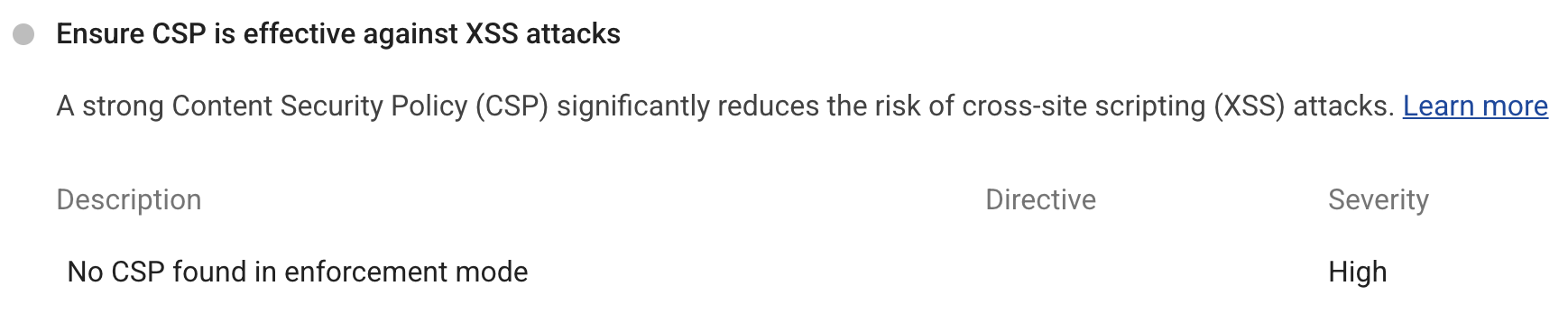
**Ensure CSP is effective against XSS attacks**

Jun 16, 2021

Appears in: [Best Practices audits](https://web.dev/lighthouse-best-practices)

A Content Security Policy (CSP) helps to ensure any content loaded in the page is trusted by the site owner. CSPs mitigate cross-site scripting (XSS) attacks because they can block unsafe scripts injected by attackers. However, the CSP can easily be bypassed if it is not strict enough. Check out [Mitigate cross-site scripting (XSS) with a strict Content Security Policy (CSP)](https://web.dev/strict-csp/) for more information. Lighthouse collects CSPs enforced on the main document, and reports issues from [CSP Evaluator](https://csp-evaluator.withgoogle.com/) if they can be bypassed.

Lighthouse report warning that no CSP is found in enforcement mode.

**Required practices for a non-bypassable CSP** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#required-practices-for-a-non-bypassable-csp)

Implement the following practices to ensure that your CSP can't be bypassed. If the CSP can be bypassed, Lighthouse will emit a high severity warning.

**CSP targets XSS** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#csp-targets-xss)

To target XSS, a CSP should include the script-src, object-src, and base-uri directives. The CSP should also be free of syntax errors.

script-src and object-src secures a page from unsafe scripts and unsafe plugins respectively. Alternatively, default-src can be used to configure a broad policy in place of [many directives](https://developer.mozilla.org/docs/Web/HTTP/Headers/Content-Security-Policy/default-src) including script-src and object-src.

base-uri prevents the injection of unauthorized <base> tags which can be used to redirect all relative URLs (like scripts) to an attacker-controlled domain.

**CSP uses nonces or hashes to avoid allowlist bypasses** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#csp-uses-nonces-or-hashes-to-avoid-allowlist-bypasses)

A CSP that configures an allowlist for script-src relies on the assumption that all responses coming from a trusted domain are safe, and can be executed as scripts. However, this assumption does not hold for modern applications; some common, benign patterns such as exposing [JSONP interfaces](https://lcamtuf.blogspot.ch/2011/08/subtle-deadly-problem-with-csp.html) and [hosting copies of the AngularJS library](https://github.com/cure53/XSSChallengeWiki/wiki/H5SC-Minichallenge-3:-%22Sh*t,-it's-CSP!%22) allow attackers to escape the confines of CSP.

In practice, while it may not be obvious to application authors, [the majority of script-src allowlists can be circumvented](https://research.google.com/pubs/pub45542.html) by an attacker with an XSS bug, and provide little protection against script injection. In contrast, the [nonce-based and hash-based approaches](https://web.dev/strict-csp/#what-is-a-strict-content-security-policy) do not suffer from these problems and make it easier to adopt and maintain a more secure policy.

For example, this code uses a JSONP endpoint hosted on a trusted domain to inject an attacker controlled script:

CSP:

script-src https://trusted.example.com

HTML:

<script src="https://trusted.example.com/path/jsonp?callback=alert(document.domain)//"></script>

To avoid being bypassed, a CSP should allow scripts individually using nonces or hashes and use 'strict-dynamic' instead of an allowlist.

**Additional recommendations for a secure CSP** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#additional-recommendations-for-a-secure-csp)

Implement the following practices for added security and compatibility. If the CSP does not follow one of the recommendations, Lighthouse will emit a medium severity warning.

**Configure CSP reporting** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#configure-csp-reporting)

[Configuring a reporting destination](https://developers.google.com/web/updates/2018/09/reportingapi) will help monitor for any breakages. You can set the reporting destination by using the report-uri or report-to directives. report-to is not currently supported by all modern browsers so it is recommended to use both or just report-uri.

If any content violates the CSP, the browser will send a report to the configured destination. Make sure you have an application configured at this destination handling these reports.

**Define the CSP in an HTTP header** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#define-the-csp-in-an-http-header)

A CSP can be defined in a meta tag like this:

<meta http-equiv="Content-Security-Policy" content="script-src 'none'">

However, you should define a CSP in an HTTP response header if you can. An injection before the meta tag will bypass the CSP. Additionally, frame-ancestors, sandbox and reporting are not supported in meta tag CSPs.

**Ensure CSP is backwards compatible** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#ensure-csp-is-backwards-compatible)

Not all browsers support CSP nonces/hashes, therefore adding unsafe-inline as a fallback for non-compliant browsers is recommended. If the browser does support nonces/hashes, unsafe-inline will be ignored.

Similarly, strict-dynamic is not supported by all browsers. It is recommended to set an allowlist as a fallback for any non-compliant browsers. The allowlist will be ignored in browsers that support strict-dynamic.

**How to develop a strict CSP** [**#**](https://web.dev/csp-xss/?utm_source=lighthouse&utm_medium=devtools#how-to-develop-a-strict-csp)

Below is an example of using a strict CSP with a nonce-based policy.

CSP:

script-src 'nonce-random123' 'strict-dynamic' 'unsafe-inline' https:;  
object-src 'none';  
base-uri 'none';  
report-uri https://reporting.example.com;

HTML:

<script nonce="random123" src="https://trusted.example.com/trusted\_script.js"></script>

random123 would be any base64 string generated server-side every time the page loads. unsafe-inline and https: are ignored in modern browsers because of the nonce and strict-dynamic. For more information about adopting a strict CSP, check out the [Strict CSP guide](https://web.dev/strict-csp/#adopting-a-strict-csp).

You can check a CSP for potential bypasses using Lighthouse and [CSP Evaluator](https://csp-evaluator.withgoogle.com/). If you want to test a new CSP without the risk of breaking existing pages, define the CSP in report-only mode by using Content-Security-Policy-Report-Only as the header name. This will send CSP violations to any reporting destinations you have configured with report-to and report-uri, but it will not actually enforce the CSP.

**Content Security Policy** ([CSP](https://developer.mozilla.org/en-US/docs/Glossary/CSP)) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting ([XSS](https://developer.mozilla.org/en-US/docs/Glossary/Cross-site_scripting)) 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](https://www.w3.org/TR/CSP2) 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](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy).

To enable CSP, you need to configure your web server to return the [Content-Security-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#threats)

### [Mitigating cross-site scripting](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies) and providing automatic redirects from HTTP pages to their HTTPS counterparts. Sites may also use the [Strict-Transport-Security](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Strict-Transport-Security) HTTP header to ensure that browsers connect to them only over an encrypted channel.

## [Using CSP](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#using_csp)

Configuring Content Security Policy involves adding the [Content-Security-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#specifying_your_policy)

You can use the [Content-Security-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/default-src) directive). A policy needs to include a [default-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/default-src) or [script-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/default-src) or [style-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/style-src) directive to restrict inline styles from being applied from a [<style>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/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.

## [Examples: Common use cases](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#examples_common_use_cases)

This section provides examples of some common security policy scenarios.

### [Example 1](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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' trusted.com \*.trusted.com

### [Example 3](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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 media1.com media2.com; 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 media1.com and media2.com (and not from subdomains of those sites).
* Executable script is only allowed from userscripts.example.com.

### [Example 4](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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.jumbobank.com

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

### [Example 5](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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' \*.mailsite.com; img-src \*

Note that this example doesn't specify a [script-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src); with the example CSP, this site uses the setting specified by the [default-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/default-src) directive, which means that scripts can be loaded only from the originating server.

## [Testing your policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy-Report-Only) header and a [Content-Security-Policy](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#enabling_reporting)

By default, violation reports aren't sent. To enable violation reporting, you need to specify the [report-uri](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/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](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP#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>

<head>

<title>Sign Up</title>

<link rel="stylesheet" href="css/style.css">

</head>

<body>

... 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](https://www.w3.org/TR/CSP/#security-violation-reports) of this odd behavior. In summary, this is done to prevent leaking sensitive information about cross-origin resources.