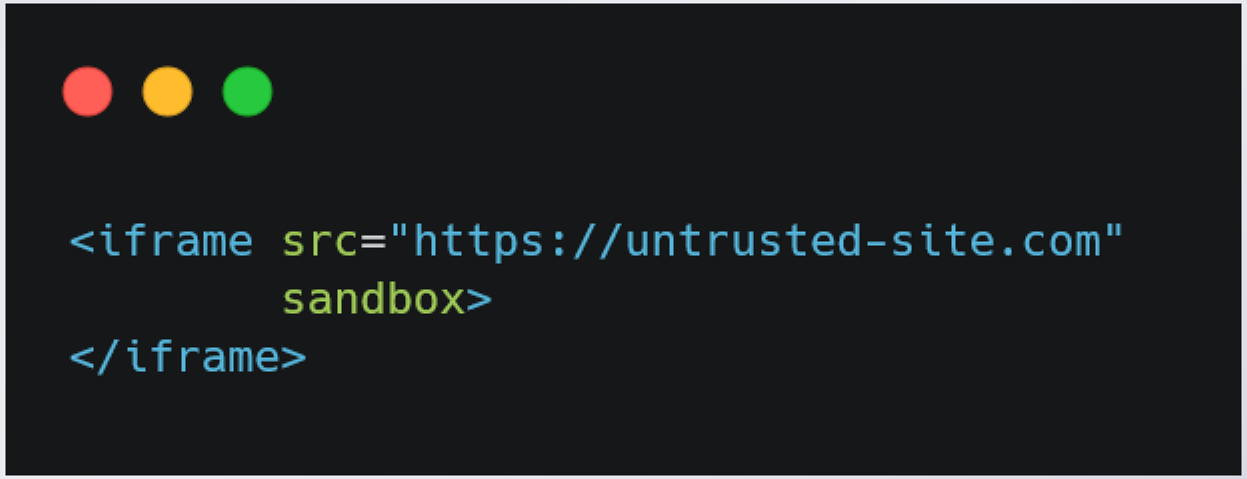
- **Embedded iframes have various ways to interact with their parent**
- **Same-Origin iframes have extensive capabilities**
- **Cross-Origin iframes are much more limited, but can still:**
  - **Can open popups/tabs via *window.open()***
  - **Can attempt top-level navigation via *window.top.location***
  - **Can trigger modals (*alert()*, *confirm()*, *prompt()*)**
  - **Can steal focus and deceive users**



```
// Redirect Parent to attacker's site
if (window.top !== window.self) {
    window.top.location.href = "https://malicious-scam-site.com/win-iphone";
}
```

The iframe sandbox Attribute

- The *sandbox* attribute can apply additional restrictions for the content in an iframe
- Adding the empty *sandbox* attribute applies ALL restrictions
- In particular, *sandbox* forces the iframe to a unique Origin (*opaque Origin)*



```
<iframe src="https://untrusted-site.com"
        sandbox>
</iframe>
```

The sandbox Attribute: Restrictions

Full restrictions include:

- **Preventing JavaScript execution**
- **Preventing navigation**
- **Assigning a unique Origin**
- **Preventing Form submission**
- **Preventing data storage/access**
- **Preventing media and hardware capabilities**

Capabilities can be selectively permitted:

- ***sandbox="allow-scripts allow-forms ..."***

DBG App Test Finding
Inline Frame Content Restriction Not Implemented

The sandbox Attribute: Special Conditions

- *sandbox* can be relaxed with ‘allow-same-origin’ to permit the iframe content to possess its natural Origin rather than an Opaque Origin
  - Scenario: authenticated (with cookies) social media interactions
- Sandboxed iframes with both ‘allow-scripts’ and ‘allow-same-origin’ are strongly discouraged when the content is Same-Origin
  - The embedded iframe can execute JS and interact with the parent’s DOM
  - The same potential risk is not posed Cross-Origin (‘allow-same-origin’ does not mean “make the iframe Same-Origin to the parent”)

OWASP ASVS 5.0

V3 Web Frontend Security: V3.4 Browser Security Mechanism Headers

- 3.4.6: Implement CSP: *frame-ancestors*
- 3.4.8: Implement Cross-Origin-Opener-Policy HTTP response header

V3 Web Frontend Security: V3.5 Browser Origin Separation

- 3.5.5: Verify postMessage interfaces validate Origin

https://github.com/OWASP/ASVS/blob/master/5.0/en/0x12-V3-Web-Frontend-Security.md

Security Testing Considerations

- Can abusable pages be framed?
- Do framed pages expand surface area? Are they sandboxed?
- Is postMessage used? Is it secure?
  - Look for listeners/messages
- Can attackers manipulate *window.open* or its targets?
- Does the application permit user-hosted content? How is it hosted and served?