SWAG: What are Web Developers Doing about Security?

Dan Appelquist, Samsung Open Source Group

Photo credit: Parsoa Khorsand (unsplash)

Intro: Daniel Appelquist

SAMSUNG

Open Source Strategist at Samsung Open Source Group

W3C AB

Co-Chair of the W3C Advisory Board

Co-Chair of the OpenSSF Global Cybersecurity Policy WG

Web security is a special case

The web platform is the most widely deployed application runtime in the world.

Yet, web is often underrepresented in broader software security efforts.

We're talking about the browser-based web.

What we talk about when we talk about web security

Cross-Site Scripting (XSS)

- Injection of malicious scripts that run in the user's browser.
- Still one of the most persistent web vulnerabilities.

Cross-Site Request Forgery (CSRF)

- Tricking a user's browser into submitting unwanted requests to a site where they're authenticated.

Clickjacking

- UI redressing attacks that trick users into clicking on invisible or disguised elements.

Phishing & Typosquatting

- Leveraging browser trust indicators or domain name tricks (IDNs, subdomains) to deceive users.

Supply Chain Injection (& Typosquatting)

- Compromised npm packages, CDNs, or third-party widgets injecting malicious code into trusted web apps.

Credential Stuffing & Session Hijacking

- Reuse of stolen credentials; exploiting weak session tokens or leaking tokens via URLs or referrers.

Man-in-the-Middle Attacks

- Intercepting or altering communication when HTTPS is misconfigured or missing

Fake Consent / Permission Abuse

- Game users into granting dangerous permissions (e.g., camera, notifications, clipboard) via deceptive UI.

Insecure Storage / Data Leakage

- Sensitive data stored in localStorage, cookies, or similar and exposed to cross-site scripts.

Spectre / Meltdown Attacks

- CPU-level attacks that use high resolution timers in conjunction with shared array buffers to leak data between contexts.

“It should be safe to visit a web page.” – Web Platform Design Principles §1.2

Every site is potentially untrusted.

Web browsers have a “duty of protection” - against 1st party code.

Web applications also have a duty of protection - against untrusted 3rd party code!

How Web Security Is Different

Web developers operate in an environment unlike traditional software dev:

- The attacker has direct access to the runtime. The code runs in the user's browser—your threat surface is the internet.
- No centralized runtime: Every browser is slightly different; updates are automatic but variable.
- Highly dynamic content models: Includes third-party scripts, CDNs, embedded widgets.
- Mixed trust boundaries: First-party, third-party, same-origin, cross-origin — it's a lot.
- Developer workflow is different: npm, frameworks, build tools — security concerns get abstracted away.

One example: a front-end developer installing a React carousel component that silently includes a malicious script.

SWAG Origin Story

The Secure the Web Forward workshop (2023)

Full report & videos available: https://www.w3.org/2023/03/secure-the-web-forward/

Key takeaways:

- Fragmentation in web security guidance.
- Misalignment between browser capabilities and developer understanding.
- Lack of tailored security practices for modern web development workflows.

Formation of the SWAG Community Group (under W3C), with OpenSSF coordination.

Small but dedicated band.

More about the findings of the workshop

Supply Chain Security: Adoption software supply chain security best practices are key to managing web application dependencies and integrity.

JavaScript Isolation: New tools and standards aim to safely sandbox third-party code, but require further work.

Cookie Security Reform: The deprecation of third-party cookies presents an opportunity to redesign cookie behavior for better security by default.

Developer-Centric Documentation: Developers face complexity securing apps; clearer, actionable documentation and best practices are urgently needed.

Enter the SWAG Group

Secure Web Application Guidelines

Run as a W3C Community Group (open), working with OpenSSF Best Practices wg

Fill the gap between browser security features and actionable guidance for developers.

Deliverables so far:

- Core guidelines document
- Helping to write and revise security documentation on MDN

W3C Community Group Draft Report

TABLE OF CONTENTS

Secure Web Application Guidelines (SWAG)

Draft Community Group Report 28 April 2026

Latest published version:
https://w3c-cg.github.io/swag/docs/swag.html

Latest editor's draft:
https://w3c-cg.github.io/swag/docs/swag.html

Editors:
Daniel Appelquist (Samsung Electronics)
William Bamberg (Open Web Docs)
Florian Scholz (Open Web Docs)

Feedback:
GitHub w3c-cg/swag (pull requests, new issue, open issues)

Abstract

Secure Web Application Guidelines (SWAG) provide recommendations on how to develop more secure web applications. SWAG are best practices you can follow to reduce the risk of introducing security vulnerabilities into your web application, or help you respond to vulnerabilities. In this document, we collect web platform features that you can implement in a web application and practices you can follow to help mitigate or respond to various attacks.

The SWAG document is developed in context of the Best Practices for OSS Developers Working Group which is dedicated to raising awareness and education of secure code best practices for open source developers. SWAG presents security guidelines specific to web developers.

https://w3c-cg.github.io/swag/docs/swag.html

§ 3.2 Require strong authentication for project maintainers

Supply chain attacks often target project maintainers: by gaining control of a maintainer's account an attacker can introduce malicious code or ship a malicious update of the project.

This means that a project must use a strong authentication method for maintainer accounts. In particular, attackers often use phishing to gain control of maintainer accounts, so authentication methods should aim to be phishing-resistant.

Requiring multi-factor authentication for project maintainers makes phishing more difficult but, depending on the authentication methods used, does not always prevent it. Passkeys provide the strongest defense against phishing attacks.

Meeting developers where they are (MDN).

Cross-site leaks (XS-Leaks)

Cross-site leaks (also called XS-Leaks) are a class of attack in which an attacker's site can derive information about the target site, or about the user's relationship with the target site, by using web platform APIs that enable sites to interact with one another. The information leaked could include, for example:

- Whether the user has visited the target site.
- Whether the user is logged into the target site.
- What the user's ID on the site is.
- What the user has recently searched for on the site.

This might seem to be a much less damaging problem than, for example, a cross-site scripting attack, but it can still have serious consequences for users. For example:

- A user might have accounts on websites that they don't want to make public. Leaking this information to an attacker could expose them to extortion or retaliation from an oppressive government (for example, against a user seeking information about specific medical procedures).
- Knowing a user has an account on a site, especially if their user ID can be determined, can make a subsequent phishing attack much more convincing.

Unlike other attacks such as XSS or Clickjacking, cross-site leaks are not a single technique. Instead, they are a term for a whole class of attack which exploit weaknesses in the ways that browsers isolate websites from each other.

In this guide we will not attempt to describe every cross-site leak attack and defense. Instead, we'll

The Guidelines

“Use HTTPS”

“Implement a content security policy (CSP)”

“Set the SameSite attribute on sensitive cookies”

“Use Fetch metadata to defend against cross-site attacks”

“Validate and sanitize user input”

“Implement restrictions on framing”

“Limit the use of third-party scripts and resources” (using Subresource integrity)

“Validate and sanitize all server-side requests” (protects against server-side request forgery)

“Evaluate the security metrics of open source libraries” (e.g. using Scorecard)

“Limit the use of potentially unsafe DOM APIs to minimize injection risk and ensure compatibility with Trusted Types”

+ all the regular software security best practices (e.g. use 2fa in your SCM, etc… etc…)

https://github.com/w3c-cg/swag/blob/main/docs/security_guidelines.md

The Survey: Early 2026

Who Responded?

108 respondents

61% had more than 10 years of web development experience

62% rated their general web development knowledge as "expert"

70% rated their web security knowledge as "intermediate", 18% as "expert"

77% of respondents told us that they were responsible for security themselves

← web developers are not delegating security to security specialists

Web Security Features

Do you implement any mechanism to control whether **third-party** sites can **embed** your site as an iframe?

When you set **cookies** that contain a user's login credentials (such as a session ID), do you set the **SameSite attribute** to "Lax" or "Strict", to control whether the cookie is included in cross-site requests?

When you set **cookies** that contain sensitive user information (such as a session ID), do you set the **Secure attribute** to ensure they are only sent over an encrypted (HTTPS) connection?

When you set cookies, do you use the **HttpOnly attribute** to prevent them from being accessed by client-side JavaScript?

Do you use **Fetch metadata** to control whether certain cross-site requests are allowed?

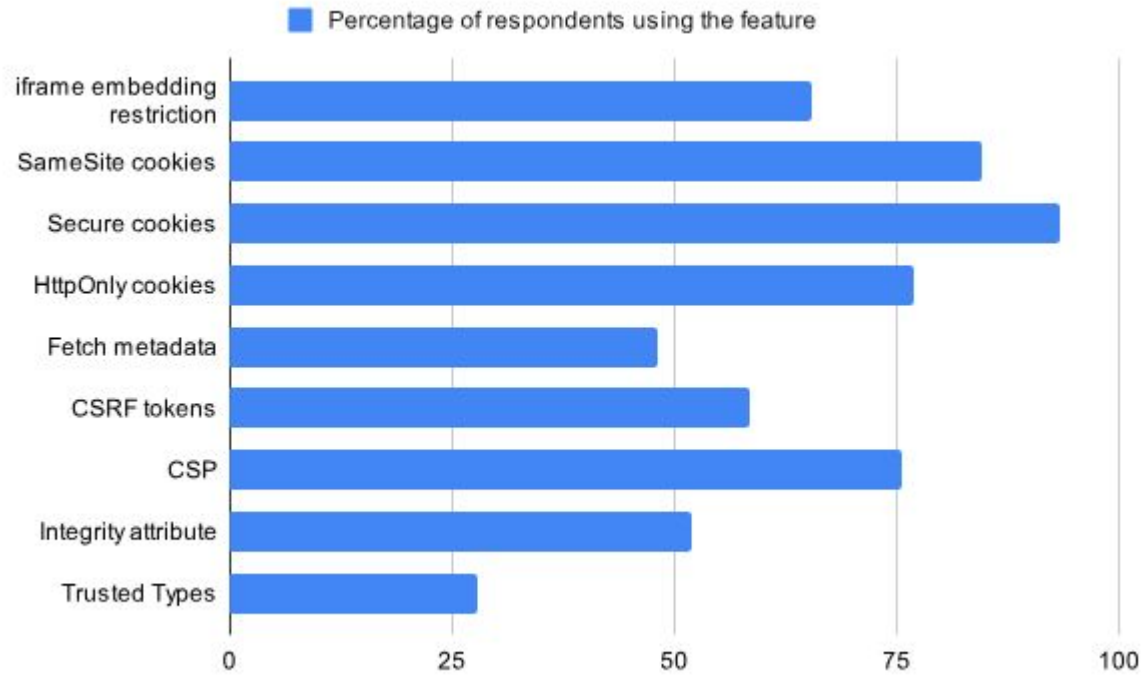
Do you use **CSRF tokens** in forms on your site?

Do you use a **Content-Security-Policy** (CSP) header on your site?

Do you set the **"integrity" attribute on scripts** you load from a third-party site such as a CDN?

Do you enforce the use of **Trusted Types** when passing input into JavaScript injection sinks such as innerHTML?

What did we learn?



HTTP headers

Strict-Transport-Security (HSTS)

Referrer-Policy

X-Content-Type-Options

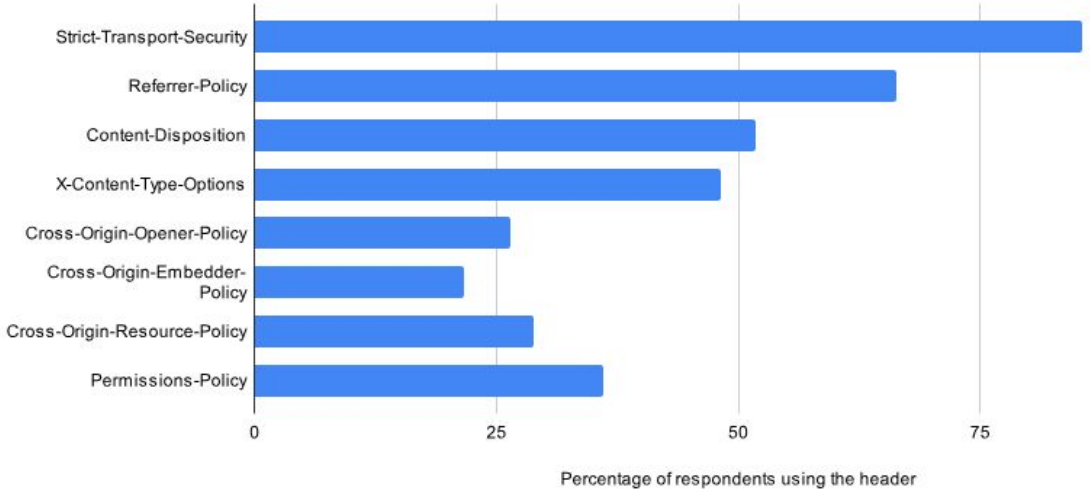
Content-Disposition

Cross-Origin-Embedder-Policy (COEP)

Cross-Origin-Opener-Policy (COOP)

Cross-Origin-Resource-Policy (CORP)

Permissions-Policy



Findings

Web developers need security documentation

Our respondents were almost all generalist web developers: less than 5% described themselves as security specialists, and only 18% described their web security knowledge as "expert".

There are certain security topics that are under-used

This led to additional work on web security documentation, such as major updates to MDN pages on Trusted Types, Subresource Integrity, and Fetch Metadata. This work was carried out by **Open Web Docs**, funded by a **Sovereign Tech Agency** grant.

What's Next for SWAG

We are finalizing the SWAG recommandations document

(We're still very interested in your contributions and PRs)

Finish review of additional security articles on MDN.

Work with the **OpenSSF Best Practices Working Group** to promote this work

Considering spinning down SWAG, and handing over maintenance of this work to the W3C Documentation Community Group (Docs CG).

Experts Agree: Sometimes Web Developers DO have To worry about the CRA

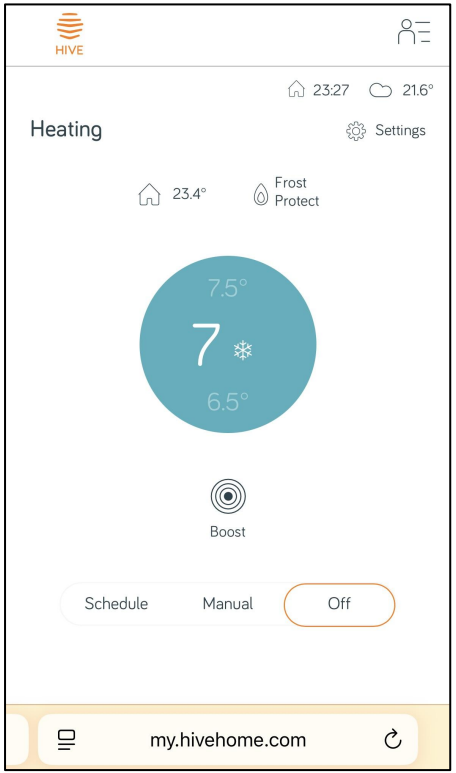
Web applications that are part of packaged apps

- Ever check in for a flight?

Web applications that are part of a system, along with products with digital elements

- Home automation

JS libraries that you want any of the above to use



Thank you!

Dan Appelquist, Samsung Open Source Group

@torgo@mastodon.social (#Fediverse)

@torgo.com (BlueSky / EuroSky)

Guidelines:
https://w3c-cg.github.io/swag/docs/swag.html

SWAG On GitHub https://github.com/w3c-cg/swag

To join SWAG: https://www.w3.org/community/swag/

… or join the **OpenSSF Best Practices** working group!

Credit: Leon Appelquist