Content Security Policy
The Past, the Present, the Future?

Based on joint work with Sebastian Roth

as well as Tim Barron, Nick Nikiforakis, Stefano Calzavara, Martin Johns, and Marius Musch

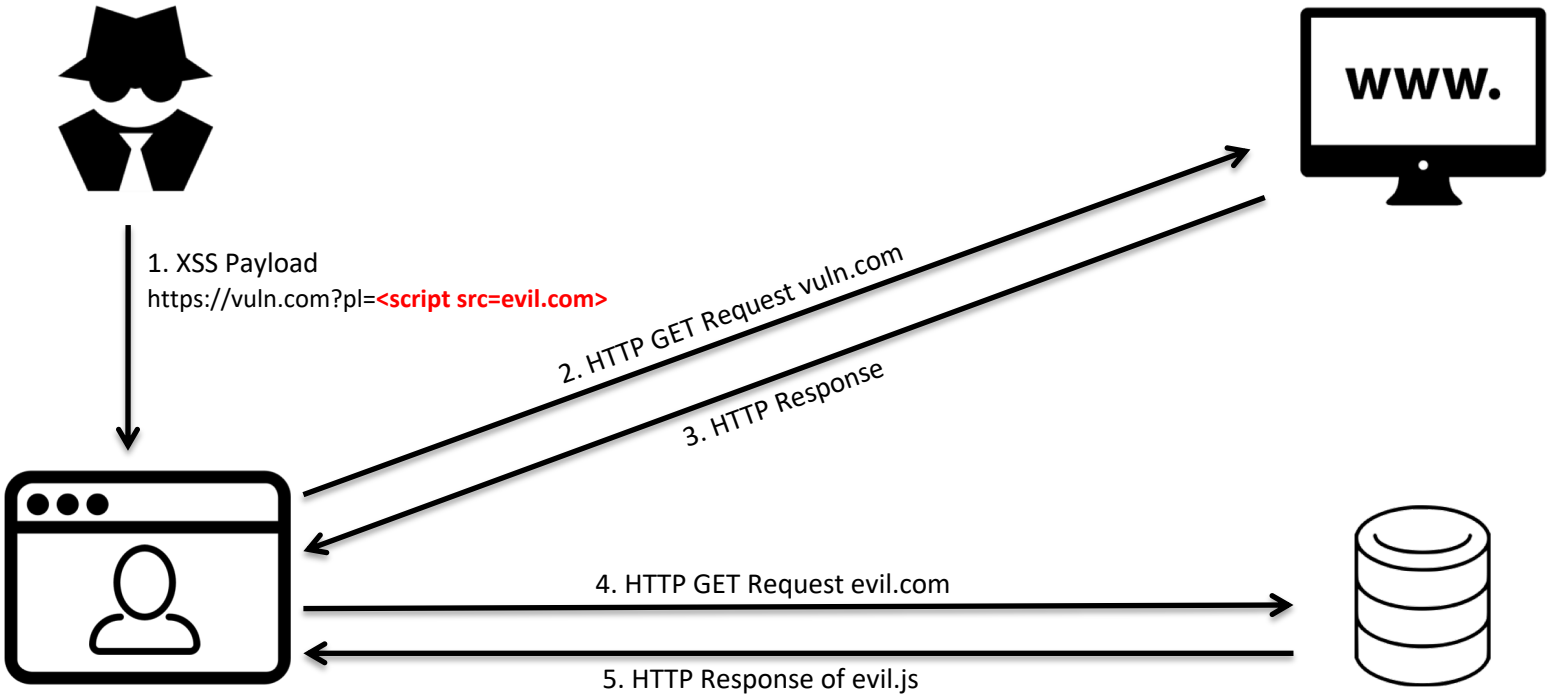
Who is behind these works

Ben Stock

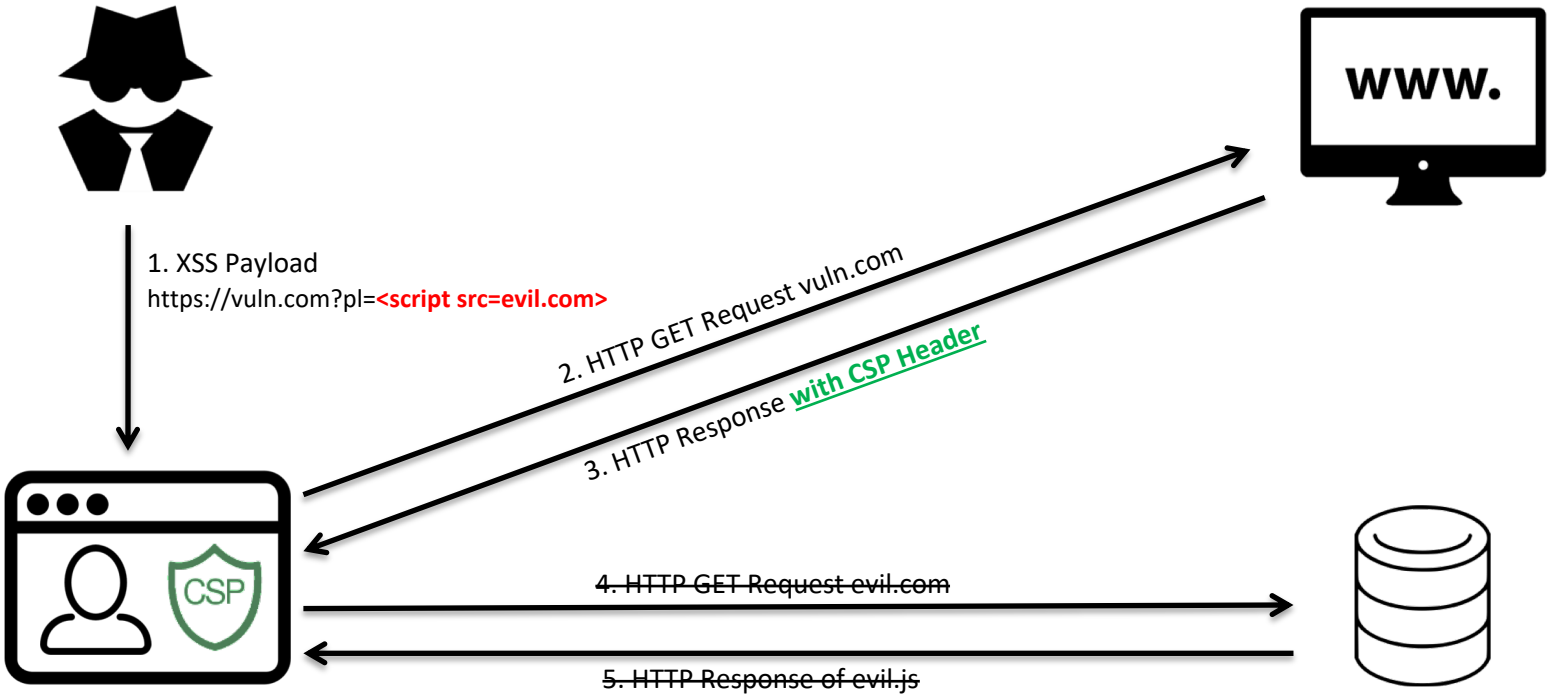
Marius Steffens

Sebastian Roth

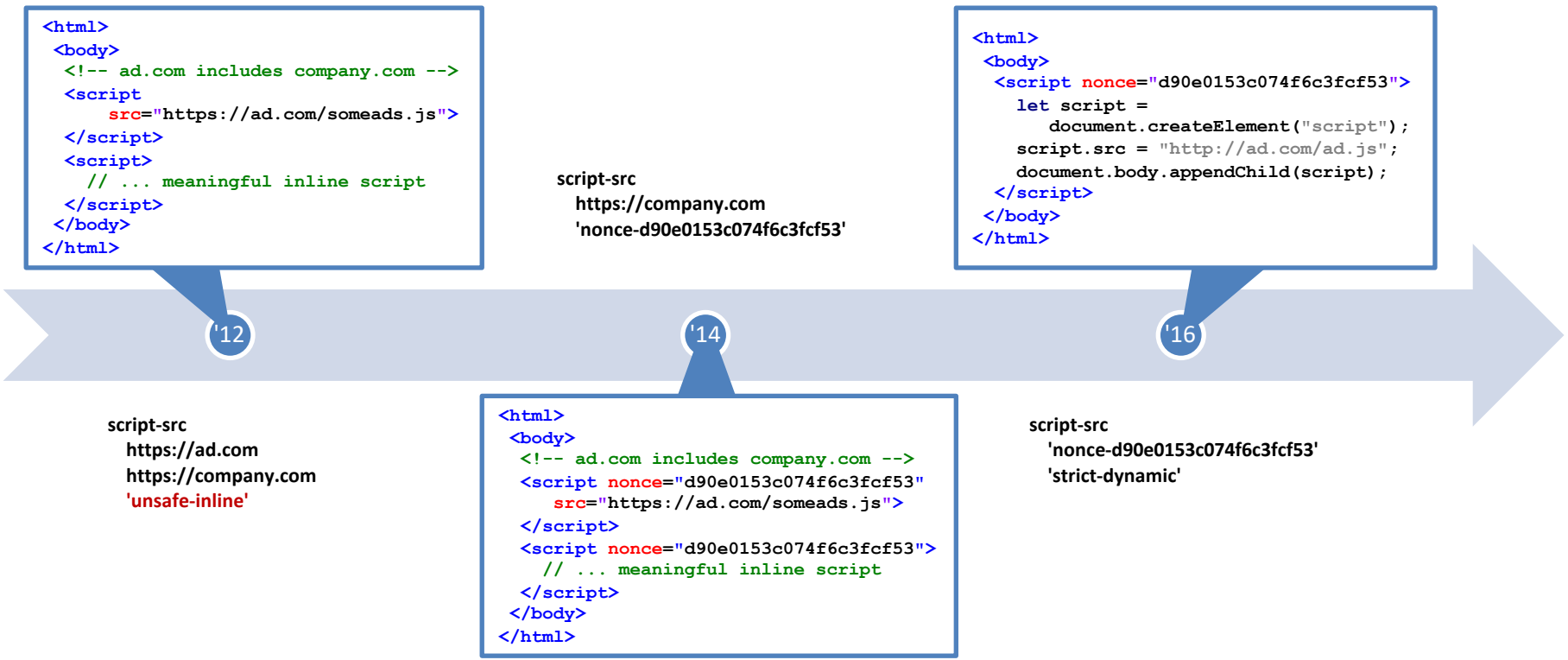
Cross-Site Scripting (XSS)



Content Security Policy (CSP)

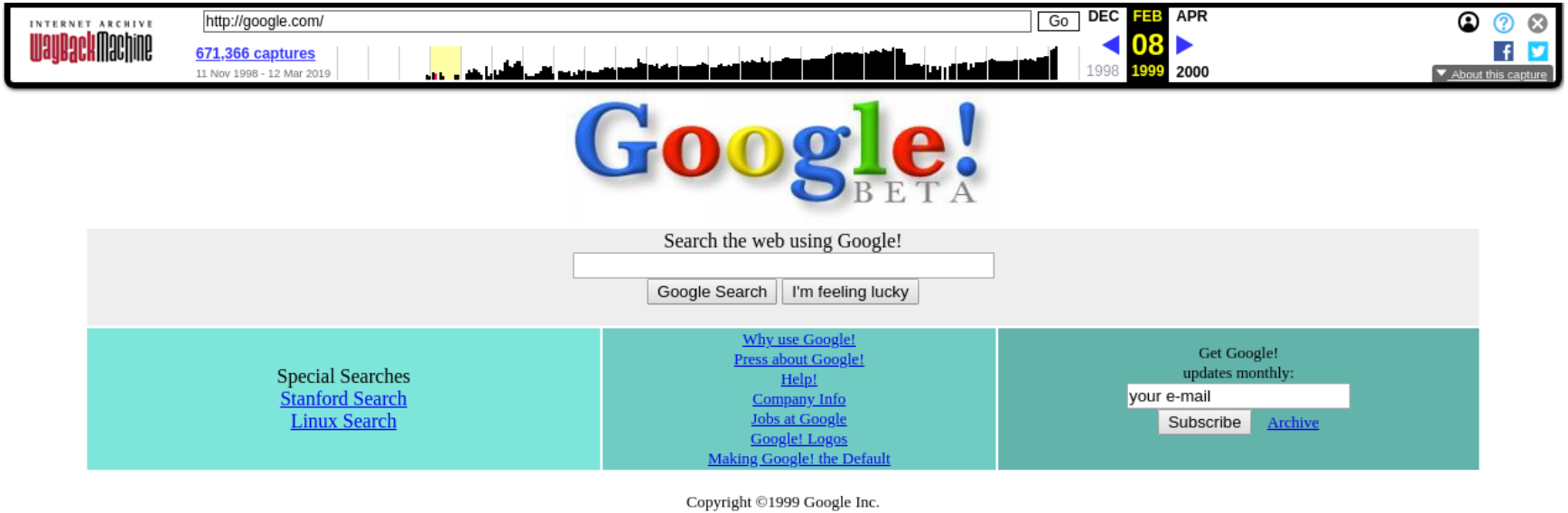


Content Security Policy (CSP)



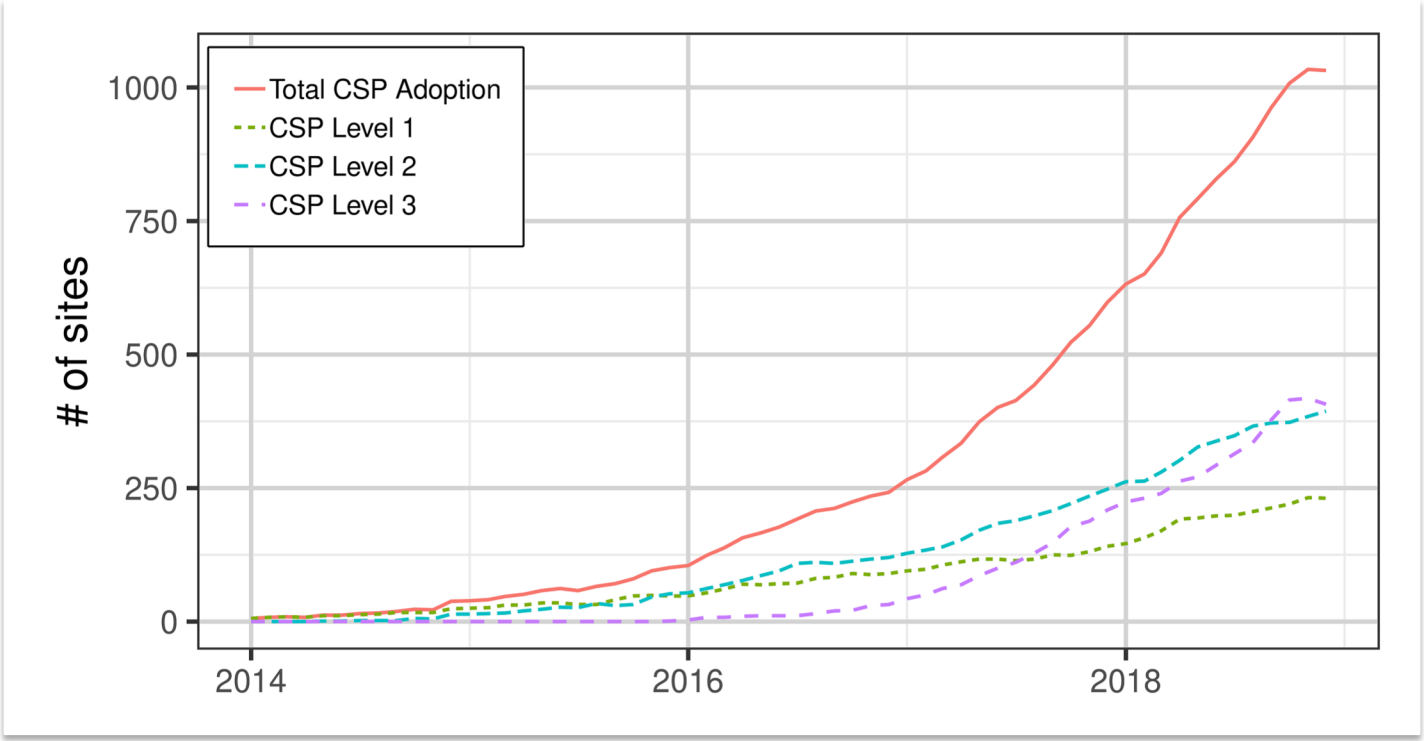
Content Security Policy
The Past

How to go back in time?

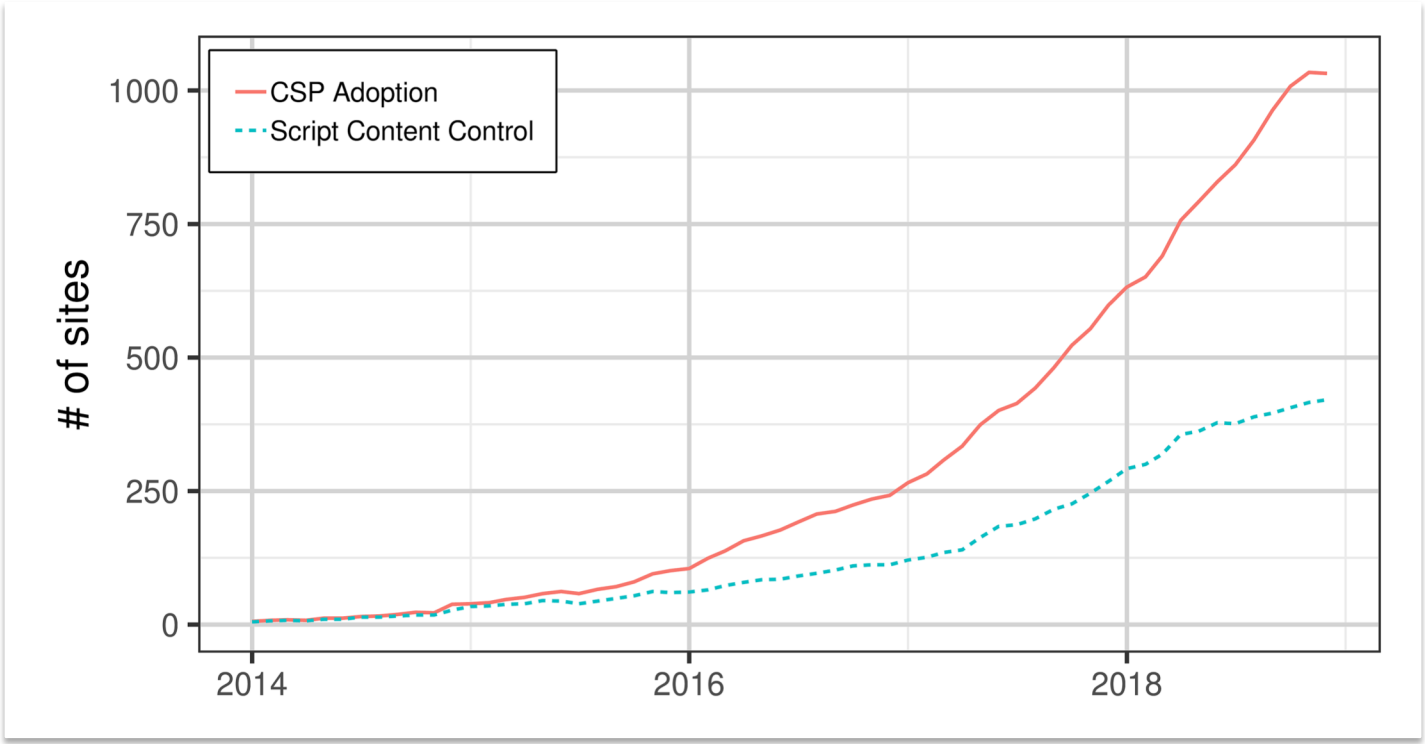


Also stores original HTTP headers prefixed with `X-Archive-Orig-`

CSP Adoption – Level



CSP Adoption – Use Cases



Script Content Control

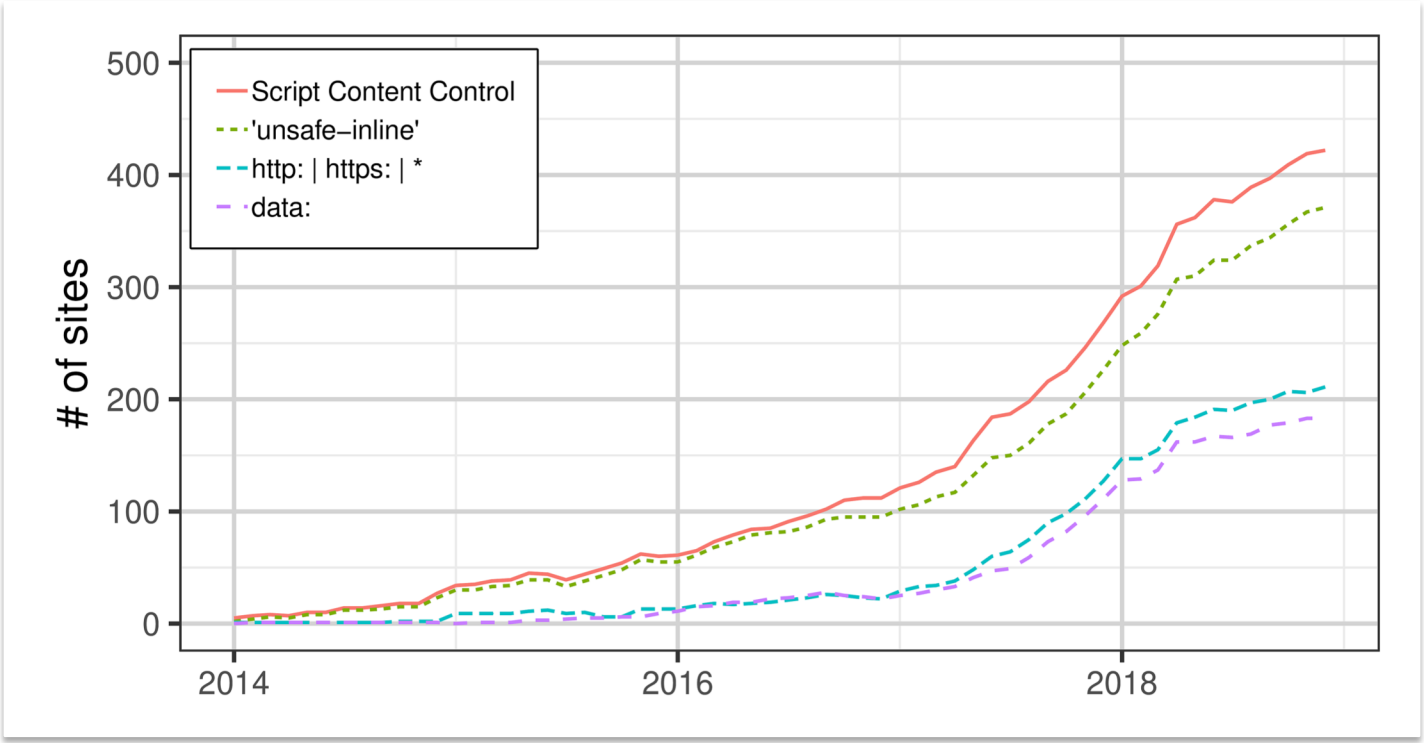
- Insecure Practices / Source Expressions:
  - `'unsafe-inline'`

    Allows the execution of any inline JS code
  - `http: | https: | http://* | https://* | *`

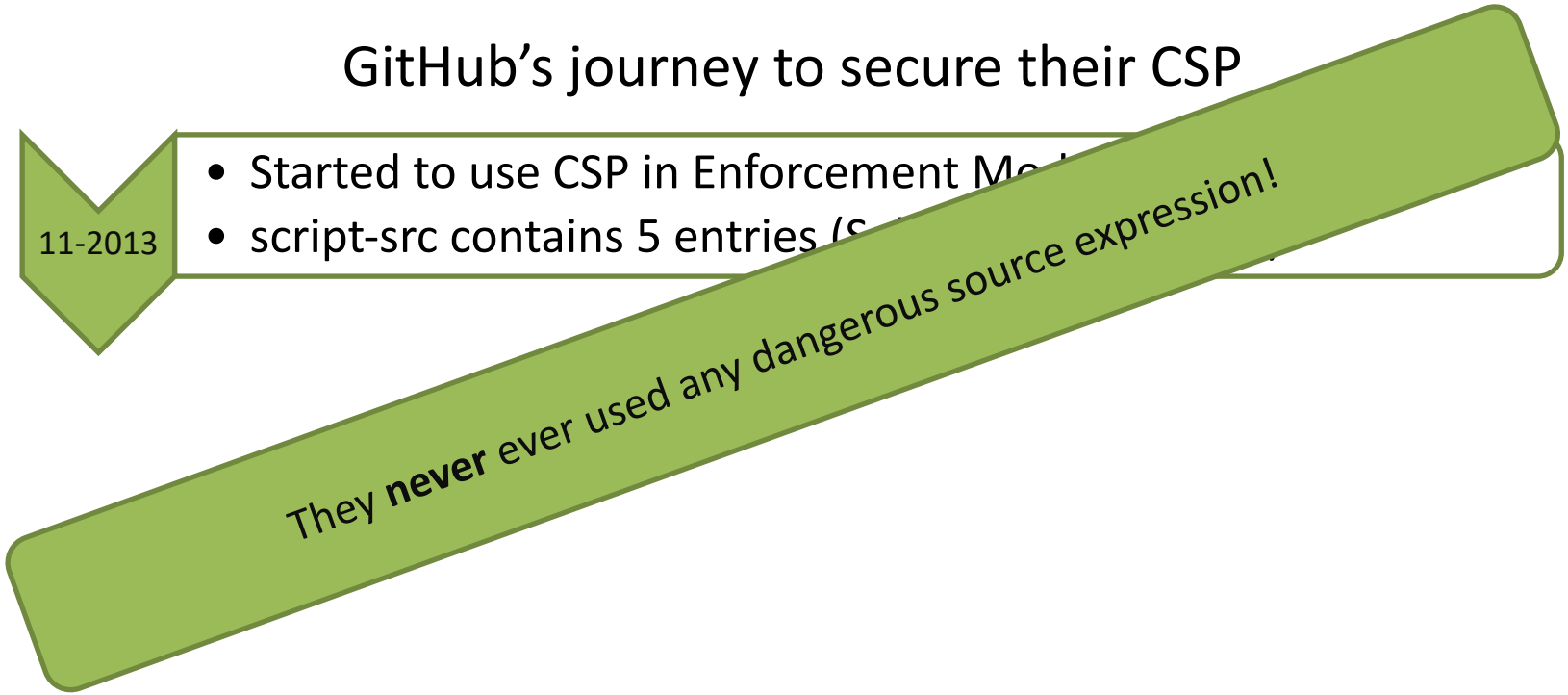
    Allows scripts to be loaded from "any" source
  - `data:`

    Allows data URIs to be used as script source

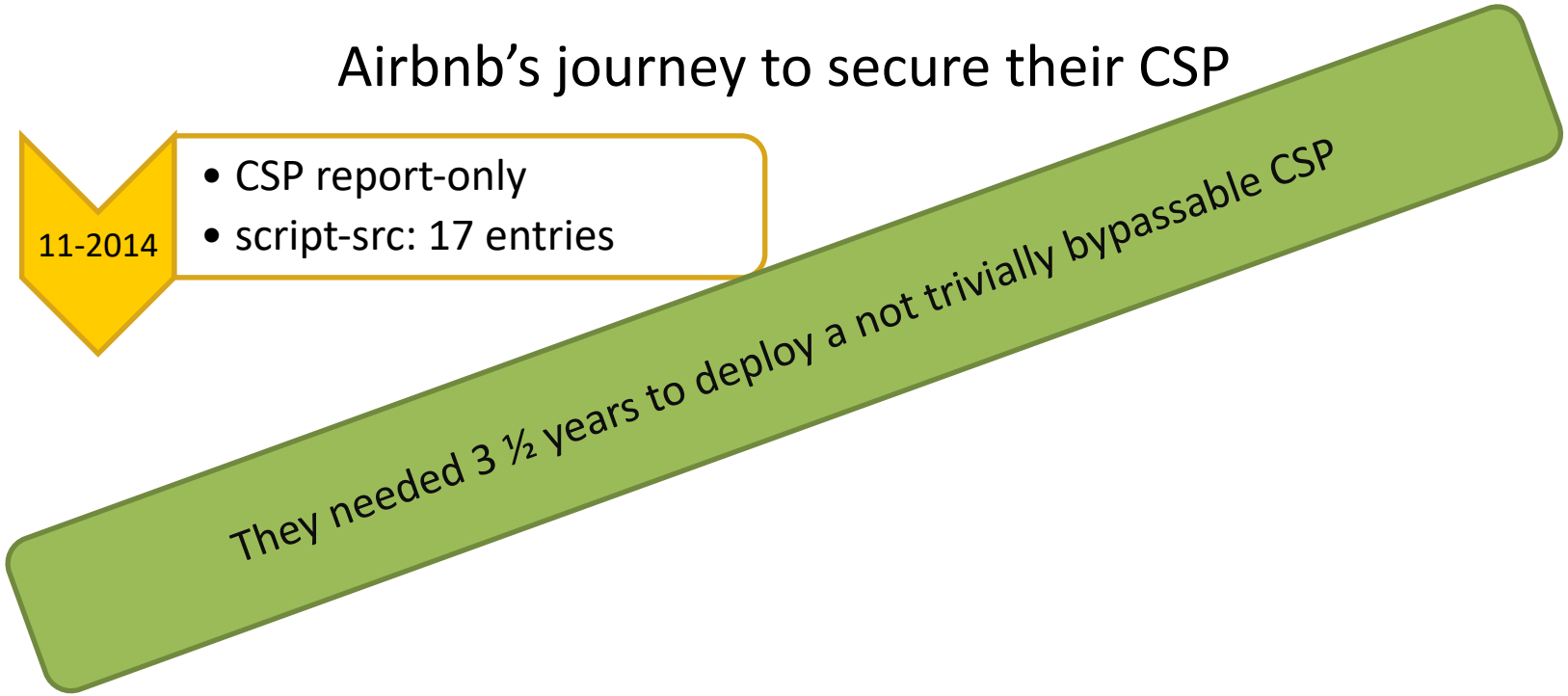
Script Content Control



Script Content Control – Example



Script Content Control – Example



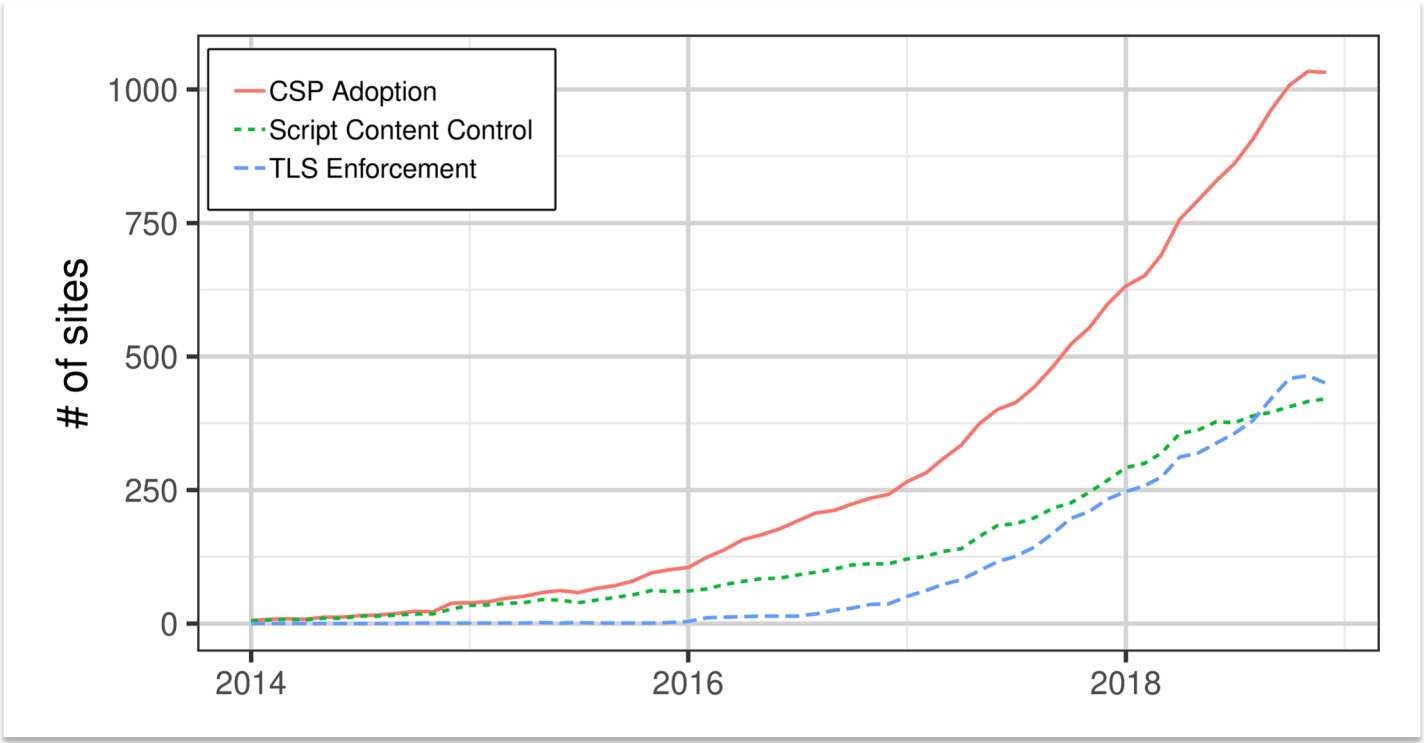
Take Aways – XSS Mitigation

- **Most CSPs do not effectively protect against XSS**
  - Insecure CSP practices are used by the majority of all Websites in the wild

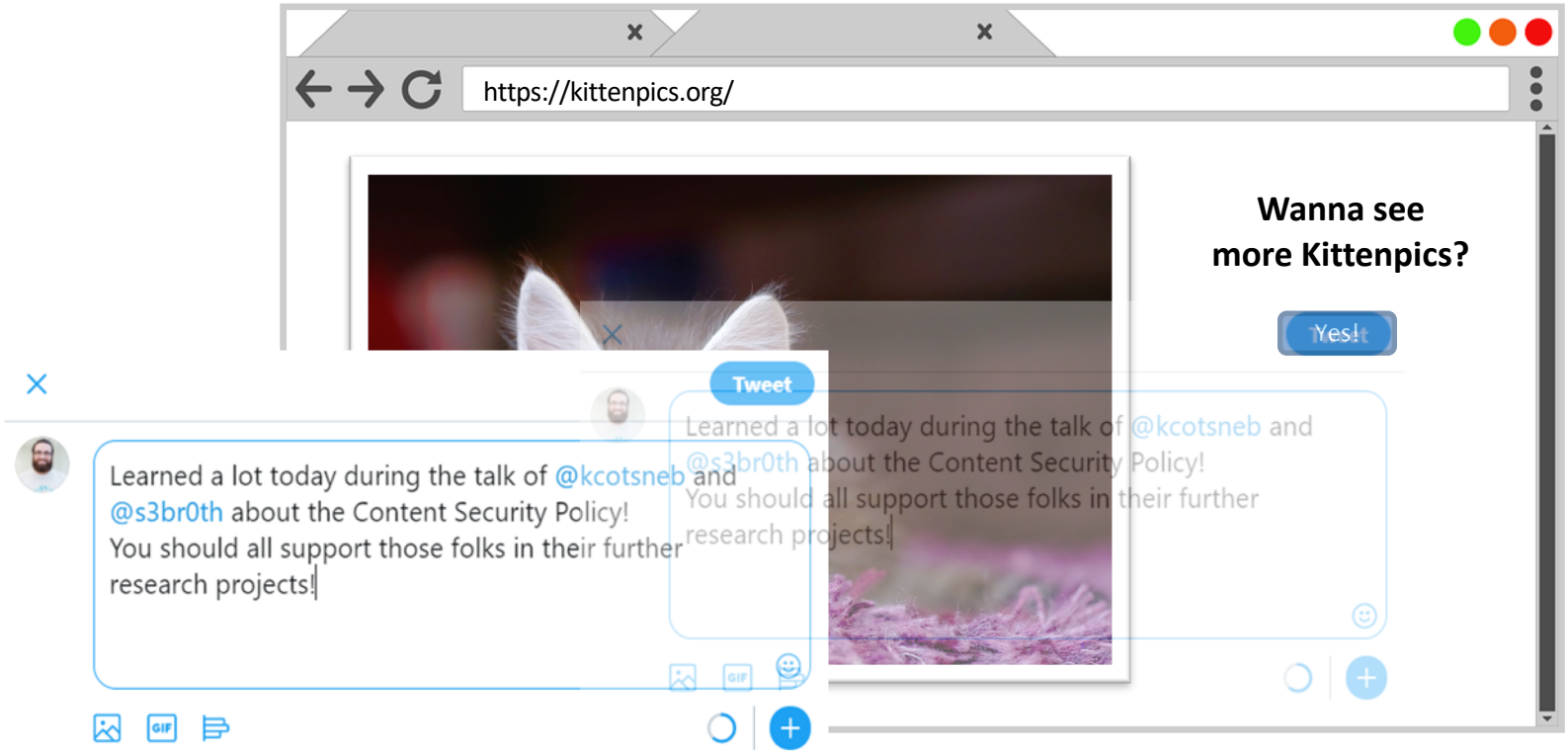
- Building a secure policy is very hard and requires a massive amount of time and engineering effort

Enforce Secure Network Connections

TLS Enforcement



Who doesn‘t like cats?

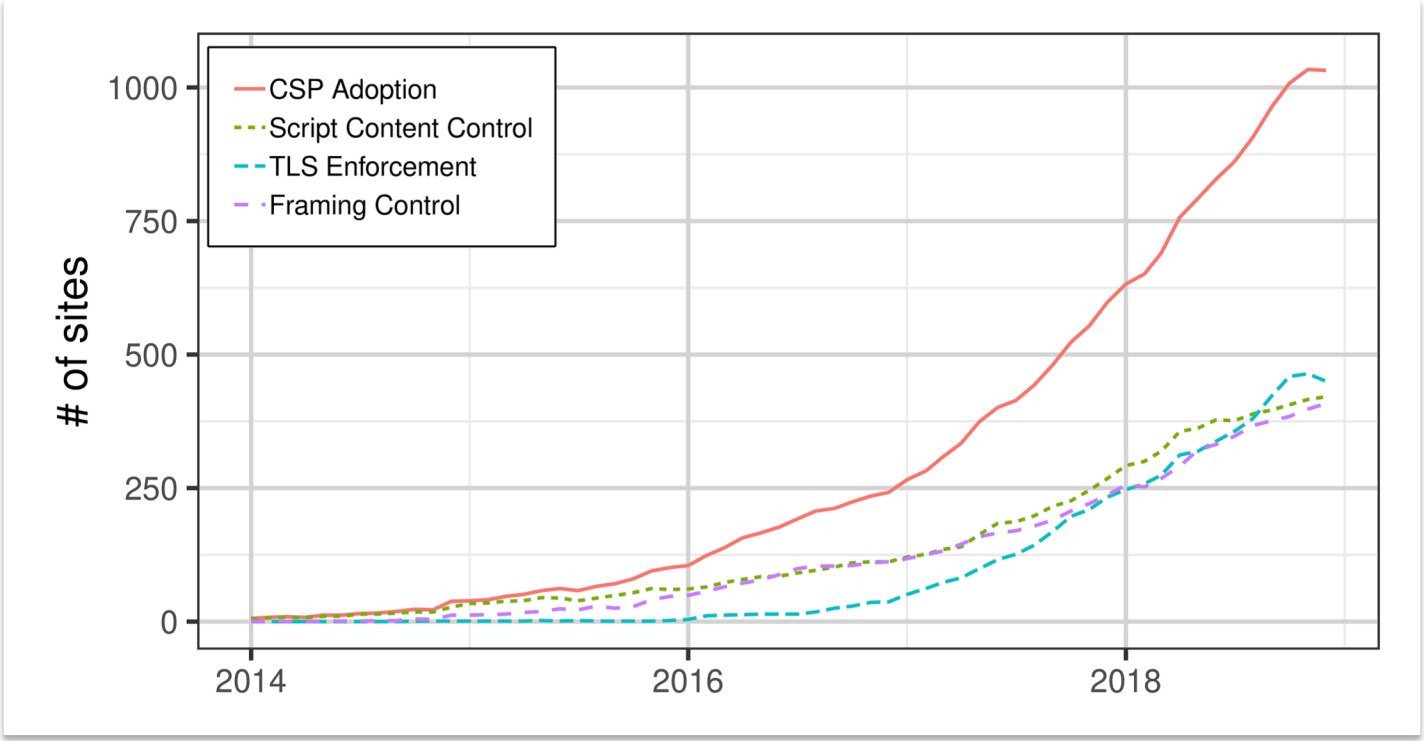


Framing Control – X-Frame-Options

- X-Headers are not standardized!
  - ➢ Inconsistent implementation

- Leads to security problems:
  - Partial support
  - Double Framing
- .. as well as functionality problems
  - X-Frame-Options can only have a single whitelist entry

Framing Control – CSP frame-ancestors



Framing Control – CSP frame-ancestors

- How does CSP frame-ancestors fix these problems:
  - **Inconsistent implementation:**

    CSP frame-ancestors is a well-defined standard in CSP level 2
  - **Partial support / Double Framing:**

    All “modern” browsers support it.

    Applies to all of a frame's ancestors not only the top frame.
  - **Explicit whitelist:**

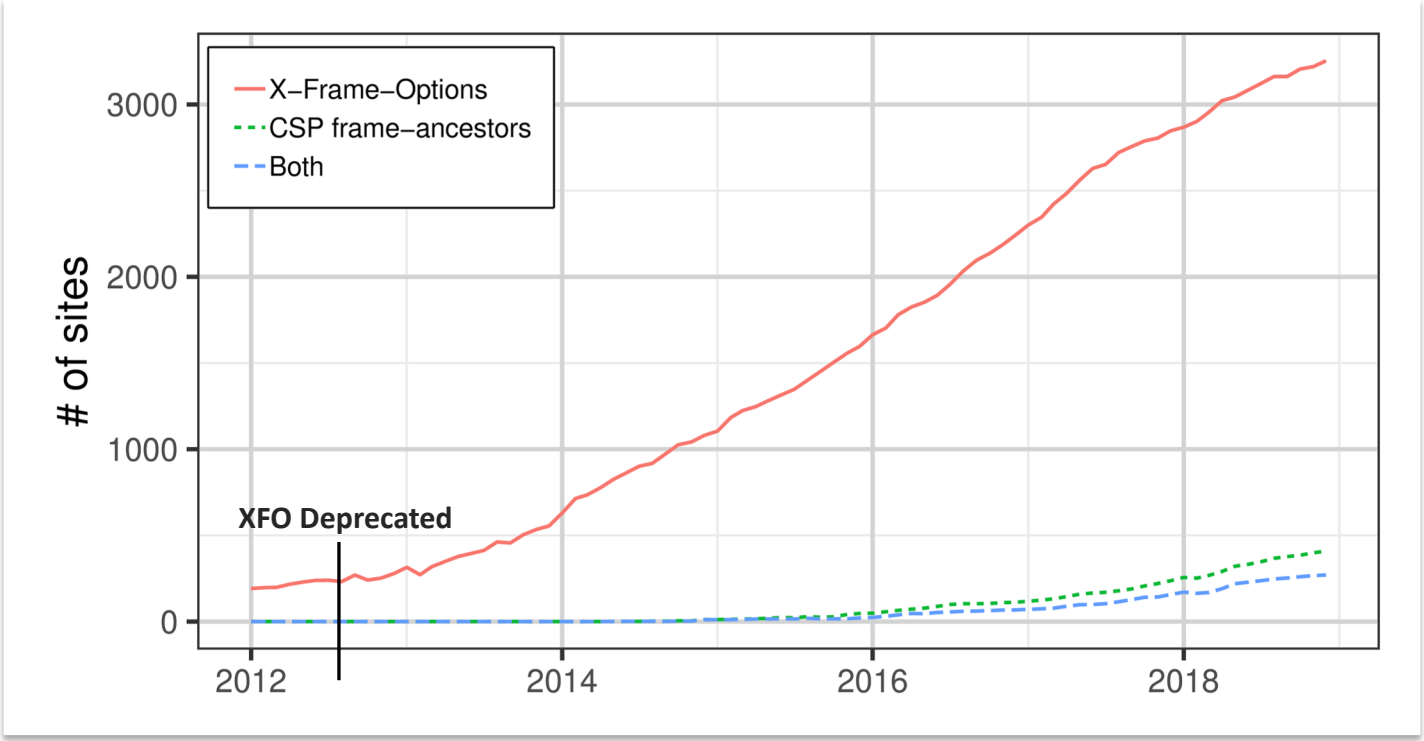
    ```
    frame-ancestors www.foo.com 'self' *.partner.com
    ```
  - **`frame-ancestors` can be used in isolation!**

    No need to restrict any of the page content.

Best practice for framing control

- CSP Level 2 browsers ignore X-Frame-Options in presence of frame-ancestors
- Securing sites for all browsers through combination of both
  - `X-Frame-Options DENY`
  - `frame-ancestors 'self'`

Framing Control – XFO vs. CSP



Content Security Policy
The Present

Framing Control – Notifications

Misconceptions about CSP

> CSP is a complex beast [...]. Some of our partner are iframing our site. We already had issue to implement the X-Frame header, that we did not want to deal with CSP.

Framing Control – Developer Study

CSP destroys Web applications

> [...] adding CSP [...] already placed on the roadmap in August of last year. We ran into some trouble with properly enabling the policies, as they ended up effectively killing the website.

Framing Control – Developer Study

CSP is too complex to deploy

> […] many first and third party integrations […] having a generic CSP policy that adds value and which is suitable for our entire estate is something that is very difficult to achieve.

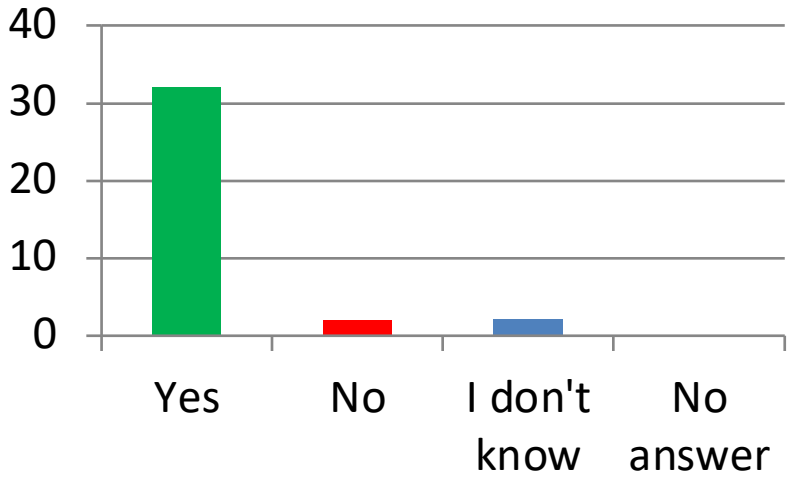
Framing Control – Developer Study

Building a CSP requires massive effort

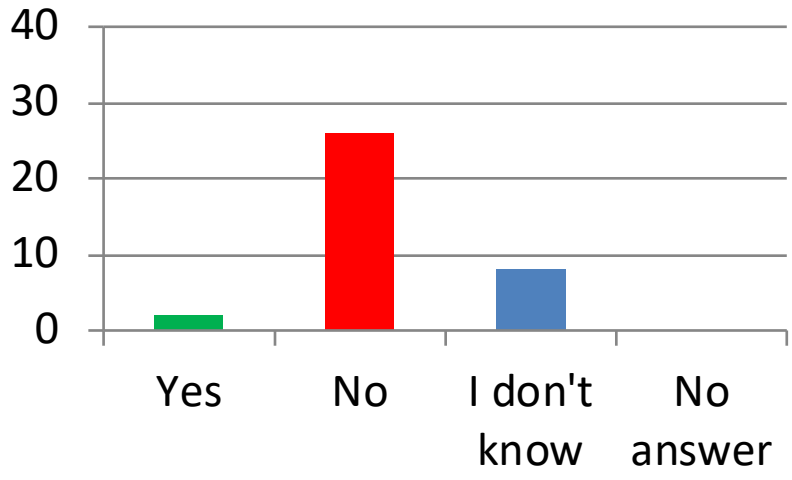
> We have a small team. Do we want to update our version of python or do we want to add CSP? Do we want to move to the new LTS version of Ubuntu or CSP? […] CSP always loose.

Framing Control – Developer Study

Do you believe CSP is a viable option to improve your site's resilience against XSS attacks?



Would your site work out of the box if you deployed a script-content restricting CSP today (disallow eval, inline scripts, and event handlers)?



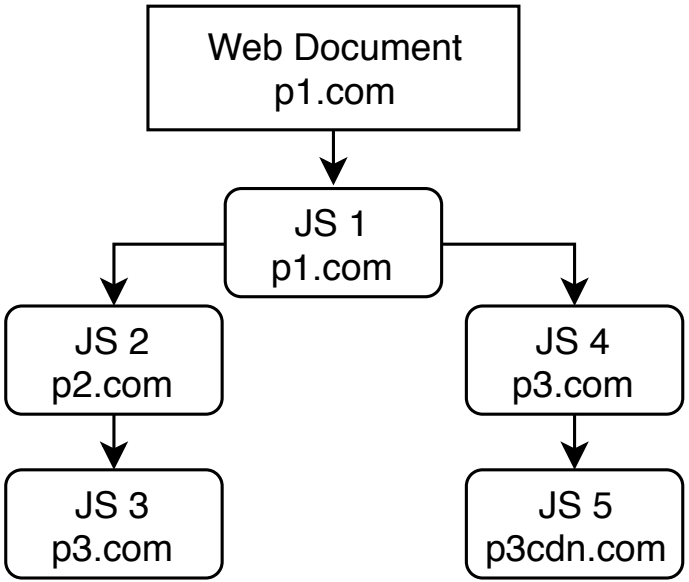
Why is CSP so insecure in practice?

- >95% of all policies in the wild are meaningless against XSS
- **Developers must be to blame!**

- Actually, let's try to understand this problem a little better
  - by looking at the inclusions relations that a modern Web application has
- Key question: whose behaviour interferes with CSP deployment?

Modelling inclusion relations on the Web

- JavaScript can include arbitrary other JavaScript
  - i.e., including one third party means the developer yields control over their inclusions
- Trust in third parties must be seen for a complete application

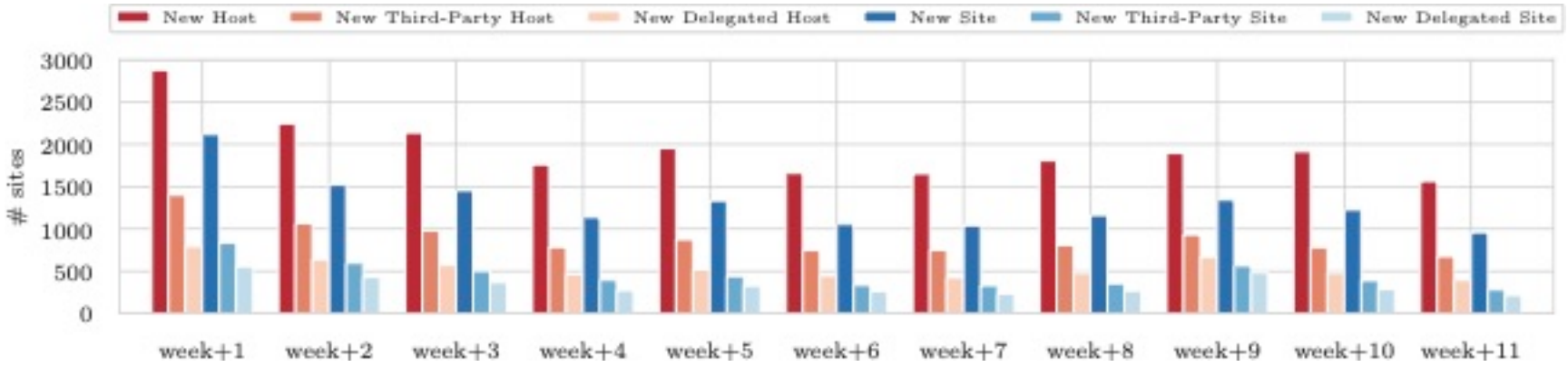


Which behaviour interferes with CSP?

1. Rotating hosts from which to include content
   - Can be mitigated by strict-dynamic, but that requires programmatic additions only
2. Using inline scripts or event handlers
   - Inline scripts can be nonced, event handlers cannot
3. Using eval
   - Well... don't!

CSP Problem #1: rotating hosts

- We ran a 12 week experiment on the Tranco top 10,000 sites
  - One crawl each week to collect inclusions/included parties



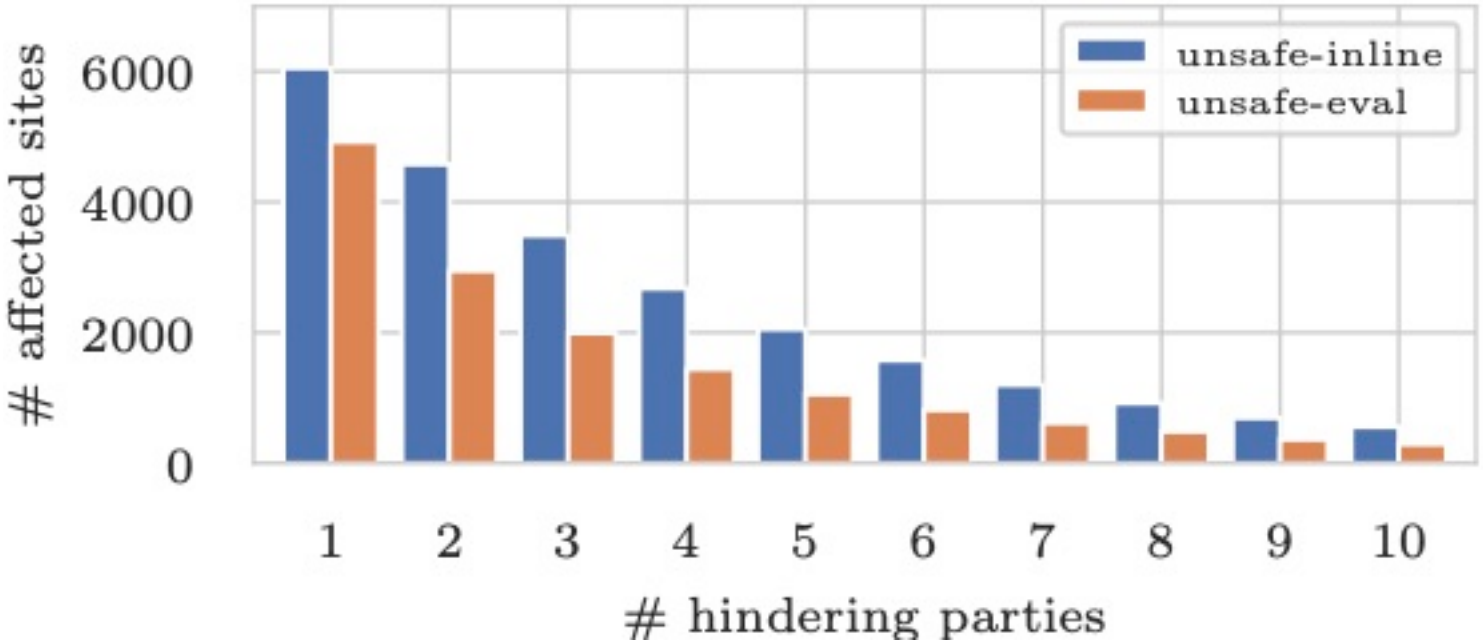
CSP Problem #2: unsafe-inline

	total	event handler	script
mandated by any	7,667	6,879	7,650
mandated by 1st party	7,643	4,972	7,618
mandated by 3rd party	**6,041**	**5,977**	**3,601**
- only 3rd party	24	**1,907**	32
- multiple 3rd parties	**4,573**	**4,446**	**1,663**
- delegated parties	1,317	1,263	343

CSP Problem #3: unsafe-eval

	total
mandated by any	6,334
mandated by 1st party	4,424
mandated by 3rd party	**4,911**
- only 3rd party	1,910
- multiple 3rd parties	**2,943**
- delegated parties	955

CSP Problems #2 & #3: multiple parties



`strict-dynamic` to the rescue?

- Third parties tend to mostly programmatically add new scripts
  - Only 1,141 / 8,041 (18%) of the sites in our data set have incompatible third party
- **BUT**: strict-dynamic only works with nonces or hashes
  - ➔ we cannot use unsafe-inline (no-op)
- Only 1,884 sites have third parties that fulfil both criteria
  - ➔ 6,157 (76%) cannot use strict-dynamic

Actual CSP usage in our data set

- 1,052 applications use CSP
- 1,006 have unsafe-inline
  - 712 of those have third parties with inline scripts
- 860 have unsafe-eval
  - 545 of those have third parties which use eval
- ➔ > 95% of polices are insecure, many of them need to be because of third parties

Content Security Policy
The Future

Why does CSP fail?

- History has shown: CSP for content restriction is very complex to deploy
  - Even major companies without third parties take years
- Third parties make life much harder to first-party developers
  - Rotating inclusions, incompatible inline scripts, etc.
- Both means that CSP has a bad reputation
  - "too complex to deploy"

Why does CSP fail?

- The "good parts" of CSP are often neglected
  - TLS enforcement very handy to migrate
  - Framing Control more expressive and secure than XFO

How can we address CSP's failure?

- Understand why developers shy away from CSP
  - Determine their mental models
  - Re-build CSP mechanism around understanding of developers, not handful of tier-1 companies
- Call to action for third parties
  - Become enabler of CSP, not deterrent

How can we address CSP's failure?

- Better support for developers
  - "I noted you started typing an event handler, use programatic addition instead"
  - "if you include this script, you are incompatible with CSP"
- (Evaluate incentives)
  - Only allow access to new APIs if a "sane" CSP is deployed
  - Third parties may block sites, though

Want to improve CSP?

- We are running a study on CSP's usability roadblocks
  - 45 minute interview
  - 45 minute coding task
  - 50€ Amazon gift card

- https://survey.swag.cispa.saarland

Summary

- CSP fails in practice for content restriction
  - Perceived complexity is high
  - Third parties are a major roadblock
- CSP can succeed if
  - Developers are better supported (from the get-go)
  - Third parties "play by the rules"