Content Security Policy (CSP)

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft, to site defacement, to malware distribution.

CSP is designed to be fully backward compatible (except CSP version 2 where there are some explicitly-mentioned inconsistencies in backward compatibility; more details here section 1.1). Browsers that don't support it still work with servers that implement it, and vice-versa: browsers that don't support CSP ignore it, functioning as usual, defaulting to the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy.

To enable CSP, you need to configure your web server to return the `Content-Security-Policy` HTTP header. (Sometimes you may see mentions of the `X-Content-Security-Policy` header, but that's an older version and you don't need to specify it anymore.)

Alternatively, the `<meta>` element can be used to configure a policy, for example:

```
<meta http-equiv="Content-Security-Policy"
      content="default-src 'self'; img-src https://*; child-src 'none';">
```

Threats

Mitigating cross-site scripting

A primary goal of CSP is to mitigate and report XSS attacks. XSS attacks exploit the browser's trust in the content received from the server. Malicious scripts are executed by the victim's browser because the browser trusts the source of the content, even when it's not coming from where it seems to be coming from.

CSP makes it possible for server administrators to reduce or eliminate the vectors by which XSS can occur by specifying the domains that the browser should consider to be valid sources of executable scripts. A CSP compatible browser will then only execute scripts loaded in source files received from those allowed domains, ignoring all other scripts (including inline scripts and event-handling HTML attributes).

As an ultimate form of protection, sites that want to never allow scripts to be executed can opt to globally disallow script execution.

Mitigating packet sniffing attacks

In addition to restricting the domains from which content can be loaded, the server can specify which protocols are allowed to be used; for example (and ideally, from a security standpoint), a server can specify that all content must be loaded using HTTPS. A complete data transmission security strategy includes not only enforcing HTTPS for data transfer, but also marking all cookies with the `secure` attribute and providing automatic redirects from HTTP pages to their HTTPS counterparts. Sites may also use the `Strict-Transport-Security` HTTP header to ensure that browsers connect to them only over an encrypted channel.

Using CSP

Configuring Content Security Policy involves adding the `Content-Security-Policy` HTTP header to a web page and giving it values to control what resources the user agent is allowed to load for that page. For example, a page that uploads and displays images could allow

images from anywhere, but restrict a form action to a specific endpoint. A properly designed Content Security Policy helps protect a page against a cross-site scripting attack. This article explains how to construct such headers properly, and provides examples.

Specifying your policy

You can use the `Content-Security-Policy` HTTP header to specify your policy, like this:

```
Content-Security-Policy: policy
```

The policy is a string containing the policy directives describing your Content Security Policy.

Writing a policy

A policy is described using a series of policy directives, each of which describes the policy for a certain resource type or policy area. Your policy should include a `default-src` policy directive, which is a fallback for other resource types when they don't have policies of their own (for a complete list, see the description of the `default-src` directive). A policy needs to include a `default-src` or `script-src` directive to prevent inline scripts from running, as well as blocking the use of `eval()`. A policy needs to include a `default-src` or `style-src` directive to restrict inline styles from being applied from a `<style>` element or a `style` attribute. There are specific directives for a wide variety of types of items, so that each type can have its own policy, including fonts, frames, images, audio and video media, scripts, and workers.

For a complete list of policy directives, see the reference page for the Content-Security-Policy header.

Examples: Common use cases

This section provides examples of some common security policy scenarios.

Example 1

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

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

Example 2

A web site 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
```

Example 3

A web site 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
```

Here, by default, content is only permitted from the document's origin, with the following exceptions:

- Images may load from anywhere (note the "*" wildcard).
- Media is only allowed from example.org and example.net (and not from subdomains of those sites).

- Executable script is only allowed from userscripts.example.com.

Example 4

A web site administrator for an online banking site wants to ensure that all its content is loaded using TLS, in order to prevent attackers from eavesdropping on requests.

```
Content-Security-Policy: default-src https://onlinebanking.example.com
```

The server permits access only to documents being loaded specifically over HTTPS through the single origin onlinebanking.example.com.

Example 5

A web site administrator of a web mail site wants to allow HTML in email, as well as images loaded from anywhere, but not JavaScript or other potentially dangerous content.

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

Note that this example doesn't specify a `script-src`; with the example CSP, this site uses the setting specified by the `default-src` directive, which means that scripts can be loaded only from the originating server.

Testing your policy

To ease deployment, CSP can be deployed in report-only mode. The policy is not enforced, but any violations are reported to a provided URI. Additionally, a report-only header can be used to test a future revision to a policy without actually deploying it.

You can use the `Content-Security-Policy-Report-Only` HTTP header to specify your policy, like this:

```
Content-Security-Policy-Report-Only: policy
```

If both a `Content-Security-Policy-Report-Only` header and a `Content-Security-Policy` header are present in the same response, both policies are honored. The policy specified in `Content-Security-Policy` headers is enforced while the `Content-Security-Policy-Report-Only` policy generates reports but is not enforced.

Enabling reporting

By default, violation reports aren't sent. To enable violation reporting, you need to specify the `report-uri` policy directive, providing at least one URI to which to deliver the reports:

```
Content-Security-Policy: default-src 'self'; report-uri http://reportcollector.example.com/collector.cgi
```

Then you need to set up your server to receive the reports; it can store or process them in whatever manner you determine is appropriate.

Violation report syntax

The report JSON object contains the following data:

`blocked-uri`

The URI of the resource that was blocked from loading by the Content Security Policy. If the blocked URI is from a different origin than the `document-uri`, then the blocked URI is truncated to contain just the scheme, host, and port.

`disposition`

Either `"enforce"` or `"report"` depending on whether the `Content-Security-Policy-Report-Only` header or the `Content-Security-Policy` header is used.

`document-uri`

The URI of the document in which the violation occurred.

`effective-directive`

The directive whose enforcement caused the violation. Some browsers may provide different values, such as Chrome providing `style-src-elem` / `style-src-attr`, even when the actually enforced directive was `style-src`.

`original-policy`

The original policy as specified by the `Content-Security-Policy` HTTP header.

`referrer`

The referrer of the document in which the violation occurred.

`script-sample`

The first 40 characters of the inline script, event handler, or style that caused the violation. Only applicable to `script-src*` and `style-src*` violations, when they contain the `'report-sample'`

`status-code`

The HTTP status code of the resource on which the global object was instantiated.

`violated-directive`

The name of the policy section that was violated.

Sample violation report

Let's consider a page located at `http://example.com/signup.html`. It uses the following policy, disallowing everything but stylesheets from `cdn.example.com`.

```
Content-Security-Policy: default-src 'none'; style-src cdn.example.com; report-uri /_/csp-reports
```

The HTML of `signup.html` looks like this:

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="UTF-8">
    <title>Sign Up</title>
    <link rel="stylesheet" href="css/style.css">
  </head>
  <body>
    Here be content.
  </body>
</html>
```

Can you spot the mistake? Stylesheets are allowed to be loaded only from `cdn.example.com`, yet the website tries to load one from its own origin (`http://example.com`). A browser capable of enforcing CSP would send the following violation report as a POST request to `http://example.com/_/csp-reports`, when the document is visited:

```
{
  "csp-report": {
    "document-uri": "http://example.com/signup.html",
    "referrer": "",
    "blocked-uri": "http://example.com/css/style.css",
    "violated-directive": "style-src cdn.example.com",
    "original-policy": "default-src 'none'; style-src cdn.example.com; report-uri /_/csp-reports"
  }
}
```

As you can see, the report includes the full path to the violating resource in `blocked-uri`. This is not always the case. For example, if the `signup.html` attempted to load CSS from `http://anothercdn.example.com/stylesheet.css`, the browser would *not* include the full path, but only the origin (`http://anothercdn.example.com`). The CSP specification gives an explanation of this odd behavior. In summary, this is done to prevent leaking sensitive information about cross-origin resources.

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	
`Content-Security-Policy`	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android	Ye
`base-uri`	Chrome 40	Edge 79	Firefox 35	Opera 27	Safari 10	Chrome Android	Ye
`block-all-mixed-content`	Chrome Yes	Edge 79	Firefox 48	Opera Yes	Safari ?	Chrome Android	Ye
`child-src`	Chrome 40	Edge 15	Firefox 45	Opera 27	Safari 10	Chrome Android	Ye
`connect-src`	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android	Ye
`default-src`	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android	Ye
`font-src`	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android	Ye
`form-action`	Chrome 40	Edge 15	Firefox 36	Opera 27	Safari 10	Chrome Android	Ye
`frame-ancestors`	Chrome 40	Edge 15	Firefox 33	Opera 26	Safari 10	Chrome Android	Ye

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android
frame-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
img-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
manifest-src	Chrome Yes	Edge 79	Firefox 41	Opera Yes	Safari No	Chrome Android Ye
media-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
<meta> element support	Chrome Yes	Edge 18	Firefox 45	Opera Yes	Safari Yes	Chrome Android Ye
navigate-to	Chrome No	Edge No	Firefox No	Opera No	Safari No	Chrome Android N
object-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
plugin-types	Chrome 40–90	Edge 15–90	Firefox No	Opera 27–76	Safari 10	Chrome Android Ye
prefetch-src	Chrome No	Edge No	Firefox No	Opera No	Safari No	Chrome Android No
referrer	Chrome 33–56	Edge No	Firefox 37–62	Opera Yes	Safari No	Chrome 33–5 Android
report-sample	Chrome 59	Edge 79	Firefox ?	Opera 46	Safari 15.4	Chrome Android 5
report-to	Chrome 70	Edge 79	Firefox No	Opera No	Safari No	Chrome Android 7
report-uri	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
require-sri-for	Chrome 54	Edge 79	Firefox No	Opera 41	Safari No	Chrome Android 5
require-trusted-types-for	Chrome 83	Edge 83	Firefox No	Opera 69	Safari No	Chrome Android 8
sandbox	Chrome 25	Edge 14	Firefox 50	Opera 15	Safari 7	Chrome Android Ye
script-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
With external scripts	Chrome 59	Edge 79	Firefox ?	Opera ?	Safari ?	Chrome Android 5

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android
script-src-attr	Chrome 75	Edge 79	Firefox No	Opera 62	Safari TP	Chrome Android 7
script-src-elem	Chrome 75	Edge 79	Firefox No	Opera 62	Safari TP	Chrome Android 7
strict-dynamic	Chrome 52	Edge 79	Firefox 52	Opera 39	Safari 15.4	Chrome Android 5
style-src	Chrome 25	Edge 14	Firefox 23	Opera 15	Safari 7	Chrome Android Ye
style-src-attr	Chrome 75	Edge 79	Firefox No	Opera 62	Safari TP	Chrome Android 7
style-src-elem	Chrome 75	Edge 79	Firefox No	Opera 62	Safari TP	Chrome Android 7
trusted-types	Chrome 83	Edge 83	Firefox No	Opera 69	Safari No	Chrome Android 8
unsafe-hashes	Chrome 69	Edge 79	Firefox No	Opera 56	Safari 15.4	Chrome Android 6
upgrade-insecure-requests	Chrome 43	Edge 17	Firefox 42	Opera 30	Safari 10.1	Chrome Android 4
worker-src	Chrome 59	Edge 79	Firefox 58	Opera 48	Safari 15.5	Chrome Android 59
Worker support	Chrome Yes	Edge 79	Firefox 50	Opera ?	Safari 10	Chrome Android Ye

Tip: you can click/tap on a cell for more information.

Full support | In development. Supported in a pre-release version. | No support | Compatibility unknown

Experimental. Expect behavior to change in the future. | Non-standard. Check cross-browser support before using.

Deprecated. Not for use in new websites. | See implementation notes. | Uses a non-standard name. | Has more compatibility info.

Compatibility notes

A specific incompatibility exists in some versions of the Safari web browser, whereby if a Content Security Policy header is set, but not a Same Origin header, the browser will block self-hosted content and off-site content, and incorrectly report that this is due to the Content Security Policy not allowing the content.

See also

- `Content-Security-Policy` HTTP Header

- `Content-Security-Policy-Report-Only` HTTP Header
- Content Security in WebExtensions
- CSP in Web Workers
- Privacy, permissions, and information security
- CSP Evaluator - Evaluate your Content Security Policy
- CSP Scanner - Improve your Content Security Policy

Last modified: Aug 11, 2022, by MDN contributors