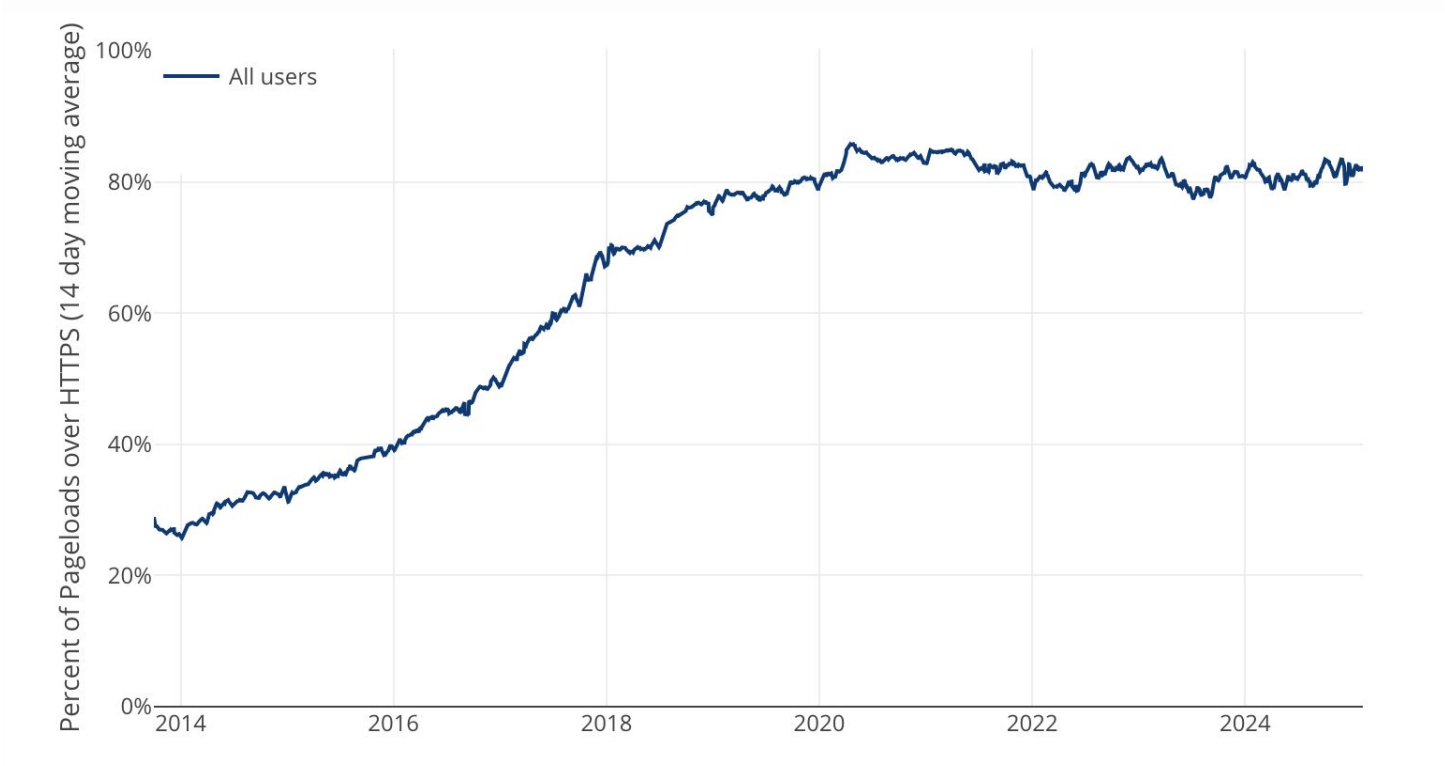
With Carrots and Sticks

Can the Browser Handle Web Security?

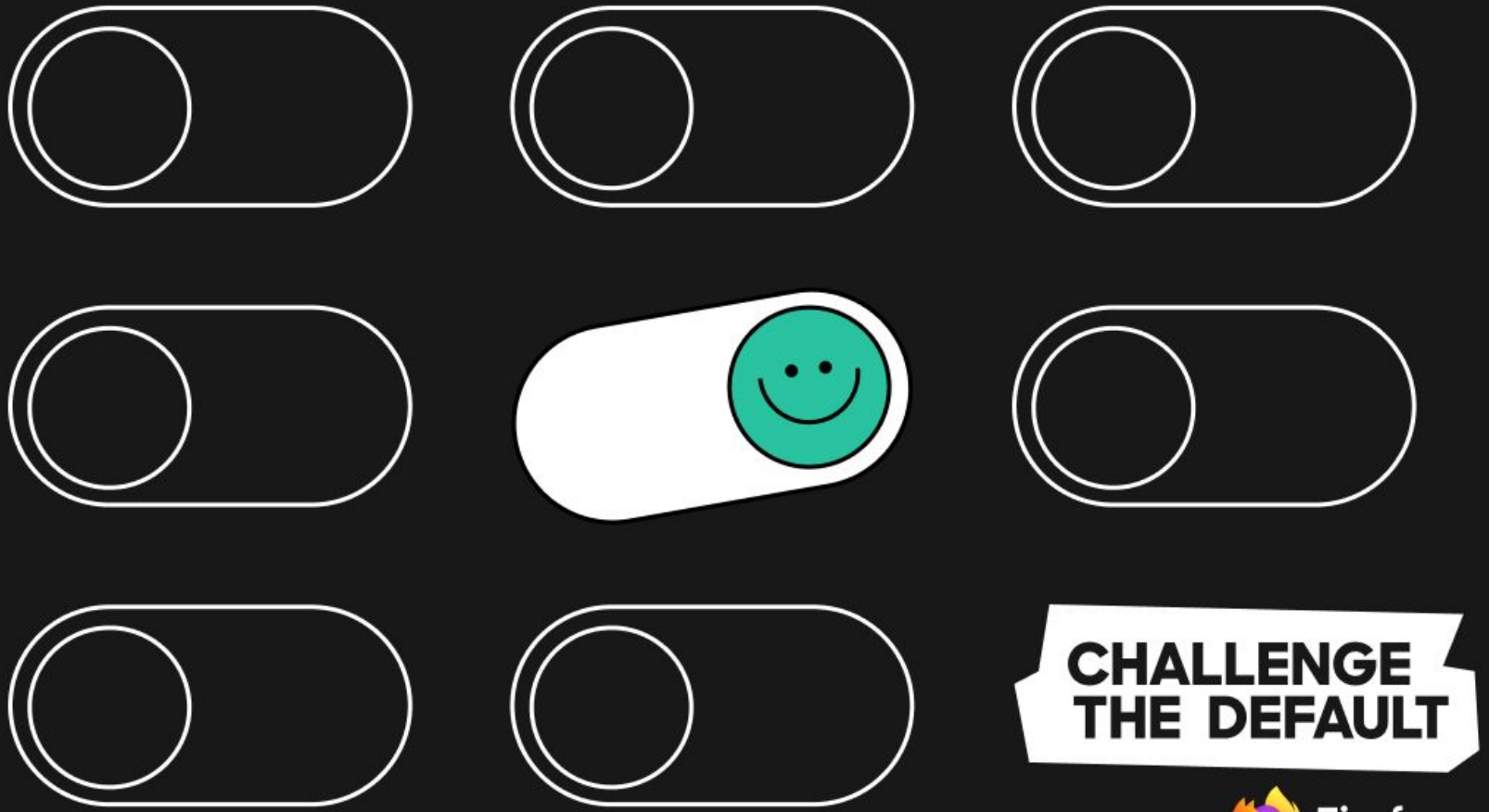
Motivation


Percent of Pageloads over HTTPS (14 day moving average)
100%
80%
60%
40%
20%
0%
All users
2014
2016
2018
2020
2022
2024

Motivation

2024 CWE Top 25 Most Dangerous Software Weaknesses

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) ▲


CHALLENGE
THE DEFAULT
Firefox

Obstacles

Removing bad APIs: in a distributed system

1. Write a new, improved function
2. Disallow new code to use the bad function
3. Rewrite all existing code to use the new API
4. Remove the bad API.

Removing bad APIs from the web

1. Write a new, improved function
2. Deprecate
3. Allow websites to opt-out of existing behavior
4. Wait.
5. Make the bad API opt-in (or fully remove the bad API).

W3C Working Draft

HTML Design Principles

W3C Working Draft

When considering changes to legacy features or behavior...

... the benefit of the proposed change should be weighed against the likely **cost of breaking content**.

W3C Working Draft

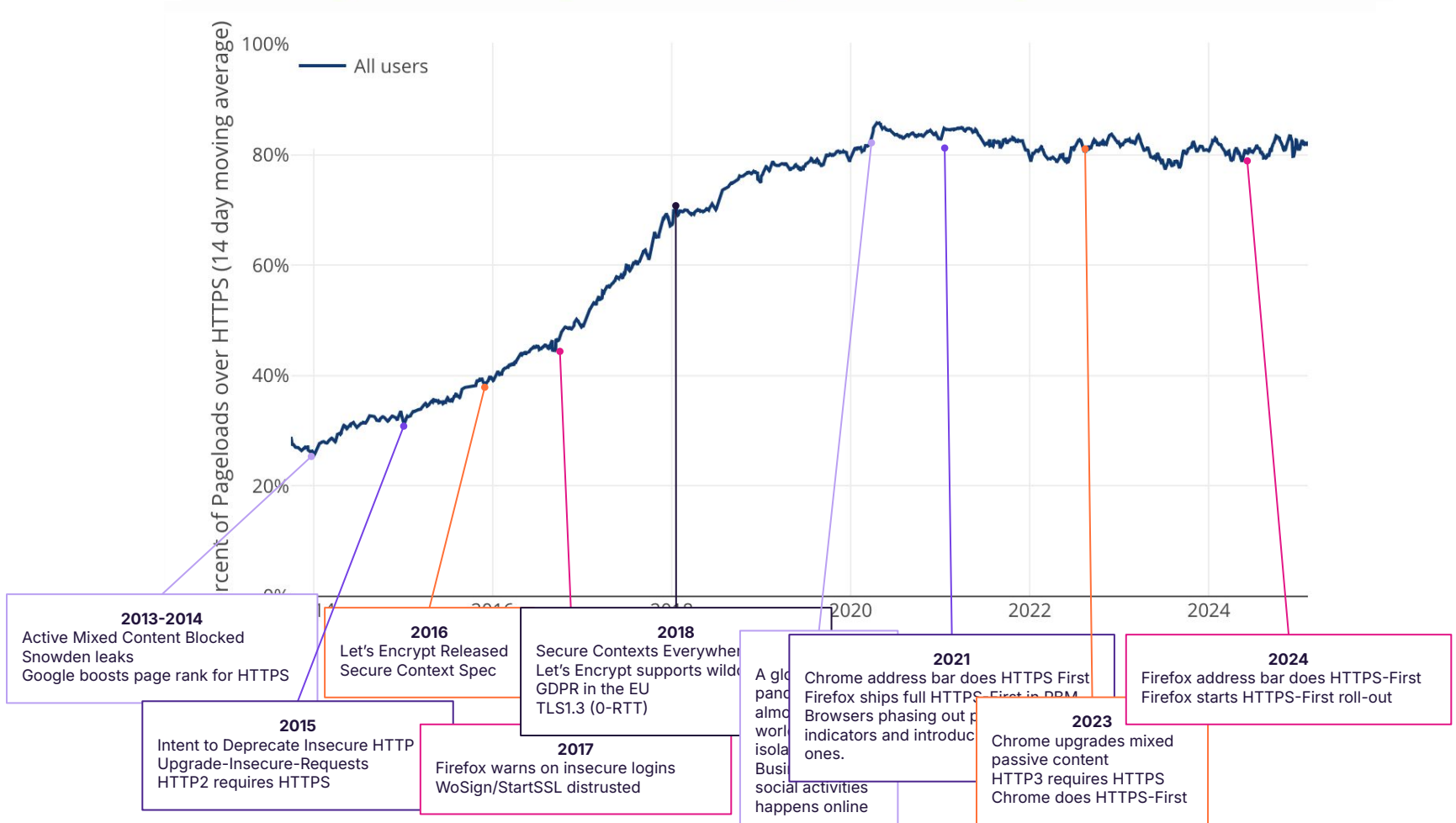
HTML Design Principles

W3C Working Draft

In case of conflict, **consider users over authors over implementors over specifiers over theoretical purity**.

In other words costs or difficulties to the user should be given more weight than costs to authors; which in turn should be given more weight than costs to implementors; [...] Of course, it is preferred to make things better for multiple constituencies at once.

Percentage of Web Pages Loaded by Firefox Using HTTPS



Outlook

What’s left?

Encrypting the un-encryptable

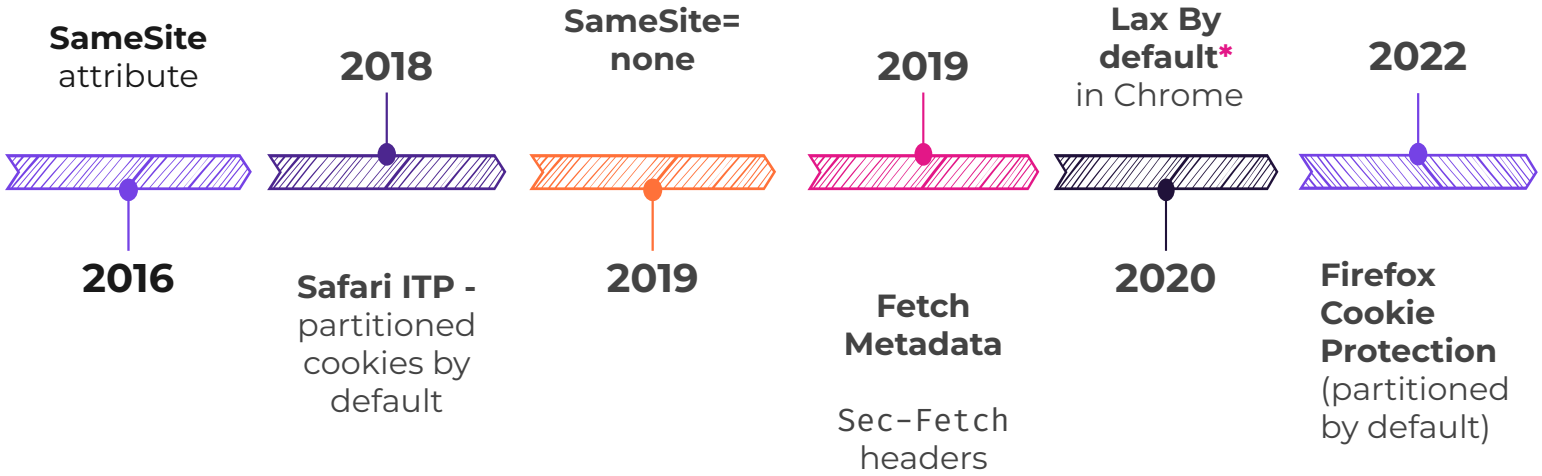
What about local networks?

What about regional differences?

CSRF

Cross-Site Request Forgery

CSRF Interventions



Outlook

What’s left?

Lax By Default

Caution: Chrome's default behavior is slightly more permissive than an explicit `SameSite=Lax`, because it lets sites send some cookies on top-level POST requests. For details, see the blink-dev announcement. This is intended as a temporary mitigation. You'll still need to update your cross-site cookies to `SameSite=None; Secure` as described in the next section.

https://web.dev/articles/samesite-cookies-explained#samesitelax_by_default

What would it take to make

lax-by-default more impactful?

CHIPS & 3rd Party Cookies

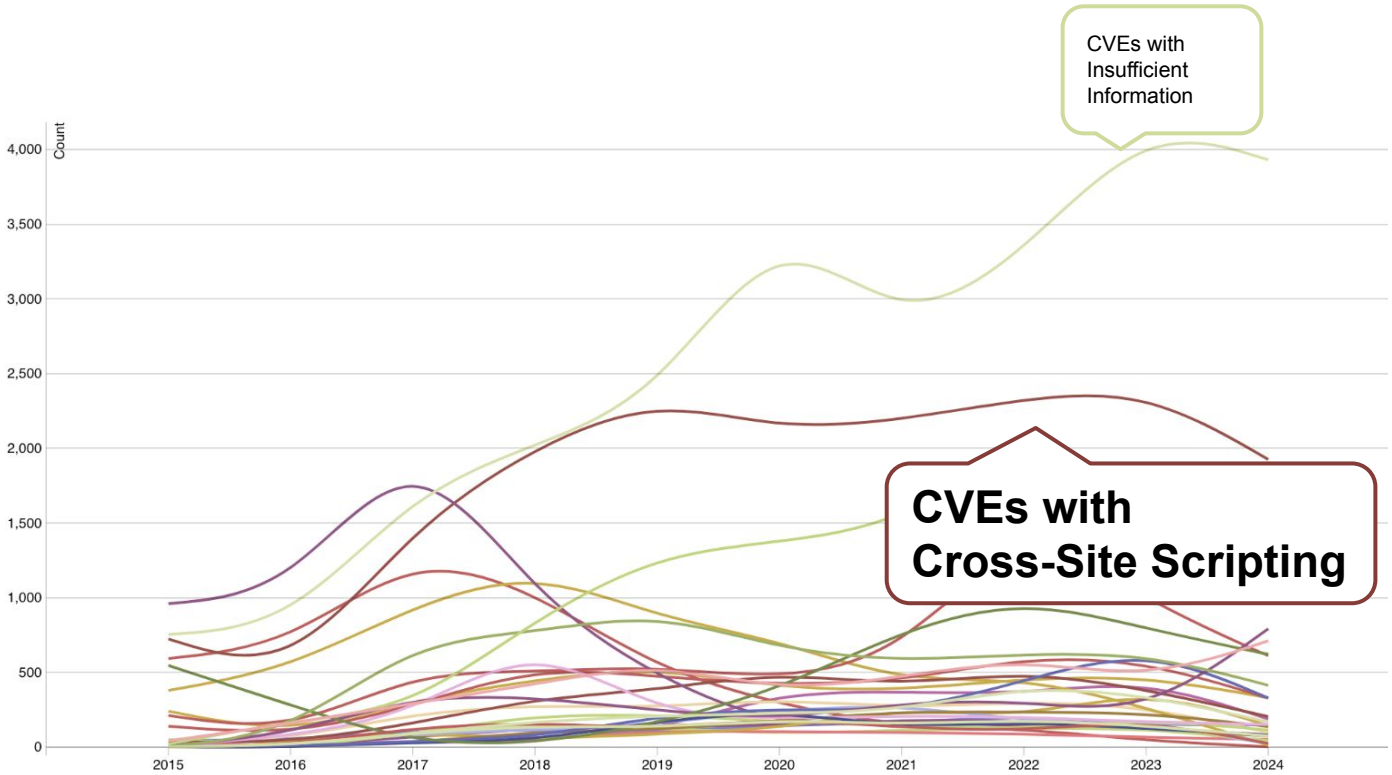
Are Third Party Cookies really going

to be deprecated?

XSS

Cross-Site Scripting

[Source: https://twitter.com/joernchen/status/1086237923652046849]



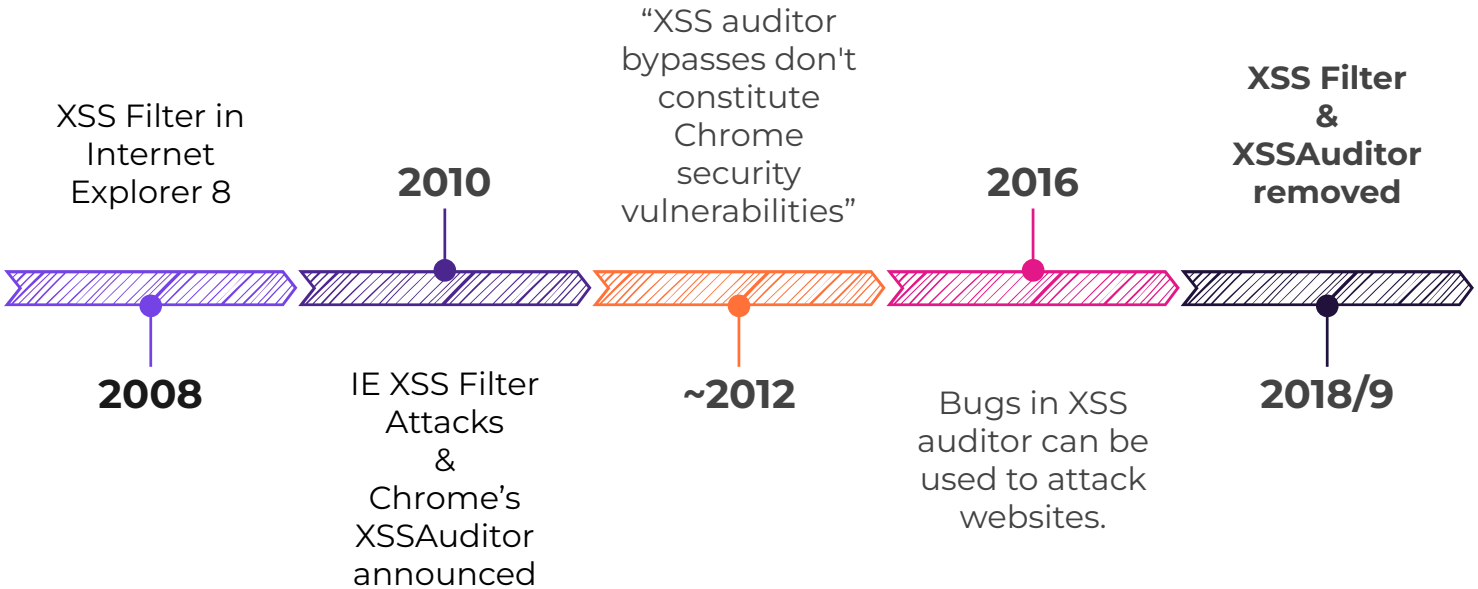
[Source: https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cwe-over-time]

Browser-based XSS defenses

Two case studies

Browser-based XSS ~~defenses~~ mitigations

XSS Filters


XSS Filter in Internet Explorer 8
2008
2010
IE XSS Filter Attacks & Chrome's XSSAuditor announced
"XSS auditor bypasses don't constitute Chrome security vulnerabilities"
~2012
2016
Bugs in XSS auditor can be used to attack websites.
XSS Filter & XSSAuditor removed
2018/9

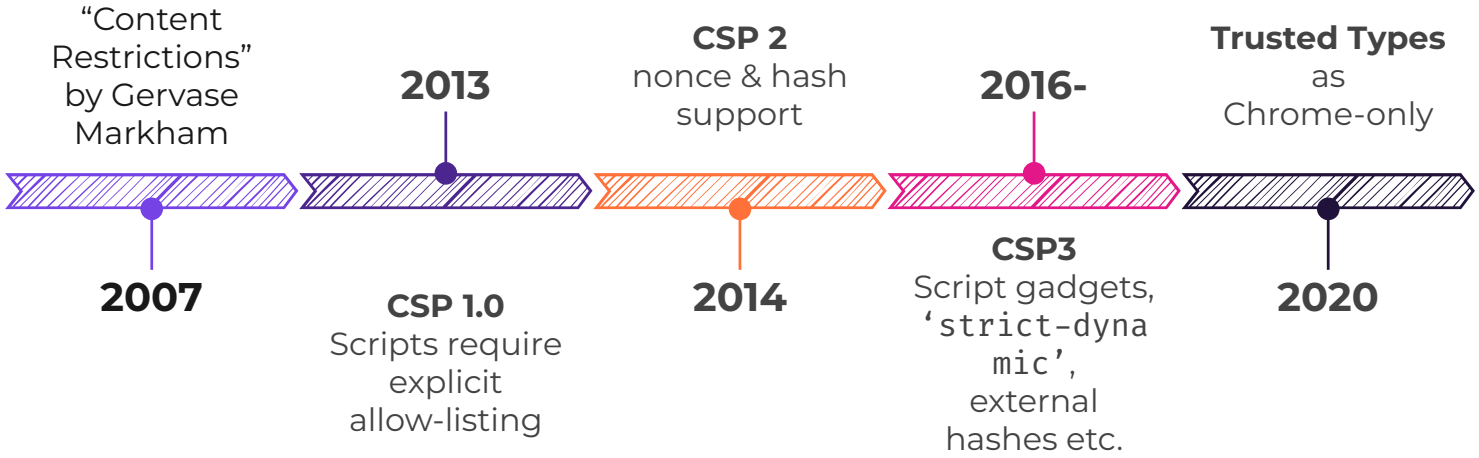
(*2008 - † 2019)

XSS Filters

“Better than RegEx” is not good enough.
(Cf. http://langsec.org/occupy/)

You might cause more harm than good

Content Security Policy (CSP)



*2007 -

Content Security Policy

<20% of websites have a CSP which controls script

90% of CSPs allow inline scripts

https://almanac.httparchive.org/en/2024/security#content-security-policy

Lessons learned

Adoption must be very easy:
Low Complexity

We need *High Compatibility* with existing content

Can we focus on Prevention rather than Mitigation?

Outlook

DOM-based XSS

Trusted Types

Opt-In

Disallowing strings assigned to `innerHTML`.

Requires minting of e.g., a TrustedString object.

Trusted Types

has been highly effective. In the past three years, **for hundreds of complex web applications that are built on Google's hardened and safe-by-design frameworks, we've averaged less than one XSS report per year in total**. As an example, Google Photos was developed on secure-by-design frameworks from the outset, and has had no XSS vulnerabilities discovered

https://static.googleusercontent.com/media/publicpolicy.google/en//resources/google_commitment_secure_by_design_overview.pdf

Sanitizers Today

```
let clean = DOMPurify.sanitize(evil, options);

div.innerHTML = clean;
```

Sanitizer API

```
let clean = DOMPurify.sanitize(evil, options);

div.setHTML(evil);
```

Sanitizer API

~~let clean = DOMPurify.sanitize(evil, options);~~

```
div.setHTML(evil, options);
```

What’s left?

and what would a 🏏 look like for XSS?

Sanitizer API

What if the Sanitizer could be applied globally, with a header?

Such that `innerHTML=` was be implicitly rewritten to auto-sanitize like `setHTML()`?

What if this became opt-out, rather than opt-in?

Sanitizer API

HARDENING | SECURITY INTERNALS

Hardening Firefox against Injection Attacks – The Technical Details

Christoph Kerschbaumer, Tom Ritter and Frederik Braun | July 7, 2020

https://blog.mozilla.org/attack-and-defense/2020/07/07/hardening-firefox-against-injection-attacks-the-technical-details/

Summary

How to deprecate

Lots of various initiatives

Start with the carrots & opt-in security

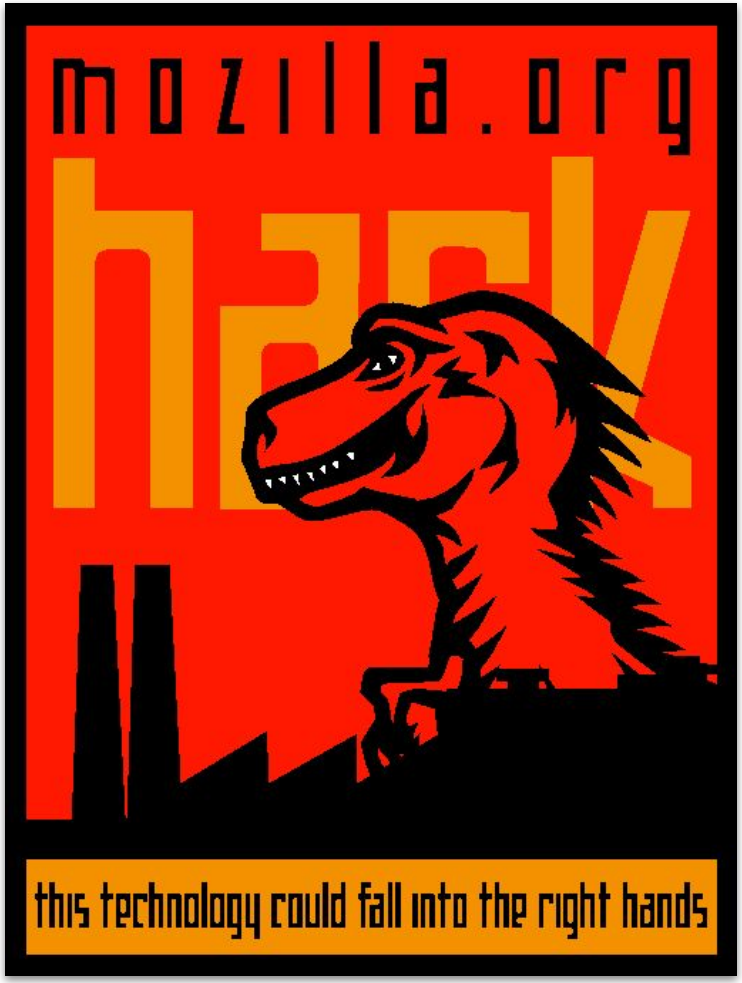
Expect long timelines.

Secure APIs, safer web

Thank you

Questions & Comments

- Matrix
  - @fbraun:mozilla.org
- Fediverse
  - @freddy@security.plumbing
- E-Mail
  - freddy@mozilla.com



Thank you

Questions & Comments

- Matrix
  - @fbraun:mozilla.org
- Fediverse
  - @freddy@security.plumbing
- E-Mail
  - freddy@mozilla.com

Slides