Beyond Blind Defense: Gaining Insights from Proactive AppSec

Bryant Zadegan
Director, Application Security
The Advisory Board Company

keybase.io/bryant
@eganist
Beyond Blind Defense

• In a nutshell
  – *Content Security Policy* (+ CSP2, + some CSP3)
  – *HTTP Public Key Pinning*

• Reporting!
  – *Security, QA, & Infrastructure*
    benefits and considerations
  – *How?* (The easy way)
"Enforcing markup and scripting assumptions client-side"

i.e. "you should never see this kind of code from us"

Content Security Policy in a nutshell
Content Security Policy

Quickstart

Content-Security-Policy-Report-Only:
default-src 'none';
object-src 'none';
script-src 'self';
connect-src 'self';
img-src 'self';
style-src 'self';
report-uri https://[id].report-uri.io/r/default/csp/[/mode]
Content Security Policy

• Existing site? Start with Reporting. Refine further.
• New application? Build it in from day one.
• Does not replace safe input/output
• w3.org/TR/CSP1/
• caniuse.com/contentsecuritypolicy
Content Security Policy

Threat Model (intended)\(^4\)

- `<xss />`
- Cross-Site Scripting
- Clickjacking
- Mixed Content
Content Security Policy

Threat Model *(stretched)*

Mismanaged Change

Internal Threat
Content Security Policy

Directives

- default-src \(\text{(applies to)}\)
- connect-src
- font-src
- img-src
- media-src
- object-src
- script-src
- style-src

\(\text{(does not apply to)}\)

- frame-src
- report-uri
- sandbox \(-\text{specifies an HTML sandbox policy that the user agent applies to the protected resource.}\)
Updates CSP with new directives. E.g.:

- base-uri, child-src, form-action, plugin-types
  - frame-ancestors supplants the x-frame-options header.
  - form-action and plugin-types restrict forms and plugins.

- For unsafe directives, Nonces and Hashes can now validate inline resources.
Updated Reporting:

- `effectiveDirective, statusCode, sourceFile, lineNumber, columnNumber`
- Also exposed through a `SecurityPolicyViolationEvent`
- Aids XSS triage specifically.

- [caniuse.com/contentsecuritypolicy2](https://caniuse.com/contentsecuritypolicy2)
You're probably doing it wrong

Allowing unsafe-inline
unbounded

Missing object-src
but permitting default-src

Allowing unsafe-eval
Content Security Policy 2

content-security-policy: default-src 'none';
script-src 'sha256-BOHH2w65dTag9u/qv3W+TOprNupZC7kCtCjUgCviuKU='

[...]

<!-- Hash-Source-->
<script>
    alert(123);
</script>
Content Security Policy 2

ccontent-security-policy: default-src 'none';
script-src 'nonce-2726c7f26c'

[...]

<!-- Nonce-Source-->
<script nonce="2726c7f26c">
    alert(123);
</script>
Nonce- and Hash-source will not protect you:

• If you drop untrusted data into a JS context.
• If you’re being stupid with `eval`.
• If you’re literally hashing or noncing every resource on a page as a post-processing step.

But they're still better than whitelists.
Considerations for refactoring:

- Hash-source **needs a hash for every script.**
- Nonces do not carry over to new scripts.
  - Fix by Google: "strict-dynamic"\(^1\)
- Whitelists are very hard to do correctly.\(^3\)
  - Hashes and Nonces statistically more effective\(^4\)
Content Security Policy 3

• Working Draft!
• Changes to brace for:
  – CSP 3 rewritten with FETCH in mind
    (fetch.spec.whatwg.org/)
  – Reporting slated for overhaul. "report-uri" deprecated in favor of "report-to"
    (w3c.github.io/reporting/)
Changes to enjoy:

- "strict-dynamic" (allows new scripts to inherit authorization from a nonced script)
- Sub-Resource Integrity matching work-in-progress (github.com/w3c/webappsec-csp/issues/78)

w3.org/TR/CSP3/
"Trust on first use for https connections"

i.e. "if you don't see this key, we shouldn't speak."

Http Public Key Pinning in a nutshell
Http Public Key Pinning

This can break brick your site. Use Reporting!

• Have multiple keys!
• Have multiple backups!
• Use Certificate Authority Authorization.  
Http Public Key Pinning

Quickstart

Public-Key-Pins-Report-Only:
max-age=5184000; includeSubdomains;
pin-sha256="BAD+HASH/00000000000000000000000000000000000000000=";
pin-sha256="BAD+HASH/0000000000000000000000000000000001=";
report-uri="https://[id].report-uri.io/r/default/hpkp/[mode]"

• caniuse.com/hpkp
Reporting

Why?
Reporting (CSP)

Security
• Your final layer of defense!
  – Not your only defense!
• What gets through your main defenses?
  – ...but is stopped in browser?

Considerations
• Absence of reports is not a report of absence (of issues)
  • Validate the reports.
    – Literally do input validation. Reports are untrusted.
Reporting (CSP)

Quality Assurance
• Confirm expectations live.
• What gets through?
  – ...but goes against policy?
• Reports speak to application quality!

Considerations
• Run CSP in QA.
  – i.e. not just in production.
• New to CSP?
  Expect heavy reports.
  – Reports approach zero as codebase aligns with policy.
Security

• Are your connections to users trusted?
• Why not?
  – Compromised clients?
  – ...networks?
  – ......servers?
• How do you know?

Infrastructure

• Certificate management
  – Enforce expectations.
  – Gain insight into certificate management practices.
Reporting (HPKP)

Considerations

• Chrome 46+ only; no reporting in Firefox 😐

• **Use a different domain!**
  – If you brick your site, don't brick your reporting.
Reporting

The easy way

• report-uri.io
  (It's free! Thanks, Scott!)

A bit harder

• Build your own aggregator
  mathiasbynens.be/notes/csp-reports

• Validate the reports.
  – Literally do input validation.
    Reports are untrusted.
Demo (CSP)

heisenberg.co/cspdemo/
Content Security Policy:
Content Security Policy:
### Filter your CSP reports

<table>
<thead>
<tr>
<th>Action</th>
<th>Date</th>
<th>URI</th>
<th>Directive</th>
<th>Blocked URI</th>
<th>Raw</th>
</tr>
</thead>
</table>

View 100 records
Easter egg (CSP hashing)

heisenberg.co/cspdemo/
Content Security Policy:

script-src 'sha256-+Jekolag7Mp6zATnqDFRB0Srw+85EoMJYnEFsg70PdE='; report-uri https://blackhatdemo.report-uri.io/r/default/csp/enforce

Refused to execute inline script because it violates the following Content Security Policy directive: "script-src 'sha256-+Jekolag7Mp6zATnqDFRB0Srw+85EoMJYnEFsg70PdE='". Either the 'unsafe-inline' keyword, a hash ('sha256-8UJH3w5d1agzuqv3WJ1opnUpZC7KLL1jgCViUKU='), or a nonce ('nonce-...') is required to enable inline execution.
Demo *(HPKP)*

[redskins.io](http://redskins.io)
Reporting Caveats

"It's about trust."
In the end, who *sends* the reports?
Hat Tip

Demos by Ryan Lester and Bryant Zadegan. Free use of report-uri.io as well as feedback by Scott Helme. Thanks to Rami Essaid and Distil Networks for sponsoring the talk.
## Questions? (Have Some Links)

<table>
<thead>
<tr>
<th>CSP</th>
<th>w3.org/TR/CSP1/ (old)</th>
<th>w3.org/TR/CSP2/ (current)</th>
<th>w3.org/TR/CSP3/ (draft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPKP</td>
<td>RFC 7469</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report-uri.io</td>
<td>report-uri.io</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[1] "Content Security Policy 3" [https://www.w3.org/TR/CSP3/#intro](https://www.w3.org/TR/CSP3/#intro)


Thank You!

Bryant Zadegan
Director, Application Security
The Advisory Board Company

keybase.io/bryant
@eganist