CSP: Content security policy

Introduction

While it's true that CSP is being used to prevent attacks such as XSS, that is not it's only purpose. It also protects against data injection attacks for example. We all know what the impact of these issues can be and they can be devestating. Luckily this protocol has been invented to mitigate the damage attackers can do. It's important to note however CSP can not protect all evils and we need to be careful because a misconfiguration is very easy to make.

What is CSP

CSP is something that your browser needs to support before it can work. This is very important to know and it makes CSP a client-side protection. The server has nothing to do with enforcing the protection but instead only indicates which protection should be available. This can be either in the form of headers (content-security-policy header) or in the meta tag:

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

Directives

In CSP, we can call upon several "directives". These indicate which resource we are trying to restrict.

The default-src directive indicates what the default CSP behaviour should be if nothing else is specified. It has prevalence over the other directives.

We also need to know there are different types of directives which we will go over. If you are ever unsure as to which directive to use, keep the descriptions of the directive categories in mind as they will make things easier.

Fetch directives

Fetch directives control the locations from which certain resource types may be loaded. Some noteable directives are:

- img-src
- default-src
- script-src
- media-src
- style-src

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

Document directives

Document directives govern the properties of a document or worker environment to which a policy applies. Some noticeable ones include:

- `base-uri`
  - Restricts the URLs which can be used in a document's `<base>` element.
- `sandbox`
  - Enables a sandbox for the requested resource similar to the `<iframe>` `sandbox` attribute.

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

Navigation directives

Navigation directives govern to which locations a user can navigate or submit a form, for example.

- `form-action`
  - Restricts the URLs which can be used as the target of a form submissions from a given context.
- `frame-ancestors`
  - Specifies valid parents that may embed a page using `<frame>`, `<iframe>`, `<object>`, `<embed>`, or `<applet>`. `navigate-to` Restricts the URLs to which a document can initiate navigation by any means, including `<form>` (if `form-action` is not specified), `<a>`, `window.location`, `window.open`, etc.

Reporting directives

Reporting directives control the reporting process of CSP violations. See also the `Content-Security-Policy-Report-Only` header.

- `report-uri`
  - Instructs the user agent to report attempts to violate the Content Security Policy. These violation reports consist of JSON documents sent via an HTTP `POST` request to the specified URI.
- `report-to`
  - Fires a `SecurityPolicyViolationEvent`.

Other directives

- `require-sri-for`
  - Requires the use of SRI for scripts or styles on the page. `require-trusted-types-for` Enforces Trusted Types at the DOM XSS injection sinks.
- `trusted-types`
  - Used to specify an allow-list of Trusted Types policies. Trusted Types allows applications to lock down DOM XSS injection sinks to only accept non-spoofable, typed values in place of strings.
- `upgrade-insecure-requests`
  - Instructs user agents to treat all of a site's insecure URLs (those served over HTTP) as though they have been replaced with secure URLs (those served over HTTPS). This directive is intended for web sites with large numbers of insecure legacy URLs that need to be rewritten.

Values

Keyword values

- `none`
  - Won't allow loading of any resources.
- `self`
  - Only allow resources from the current origin.
- `strict-dynamic` TBD
- `report-sample` TBD

Unsafe keyword values

Only use these if there is no other option and if it's secured properly! As the word says, it's unsafe.

`unsafe-inline` Allow use of inline resources.

`unsafe-eval` Allow use of dynamic code evaluation such as `eval`, `setImmediate`, and `window.execScript`.

Hosts values

Host

Only allow loading of resources from a specific host, with optional scheme, port, and path.
e.g. `example.com`, `*.example.com`, `https://*.example.com:12/path/to/file.js`

Scheme

Only allow loading of resources over a specific scheme, should always end with "`:`".
e.g. `https:`, `http:`, `data:` etc.

Examples

When we take this together we have a few examples before we into the test.

```
Content-Security-Policy: default-src self ; img-src: *;
```

In this one, we want to only allow scripts from the domain itself by default but images should be allowed from anywhere.

```
Content-Security-Policy: default-src * ; img-src: self;
```

In this one, we want to only allow scripts from everywhere by default but images should be allowed from the domain itself only. As you can imagine, this is insanely insecure due to the default-src being a wildcard.

Excercises CSP