

Content Security Policy Level 2

W3C Recommendation, 15 December 2016

**This version:**

<https://www.w3.org/TR/2016/REC-CSP2-20161215/>

Latest version:

<https://www.w3.org/TR/CSP2/>

Latest version in series:

<https://www.w3.org/TR/CSP/>

Editor's Draft:

<https://w3c.github.io/webappsec-csp/>

Previous Versions:

<https://www.w3.org/TR/2016/PR-CSP2-20161108/>

<https://www.w3.org/TR/2015/CR-CSP2-20150721/>

<https://www.w3.org/TR/2015/CR-CSP2-20150219/>

<https://www.w3.org/TR/2014/WD-CSP2-20140703/>

<https://www.w3.org/TR/2014/WD-CSP11-20140211/>

<https://www.w3.org/TR/2012/CR-CSP-20121115/>

Implementation Report

https://w3c.github.io/webappsec/implementation_reports/CSP2_implementation_report.html

Feedback:

public-webappsec@w3.org with subject line “[CSP2] ... message topic ...” ([archives](#))

Issue Tracking:

[GitHub](#)

Editors:

[Mike West](#) (Google Inc.)

[Adam Barth](#) (Google Inc.)

[Dan Veditz](#) (Mozilla Corporation)

Former Editors:

[Brandon Sterne](#) (formerly of Mozilla Corporation)

[Errata](#) for this document are recorded as issues.

The English version of this specification is the only normative version. Non-normative [translations](#) may also be available.

Abstract

This document defines a policy language used to declare a set of content restrictions for a web resource, and a mechanism for transmitting the policy from a server to a client where the policy is enforced.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index at https://www.w3.org/TR/](https://www.w3.org/TR/).

Errata for this document are [recorded as issues](#). The latest [CSP editors' draft](#) shows current proposed resolution of errata *in situ*.

This document was published by the [Web Application Security Working Group](#) as a Recommendation. The Working Group expects CSP Level 3 to obsolete this Recommendation. If you wish to make comments regarding this document, please raise an issue [in the specification's issue tracker](#). Historical comments may also be found in the working group's [email archives](#).

Please see the Working Group's [implementation report](#).

This document has been reviewed by W3C Members, by software developers, and by other W3C groups and interested parties, and is endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document was produced by a group operating under the [5 February 2004 W3C Patent Policy](#). W3C maintains a [public list of any patent disclosures](#) made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains [Essential Claim\(s\)](#) must disclose the information in accordance with [section 6 of the W3C Patent Policy](#).

This document is governed by the [1 September 2015 W3C Process Document](#).

W3C expects the functionality specified in this Recommendation will not be affected by changes to

referenced documents at an earlier process stage than Proposed Recommendation. Many of the referenced specifications are at Working Draft status; implementors of CSP2 should be aware that the mechanisms cited have content security implications and should track the progress of those specifications as they are included in CSP implementations.

Development of CSP Level 2 concluded in 2014. Implementors of user-agents are strongly encouraged to base their work on [Content Security Policy Level 3](#).

Table of Contents

1	Introduction
1.1	Changes from Level 1
2	Key Concepts and Terminology
2.1	Terms defined by this specification
2.2	Terms defined by reference
2.3	Relevant Concepts from HTML
2.4	Grammatical Concepts
3	Policy Delivery
3.1	Content-Security-Policy Header Field
3.2	Content-Security-Policy-Report-Only Header Field
3.3	HTML meta Element
3.4	Enforcing multiple policies.
3.5	Policy applicability
4	Syntax and Algorithms
4.1	Policy Syntax
4.1.1	Parsing Policies
4.2	Source List Syntax
4.2.1	Parsing Source Lists
4.2.2	Matching Source Expressions
4.2.2.1	Security Considerations for GUID URL schemes
4.2.2.2	Path Matching
4.2.2.3	Paths and Redirects
4.2.3	The nonce attribute
4.2.4	Valid Nonces
4.2.5	Valid Hashes

4.3 Media Type List Syntax

4.3.1 Parsing

4.3.2 Matching

4.4 Reporting

5 Processing Model

5.1 Workers

5.2 srcdoc IFrames

6 Script Interfaces

6.1 SecurityPolicyViolationEvent Interface

6.2 SecurityPolicyViolationEventInit Interface

6.3 Firing Violation Events

7 Directives

7.1 base-uri

7.2 child-src

7.2.1 Nested Browsing Contexts

7.2.2 Workers

7.3 connect-src

7.3.1 Usage

7.4 default-src

7.4.1 Usage

7.5 font-src

7.6 form-action

7.7 frame-ancestors

7.7.1 Relation to X-Frame-Options

7.7.2 Multiple Host Source Values

7.8 frame-src

7.9 img-src

7.10 media-src

7.11 object-src

7.12 plugin-types

7.12.1 Usage

7.12.2 Predeclaration of expected media types

7.13 report-uri

7.14 sandbox

7.14.1 Sandboxing and Workers

- 7.14.2 Usage
- 7.15 `script-src`
 - 7.15.1 Nonce usage for `script` elements
 - 7.15.2 Hash usage for `script` elements
- 7.16 `style-src`
 - 7.16.1 Nonce usage for `style` elements
 - 7.16.2 Hash usage for `style` elements

8 Examples

- 8.1 Sample Policy Definitions
- 8.2 Sample Violation Report

9 Security Considerations

- 9.1 Cascading Style Sheet (CSS) Parsing
- 9.2 Redirect Information Leakage

10 Implementation Considerations

- 10.1 Processing Complications

11 IANA Considerations

- 11.1 Content-Security-Policy
- 11.2 Content-Security-Policy-Report-Only

12 Acknowledgements

Conformance

Document conventions
Conformant Algorithms
Conformance Classes

Index

Terms defined by this specification
Terms defined by reference

References

Normative References
Informative References

IDL Index

1. Introduction

This section is not normative.

This document defines Content Security Policy, a mechanism web applications can use to mitigate a broad class of content injection vulnerabilities, such as cross-site scripting (XSS). Content Security Policy is a declarative policy that lets the authors (or server administrators) of a web application inform the client about the sources from which the application expects to load resources.

To mitigate XSS attacks, for example, a web application can declare that it only expects to load script from specific, trusted sources. This declaration allows the client to detect and block malicious scripts injected into the application by an attacker.

Content Security Policy (CSP) is not intended as a first line of defense against content injection vulnerabilities. Instead, CSP is best used as defense-in-depth, to reduce the harm caused by content injection attacks. As a first line of defense against content injection, server operators should validate their input and encode their output.

There is often a non-trivial amount of work required to apply CSP to an existing web application. To reap the greatest benefit, authors will need to move all inline script and style out-of-line, for example into external scripts, because the user agent cannot determine whether an inline script was injected by an attacker.

To take advantage of CSP, a web application opts into using CSP by supplying a Content-Security-Policy HTTP header. Such policies apply to the current resource representation only. To supply a policy for an entire site, the server needs to supply a policy with each resource representation.

1.1. Changes from Level 1

This document describes an evolution of the [Content Security Policy specification](#). Level 2 makes two breaking changes from Level 1, and adds support for a number of new directives and capabilities which are summarized below:

1. The following changes are backwards incompatible with the majority of user agent's implementations of CSP 1:
 1. The path component of a source expression is now ignored if the resource being loaded is the result of a redirect, as described in [§4.2.2.3 Paths and Redirects](#).

Note: Paths are technically new in CSP2, but they were already implemented in many user agents before this revision of CSP was completed, so noting the change here seems reasonable.

2. A [protected resource](#)'s ability to load Workers is now controlled via [child-src](#) rather than [script-src](#).
3. Workers now have their own policy, separate from the [protected resource](#) which loaded them. This is described in [§5.1 Workers](#).
2. The following directives are brand new in this revision:
 1. [base-uri](#) controls the [protected resource](#)'s ability to specify the [document base URL](#).
 2. [child-src](#) deprecates and replaces [frame-src](#), controlling the [protected resource](#)'s ability to embed frames, and to load Workers.
 3. [form-action](#) controls the [protected resource](#)'s ability to submit forms.
 4. [frame-ancestors](#) controls the [protected resource](#)'s ability be embedded in other documents. It is meant to supplant the X-Frame-Options HTTP request header.
 5. [plugin-types](#) controls the [protected resource](#)'s ability to load specific types of plugins.
3. Individual inline scripts and stylesheets may be whitelisted via nonces (as described in [§4.2.4 Valid Nonces](#)) and hashes (as described in [§4.2.5 Valid Hashes](#)).
4. A [SecurityPolicyViolationEvent](#) is fired upon violations, as described in [§6.3 Firing Violation Events](#).
5. A number of new fields were added to violation reports (both those POSTED via [report-uri](#), and those handed to the DOM via [SecurityPolicyViolationEvent](#) events. These include [effectiveDirective](#), [statusCode](#), [sourceFile](#), [lineNumber](#), and [columnNumber](#).
6. Certain flags present in the [sandbox](#) directive now affect Worker creation, as described in [§7.14.1 Sandboxing and Workers](#).

2. Key Concepts and Terminology

2.1. Terms defined by this specification

security policy

security policy directive

security policy directive name

security policy directive value

A **security policy** refers to both a set of security preferences for restrictions within which content

can operate, and to a fragment of text that codifies or transmits these preferences. For example, the following string is a policy which restricts script and object content:

```
script-src 'self'; object-src 'none'
```

Security policies contain a set of **security policy directives** ([script-src](#) and [object-src](#) in the example above), each responsible for declaring the restrictions for a particular resource type, or manipulating a specific aspect of the policy's restrictions. The list of directives defined by this specification can be found in [§7 Directives](#).

Each directives has a **name** and a **value**; a detailed grammar can be found in [§4 Syntax and Algorithms](#).

protected resource

A [security policy](#) is applied by a user agent to a specific [resource representation](#), known as the **protected resource**. See [§3 Policy Delivery](#) for details regarding the mechanisms by which policies may be applied to a protected resource.

2.2. Terms defined by reference

globally unique identifier

Defined in [Section 2.3 of the Origin specification](#). [\[RFC6454\]](#)

NOTE: URLs which do not use hierarchical elements as naming authorities (`data:`, for instance) have [origins](#) which are globally unique identifiers.

HTTP 200 response

Defined in [Section 6.3.1 of HTTP/1.1 -- Semantics and Content](#). [\[RFC7231\]](#)

JSON object

JSON stringification

Defined in the JSON specification. [\[RFC4627\]](#)

origin

Defined by the Origin specification. [\[RFC6454\]](#)

resource representation

Defined in [Section 3 of HTTP/1.1 -- Semantics and Content](#). [\[RFC7231\]](#)

URL

Defined by [\[URL\]](#).

SHA-256

SHA-384

SHA-512

These digest algorithms are defined by the NIST. [\[FIPS180\]](#)

2.3. Relevant Concepts from HTML

The [applet](#), [audio](#), [embed](#), [iframe](#), [img](#), [link](#), [object](#), [script](#), [source](#), [track](#), and [video](#) are defined in [\[HTML5\]](#).

The terms [auxiliary browsing contexts](#), [opener browsing context](#), and [nested browsing contexts](#) are defined in the HTML5 specification. [\[HTML5\]](#)

A [plugin](#) is defined in the HTML5 specification. [\[HTML5\]](#)

The <<@font-face>> Cascading Style Sheets (CSS) rule is defined in the CSS Fonts Module Level 3 specification. [\[CSS3-FONTS\]](#)

The XMLHttpRequest object is defined in the XMLHttpRequest specification. [\[XMLHTTPREQUEST\]](#)

The WebSocket object is defined in the WebSocket specification. [\[WEBSOCKETS\]](#)

The EventSource object is defined in the EventSource specification. [\[EVENTSOURCE\]](#)

The *runs a worker* algorithm is [defined in the Web Workers spec](#). [\[WORKERS\]](#)

The term *callable* refers to an object whose interface has one or more *callers* as defined in the [Web IDL](#) specification [\[WEBIDL\]](#).

2.4. Grammatical Concepts

The Augmented Backus-Naur Form (ABNF) notation used in this document is specified in RFC5234. [\[ABNF\]](#)

This document also uses the ABNF extension "#rule" as defined in [Section 7](#) of HTTP/1.1 -- Message Syntax and Routing. [\[RFC7230\]](#)

The following core rules are included by reference, as defined in [Appendix B.1](#) of [\[ABNF\]](#): *ALPHA* (letters), *DIGIT* (decimal 0-9), *WSP* (white space) and *VCHAR* (printing characters).

3. Policy Delivery

The server delivers a [policy](#) to the user agent via an HTTP response header (defined in [§3.1 Content-Security-Policy Header Field](#) and [§3.2 Content-Security-Policy-Report-Only Header Field](#)) or an HTML [meta](#) element (defined in [§3.3 HTML meta Element](#)).

3.1. Content-Security-Policy Header Field

The **Content-Security-Policy** header field is the preferred mechanism for delivering a policy. The grammar is as follows:

"Content-Security-Policy:" 1#[policy-token](#)

For example, a response might include the following header field:

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

A server **MUST NOT** send more than one HTTP header field named Content-Security-Policy with a given [resource representation](#).

A server **MAY** send different Content-Security-Policy header field values with different [representations](#) of the same resource or with different resources.

Upon receiving an HTTP response containing at least one Content-Security-Policy header field, the user agent **MUST** [enforce](#) each of the policies contained in each such header field.

3.2. Content-Security-Policy-Report-Only Header Field

The **Content-Security-Policy-Report-Only** header field lets servers experiment with policies by monitoring (rather than enforcing) a policy. The grammar is as follows:

"Content-Security-Policy-Report-Only:" 1#[policy-token](#)

For example, server operators might wish to develop their security policy iteratively. The operators can deploy a report-only policy based on their best estimate of how their site behaves:

```
Content-Security-Policy-Report-Only: script-src 'self';  
                                     report-uri /csp-report-endpoint/
```

If their site violates this policy the user agent will [send violation reports](#) to the URL specified in the policy's [report-uri](#) directive, but allow the violating resources to load regardless. Once a site has confidence that the policy is appropriate, they can start enforcing the policy using the [Content-Security-Policy](#) header field.

A server MUST NOT send more than one HTTP header field named Content-Security-Policy-Report-Only with a given [resource representation](#).

A server MAY send different Content-Security-Policy-Report-Only header field values with different [representations](#) of the same resource or with different resources.

Upon receiving an HTTP response containing at least one Content-Security-Policy-Report-Only header field, the user agent MUST [monitor](#) each of the policies contained in each such header field.

Note: The [Content-Security-Policy-Report-Only](#) header is *not* supported inside a [meta](#) element.

3.3. HTML [meta](#) Element

The server MAY supply policy via one or more HTML [meta](#) elements with [http-equiv](#) attributes that are an [ASCII case-insensitive match](#) for the string "Content-Security-Policy". For example:

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'">
```

Add the following entry to the [pragma directives](#) for the [meta](#) element:

Content security policy (http-equiv="content-security-policy")

1. If the Document's [head](#) element is not an ancestor of the [meta](#) element, abort these steps.
2. If the [meta](#) element lacks a [content](#) attribute, abort these steps.
3. Let *policy* be the value of the [content](#) attribute of the [meta](#) element.
4. Let *directive-set* be the result of [parsing policy](#).
5. Remove all occurrences of [report-uri](#), [frame-ancestors](#), and [sandbox](#) directives from *directive-set*.

Note: User agents are encouraged to issue a warning to developers if one or more of these directives are included in a policy delivered via [meta](#).

6. Enforce each of the [directives](#) in *directive-set*, as [defined for each directive type](#).

Authors are *strongly encouraged* to place [meta](#) elements as early in the document as possible, because policies in [meta](#) elements are not applied to content which precedes them. In particular, note that resources fetched or prefetched using the Link HTTP response header field, and resources fetched or prefetched using [link](#) and [script](#) elements which precede a [meta](#)-delivered policy will not be blocked.

Note: A [policy](#) specified via a [meta](#) element will be enforced along with any other policies active for the protected resource, regardless of where they're specified. The general impact of enforcing multiple policies is described in [§3.4 Enforcing multiple policies](#).

Note: Modifications to the [content](#) attribute of a [meta](#) element after the element has been parsed will be ignored.

Note: The [Content-Security-Policy-Report-Only](#) header is *not* supported inside a [meta](#) element.

3.4. Enforcing multiple policies.

This section is not normative.

The above sections note that when multiple policies are present, each must be enforced or reported, according to its type. An example will help clarify how that ought to work in practice. The behavior of an XMLHttpRequest might seem unclear given a site that, for whatever reason, delivered the following HTTP headers:

```
Content-Security-Policy: default-src 'self' http://example.com http://example.net;  
                        connect-src 'none';  
Content-Security-Policy: connect-src http://example.com/;  
                        script-src http://example.com/
```

Is a connection to `example.com` allowed or not? The short answer is that the connection is not allowed. Enforcing both policies means that a potential connection would have to pass through both unscathed. Even though the second policy would allow this connection, the first policy contains [connect-src](#) 'none', so its enforcement blocks the connection. The impact is that adding additional policies to the list of policies to enforce can only further restrict the capabilities of the protected resource.

To demonstrate that further, consider a script tag on this page. The first policy would lock scripts

down to 'self', <http://example.com> and <http://example.net> via the [default-src](#) directive. The second, however, would only allow script from <http://example.com/>. Script will only load if it meets both policy's criteria: in this case, the only origin that can match is <http://example.com>, as both policies allow it.

3.5. Policy applicability

This section is not normative.

Policies are associated with an [protected resource](#), and [enforced](#) or [monitored](#) for that resource. If a resource does not create a new execution context (for example, when including a script, image, or stylesheet into a document), then any policies delivered with that resource are discarded without effect. Its execution is subject to the policy or policies of the including context. The following table outlines examples of these relationships:

Resource Type		What policy applies?
Top-level Contexts	HTML as a new, top-level browsing context	The policy delivered with the resource
	SVG, as a top-level document	Policy delivered with the resource
Embedded Contexts	Any resource included via iframe , object , or embed	The policy of the embedding resource controls <i>what</i> may be embedded. The embedded resource, however, is controlled by the policy delivered with the resource, or the policy of the embedding resource if the embedded resource is a globally unique identifier (or a srcdoc frame).
	SVG, as an embedded document	The policy delivered with the resource, or policy of the creating context if created from a globally unique identifier .
	JavaScript, as a Worker, Shared Worker or Service Worker	The policy delivered with the resource, or policy of the creating context if created from a globally unique identifier
Subresources	SVG, inlined via svg	Policy of the including context

Resource Type	What policy applies?
SVG, as a resource document	Policy of the including context
HTML via XMLHttpRequest	Policy of the context that performed the fetch
Image via img element	Policy of the including context
JavaScript via a script element	Policy of the including context
SVG, via img	No policy; should be just as safe as JPG
SVG, as a WebFont	No policy; should be just as safe as WOFF

4. Syntax and Algorithms

4.1. Policy Syntax

A Content Security Policy consists of a U+003B SEMICOLON (;) delimited list of directives. Each [directive](#) consists of a [directive name](#) and (optionally) a [directive value](#), defined by the following ABNF:

```

policy-token    = [ directive-token *( ";" [ directive-token ] ) ]
directive-token = *WSP [ directive-name [ WSP directive-value ] ]
directive-name  = 1*( ALPHA / DIGIT / "-" )
directive-value = *( WSP / <VCHAR except ";" and ">" )

```

4.1.1. Parsing Policies

To *parse the policy* *policy*, the user agent MUST use an algorithm equivalent to the following:

1. Let the *set of directives* be the empty set.
2. For each non-empty token returned by [strictly splitting](#) the string *policy* on the character U+003B SEMICOLON (;):
 1. [Skip whitespace](#).

2. [Collect a sequence of characters](#) that are not [space characters](#). The collected characters are the *directive name*.
 3. If there are characters remaining in *token*, skip ahead exactly one character (which must be a [space character](#)).
 4. The remaining characters in *token* (if any) are the *directive value*.
 5. If the *set of directives* already contains a directive whose name is a case insensitive match for *directive name*, ignore this instance of the directive and continue to the next token.
 6. Add a *directive* to the *set of directives* with name *directive name* and value *directive value*.
3. Return the *set of directives*.

4.2. Source List Syntax

Many CSP directives use a value consisting of a **source list**, defined in the ABNF grammar below.

Each **source expression** in the source list represents a location from which content of the specified type can be retrieved. For example, the source expression 'none' represents the empty set of URLs, and the source expression 'unsafe-inline' represents content supplied inline in the resource itself.

```

source-list      = *WSP [ source-expression *( 1*WSP source-expression ) *WSP ]
                  / *WSP "'none'" *WSP
source-expression = scheme-source / host-source / keyword-source / nonce-source / hash-source
scheme-source    = scheme-part ":"
host-source      = [ scheme-part "://" ] host-part [ port-part ] [ path-part ]
keyword-source   = "'self'" / "'unsafe-inline'" / "'unsafe-eval'"
base64-value     = 1*( ALPHA / DIGIT / "+" / "/" ) *2( "=" )
nonce-value      = base64-value
hash-value       = base64-value
nonce-source     = "'nonce-' nonce-value '"
hash-algo        = "sha256" / "sha384" / "sha512"
hash-source      = "' ' hash-algo '-' hash-value '"
scheme-part      = <scheme production from RFC 3986, section 3.1>
host-part        = "*" / [ "*" ] 1*host-char *( "." 1*host-char )
host-char        = ALPHA / DIGIT / "-"
path-part        = <path production from RFC 3986, section 3.3>
port-part        = ":" ( 1*DIGIT / "*" )

```

If the policy contains a [nonce-source](#) expression, the server MUST generate a fresh value for the [nonce-value](#) directive at random and independently each time it transmits a policy. The generated value SHOULD be at least 128 bits long (before encoding), and generated via a cryptographically

secure random number generator. This requirement ensures that the [nonce-value](#) is difficult for an attacker to predict.

Note: Using a nonce to whitelist inline script or style is less secure than not using a nonce, as nonces override the restrictions in the directive in which they are present. An attacker who can gain access to the nonce can execute whatever script they like, whenever they like. That said, nonces provide a substantial improvement over 'unsafe-inline' when layering a content security policy on top of old code. When considering 'unsafe-inline', authors are encouraged to consider nonces (or hashes) instead.

The [host-char](#) production intentionally contains only ASCII characters; internationalized domain names cannot be entered directly into a policy string, but instead MUST be Punycode-encoded [\[RFC3492\]](#). For example, the domain `üüüüüü.de` would be encoded as `xn--tdaaaaaa.de`.

NOTE: Though IP addresses do match the grammar above, only `127.0.0.1` will actually match a URL when used in a source expression (see [§4.2.2 Matching Source Expressions](#) for details). The security properties of IP addresses are suspect, and authors ought to prefer hostnames to IP addresses whenever possible.

4.2.1. Parsing Source Lists

To *parse a source list* *source list*, the user agent MUST use an algorithm equivalent to the following:

1. [Strip leading and trailing whitespace](#) from *source list*.
2. If *source list* is an [ASCII case-insensitive match](#) for the string 'none' (including the quotation marks), return the empty set.
3. Let *set of source expressions* be the empty set.
4. For each token returned by [splitting source list on spaces](#), if the token matches the grammar for [source-expression](#), add the token to the *set of source expressions*.
5. Return the *set of source expressions*.

Note: Characters like U+003B SEMICOLON (;) and U+002C COMMA (,) cannot appear in source expressions directly: if you'd like to include these characters in a source expression, they must be [percent encoded](#) as %3B and %2C respectively.

4.2.2. Matching Source Expressions

A URL *url* is said to **match a source expression** for a *protected resource* if the following algorithm returns *does match*:

1. Let *url* be the result of processing the URL through the [URL parser](#).
2. If the source expression consists of a single U+002A ASTERISK character (*), and *url*'s [scheme](#) is not one of blob, data, filesystem, then return *does match*.
3. If the source expression matches the grammar for [scheme-source](#):
 1. If *url*'s [scheme](#) is an [ASCII case-insensitive match](#) for the source expression's [scheme-part](#), return *does match*.
 2. Otherwise, return *does not match*.
4. If the source expression matches the grammar for [host-source](#):
 1. If *url*'s [host](#) is null, return *does not match*.
 2. Let *url-scheme*, *url-host*, and *url-port* be the [scheme](#), [host](#), and [port](#) of *url*'s origin, respectively.

Note: If *url* doesn't specify a port, then its origin's port will be the [default port](#) for *url*'s [scheme](#).

3. Let *url-path-list* be the [path](#) of *url*.
4. If the source expression has a [scheme-part](#) that is not a case insensitive match for *url-scheme*, then return *does not match*.
5. If the source expression does **not** have a scheme, return *does not match* if any of the following are true:
 1. the scheme of the protected resource's URL is a case insensitive match for HTTP, and *url-scheme* is **not** a case insensitive match for either HTTP or HTTPS
 2. the scheme of the protected resource's URL is **not** a case insensitive match for HTTP, and *url-scheme* is **not** a case insensitive match for the scheme of the protected resource's URL.
6. If the first character of the source expression's [host-part](#) is an U+002A ASTERISK character (*) and the remaining characters, including the leading U+002E FULL STOP character (.), are not a case insensitive match for the rightmost characters of *url-host*, then return *does not match*.
7. If the first character of the source expression's [host-part](#) is *not* an U+002A ASTERISK character (*) and *url-host* is not a case insensitive match for the source expression's [host-part](#), then return *does not match*.

8. If the source expression's [host-part](#) matches the [IPv4address](#) production from [\[RFC3986\]](#), and is not 127.0.0.1, or is an [IPv6 address](#), return *does not match*.

Note: A future version of this specification may allow literal IPv6 and IPv4 addresses, depending on usage and demand. Given the weak security properties of IP addresses in relation to named hosts, however, authors are encouraged to prefer the latter whenever possible.

9. If the source expression does **not** contain a port-part and *url-port* is not the [default port](#) for *url-scheme*, then return *does not match*.
10. If the source expression does contain a port-part, then return *does not match* if both of the following are true:
 1. [port-part](#) does **not** contain an U+002A ASTERISK character (*)
 2. [port-part](#) does **not** represent the same number as *url-port*
11. If the source expression contains a non-empty [path-part](#), and the URL is *not* the result of a redirect, then:
 1. Let *exact-match* be true if the final character of *path-part* is not the U+002F SOLIDUS character (/), and false otherwise.
 2. Let *source-expression-path-list* be the result of splitting *path-part* on the U+002F SOLIDUS character (/).
 3. If *source-expression-path-list*'s length is greater than *url-path-list*'s length, return *does not match*.
 4. For each entry in *source-expression-path-list*:
 1. [Percent decode](#) entry.
 2. [Percent decode](#) the first item in *url-path-list*.
 3. If entry is not an [ASCII case-insensitive match](#) for the first item in *url-path-list*, return *does not match*.
 4. Pop the first item in *url-path-list* off the list.
 5. If *exact-match* is true, and *url-path-list* is not empty, return *does not match*.
12. Otherwise, return *does match*.
5. If the source expression is a case insensitive match for 'self' (including the quotation marks), then:
 1. Return *does match* if [the origin of url](#) matches [the origin of protected resource's URL](#).

Note: This includes IP addresses. That is, a document at `https://111.111.111.111/` with a [policy](#) of `img-src 'self'` can load the image `https://111.111.111.111/image.png`, as the origins match.

6. Otherwise, return *does not match*.

Note: This algorithm treats the URLs `https://example.com/` and `https://example.com./` as *non-matching*. This is consistent with browser behavior which treats documents served from these URLs as existing in distinct origins.

A URL *url* is said to ***match a source list*** for *protected resource* if at least one source expression in the set of source expressions obtained by [parsing the source list matches url for protected resource](#).

Note: No URLs match an empty set of source expressions, such as the set obtained by parsing the source list `'none'`.

4.2.2.1. Security Considerations for GUID URL schemes

This section is not normative.

As defined above, special URL schemes that refer to specific pieces of unique content, such as `"data:"`, `"blob:"` and `"filesystem:"` are excluded from matching a policy of `*` and must be explicitly listed. Policy authors should note that the content of such URLs is often derived from a response body or execution in a Document context, which may be unsafe. Especially for the [default-src](#) and [script-src](#) directives, policy authors should be aware that allowing `"data:"` URLs is equivalent to `unsafe-inline` and allowing `"blob:"` or `"filesystem:"` URLs is equivalent to `unsafe-eval`.

4.2.2.2. Path Matching

This section is not normative.

The rules for matching source expressions that contain paths are simpler than they look: paths that end with the `'/'` character match all files in a directory and its subdirectories. Paths that do not end with the `'/'` character match only one specific file. A few examples should make this clear:

1. The source expression `example.com` has no path, and therefore matches any file served from that host.

2. The source expression `example.com/scripts/` matches any file in the `scripts` directory of `example.com`, and any of its subdirectories. For example, both `https://example.com/scripts/file.js` and `https://example.com/scripts/js/file.js` would match.
3. The source expression `example.com/scripts/file.js` matches only the file named `file.js` in the `scripts` directory of `example.com`.
4. Likewise, the source expression `example.com/js` matches only the file named `js`. In particular, note that it would not match files inside a directory named `js`. Files like `example.com/js/file.js` would be matched only if the source expression ended with a trailing `/`, as in `example.com/js/`.

Note: Query strings have no impact on matching: the source expression `example.com/file` matches all of `https://example.com/file`, `https://example.com/file?key=value`, `https://example.com/file?key=notvalue`, and `https://example.com/file?notkey=notvalue`.

4.2.2.3. *Paths and Redirects*

To avoid leaking path information cross-origin (as discussed in Egor Homakov's [Using Content-Security-Policy for Evil](#)), the matching algorithm ignores the path component of a source expression if the resource being loaded is the result of a redirect. For example, given a page with an active policy of `img-src example.com not-example.com/path`:

- Directly loading `https://not-example.com/not-path` would fail, as it doesn't match the policy.
- Directly loading `https://example.com/redirector` would pass, as it matches `example.com`.
- Assuming that `https://example.com/redirector` delivered a redirect response pointing to `https://not-example.com/not-path`, the load would succeed, as the initial URL matches `example.com`, and the redirect target matches `not-example.com/path` if we ignore its path component.

This restriction reduces the granularity of a document's policy when redirects are in play, which isn't wonderful, but given that we certainly don't want to allow brute-forcing paths after redirects, it seems a reasonable compromise.

The relatively long thread ["Remove paths from CSP?"](#) from `public-webappsec@w3.org` has more detailed discussion around alternate proposals.

4.2.3. The `nonce` attribute

Nonce sources require a new `nonce` attribute to be added to both `script` and `style` elements.

```
partial interface HTMLScriptElement {
  attribute DOMString nonce;
};
```

nonce, of type `DOMString`

This attribute [reflects](#) the value of the element's *nonce* content attribute.

```
partial interface HTMLStyleElement {
  attribute DOMString nonce;
};
```

nonce, of type `DOMString`

This attribute [reflects](#) the value of the element's *nonce* content attribute.

4.2.4. Valid Nonces

An element has a *valid nonce* for a *set of source expressions* if the value of the element's `nonce` attribute after [stripping leading and trailing whitespace](#) is a case-sensitive match for the `nonce-value` component of at least one `nonce-source` expression in *set of source expressions*.

4.2.5. Valid Hashes

An *element's content* is the `script block's source` for `script` elements, or the value of the element's `textContent` IDL attribute for non-`script` elements such as `style`.

The *digest of element's content* for is the result of applying an *algorithm* to the `element's content`.

To determine whether *element* has a *valid hash* for a *set of source expressions*, execute the following steps:

1. Let *hashes* be a list of all `hash-source` expressions in *set of source expressions*.
2. For each *hash* in *hashes*:
 1. Let *algorithm* be:
 - [SHA-256](#) if the `hash-algo` component of *hash* is an [ASCII case-insensitive match](#) for the string "sha256"
 - [SHA-384](#) if the `hash-algo` component of *hash* is an [ASCII case-insensitive match](#) for

the string "sha384"

- [SHA-512](#) if the [hash-algo](#) component of *hash* is an [ASCII case-insensitive match](#) for the string "sha512"

2. Let *expected* be the [hash-value](#) component of *hash*.
 3. Let *actual* be the [base64 encoding](#) of the binary [digest of element's content](#) using the *algorithm* algorithm.
 4. If *actual* is a case-sensitive match for *expected*, return **true** and abort these steps.
3. Return **false**.

Note: If an element has an invalid hash, it would be helpful if the user agent reported the failure to the author by adding a warning message containing the *actual* hash value.

4.3. Media Type List Syntax

The [plugin-types](#) directive uses a value consisting of a *media type list*.

Each *media type* in the media type list represents a specific type of resource that can be retrieved and used to instantiate a [plugin](#) in the protected resource.

```
media-type-list   = media-type *( 1*WSP media-type )
media-type       = <type from RFC 2045> "/" <subtype from RFC 2045>
```

4.3.1. Parsing

To *parse a media type list* *media type list*, the user agent MUST use an algorithm equivalent to the following:

1. Let the *set of media types* be the empty set.
2. For each token returned by [splitting media type list on spaces](#), if the token matches the grammar for [media-type](#), add the token to the *set of media types*. Otherwise ignore the token.
3. Return the *set of media types*.

4.3.2. Matching

A media type *matches a media type list* if, and only if, the media type is an [ASCII case-insensitive](#)

[match](#) for at least one token in the set of media types obtained by [parsing the media type list](#).

4.4. Reporting

To *strip uri for reporting*, the user agent MUST use an algorithm equivalent to the following:

1. If the [origin](#) of *uri* is a [globally unique identifier](#) (for example, *uri* has a scheme of `data`, `blob`, or `filesystem`), then abort these steps, and return the ASCII serialization of *uri*'s scheme.
2. If the [origin](#) of *uri* is not the same as the [origin](#) of the protected resource, then abort these steps, and return the [ASCII serialization of uri's origin](#).
3. Return *uri*, with any [fragment](#) component removed.

To *generate a violation report object*, the user agent MUST use an algorithm equivalent to the following:

1. Prepare a JSON object *violation* with the following keys and values:

[blocked-uri](#)

The originally requested URL of the resource that was prevented from loading, [stripped for reporting](#), or the empty string if the resource has no URL (inline script and inline style, for example).

[document-uri](#)

The [address](#) of the protected resource, [stripped for reporting](#).

[effective-directive](#)

The name of the policy directive that was violated. This will contain the [directive](#) whose enforcement triggered the violation (e.g. "[script-src](#)") even if that directive does not explicitly appear in the policy, but is implicitly activated via the [default-src](#) directive.

[original-policy](#)

The original [policy](#), as received by the user agent.

[referrer](#)

The [referrer](#) attribute of the protected resource, or the empty string if the protected resource has no referrer.

[status-code](#)

The status-code of the HTTP response that contained the protected resource, if the protected resource was obtained over HTTP. Otherwise, the number 0.

[violated-directive](#)

The policy directive that was violated, as it appears in the policy. This will contain the [default-src](#) directive in the case of violations caused by falling back to the [default sources](#) when enforcing a directive.

2. If a specific line or a specific file can be identified as the cause of the violation (for example, script execution that violates the [script-src](#) directive), the user agent MAY add the following keys and values to *violation*:

source-file

The URL of the resource where the violation occurred, [stripped for reporting](#).

line-number

The line number in [source-file](#) on which the violation occurred.

column-number

The column number in [source-file](#) on which the violation occurred.

3. Return *violation*.

Note: `blocked-uri` will not contain the final location of a resource that was blocked after one or more redirects. It instead will contain only the location that the protected resource requested, before any redirects were followed.

To ***send violation reports***, the user agent MUST use an algorithm equivalent to the following:

1. Prepare a [JSON object](#) *report object* with a single key, `csp-report`, whose value is the result of [generating a violation report object](#).
2. Let *report body* be the [JSON stringification](#) of *report object*.
3. For each *report URL* in the [set of report URLs](#):
 1. If the user agent has already sent a violation report for the protected resource to *report URL*, and that report contained an entity body that exactly matches *report body*, the user agent MAY abort these steps and continue to the next *report URL*.
 2. [Queue a task](#) to [fetch](#) *report URL* from the origin of the protected resource, with the synchronous flag *not* set, using HTTP method POST, with a Content-Type header field of `application/csp-report`, and an entity body consisting of *report body*. If the origin of *report URL* is **not** the same as the origin of the protected resource, the block cookies flag MUST also be set. The user agent MUST NOT follow redirects when fetching this resource. (Note: The user agent ignores the fetched resource.) The [task source](#) for these [tasks](#) is the ***Content Security Policy task source***.

To ***report a violation***, the user agent MUST:

1. [Fire a violation event](#) at the protected resource's [Document](#).
2. If the [set of report URLs](#) is non-empty, [send violation reports](#) to each.

Note: This section of the specification should not be interpreted as limiting user agents' ability to apply restrictions to violation reports in order to limit data leakage above and beyond what these algorithms specify. For example, a user agent might offer users the option of disabling reporting entirely.

5. Processing Model

To **enforce** a policy, the user agent MUST [parse the policy](#) and enforce each of the directives contained in the policy, where the specific requirements for enforcing each directive are defined separately for each directive (See [§7 Directives](#), below).

Generally speaking, enforcing a directive prevents the protected resource from performing certain actions, such as loading scripts from URLs other than those indicated in a source list. These restrictions make it more difficult for an attacker to abuse an injection vulnerability in the resource because the attacker will be unable to usurp the resource's privileges that have been restricted in this way.

Note: User agents may allow users to modify or bypass policy enforcement through user preferences, bookmarklets, third-party additions to the user agent, and other such mechanisms.

To **monitor** a policy, the user agent MUST [parse the policy](#) and monitor each of the directives contained in the policy.

Monitoring a directive does not prevent the protected resource from undertaking any actions. Instead, any actions that would have been prevented by the directives are allowed, but a violation report is [generated and reported](#) to the developer of the web application. Monitoring a policy is useful for testing whether enforcing the policy will cause the web application to malfunction.

A server MAY cause user agents to monitor one policy while enforcing another policy by returning both [Content-Security-Policy](#) and [Content-Security-Policy-Report-Only](#) header fields. For example, if a server operator may wish to [enforce](#) one policy but experiment with a stricter policy, she can monitor the stricter policy while enforcing the original policy. Once the server operator is satisfied that the stricter policy does not break the web application, the server operator can start enforcing the stricter policy.

If the user agent monitors or enforces a policy that does not contain any directives, the user agent SHOULD report a warning message in the developer console.

If the user agent [monitors](#) or [enforces](#) a policy that contains an unrecognized directive, the user agent

SHOULD report a warning message in the developer console indicating the name of the unrecognized directive.

5.1. Workers

Whenever a user agent [runs a worker](#):

- If the worker's script's origin is a [globally unique identifier](#) (for example, the worker's script's URL has a scheme of `data`, `blob`, or `filesystem`), then:
 - If the user agent is enforcing a CSP policy for the *owner document* or *parent worker*, the user agent MUST enforce the CSP policy for the worker.
 - If the user agent is monitoring a CSP policy for the *owner document* or *parent worker*, the user agent MUST monitor the CSP policy for the worker.
- Otherwise:
 - If the worker's script is delivered with a Content-Security-Policy HTTP header containing the value *policy*, the user agent MUST [enforce](#) *policy* for the worker.
 - If the worker's script is delivered with a Content-Security-Policy-Report-Only HTTP header containing the value *policy*, the user agent MUST [monitor](#) *policy* for the worker.

5.2. srcdoc IFrames

Whenever a user agent creates [an iframe srcdoc document](#) in a browsing context nested in the protected resource, if the user agent is [enforcing](#) any [policies](#) for the protected resource, the user agent MUST [enforce](#) those [policies](#) on the [iframe](#) srcdoc document as well.

Whenever a user agent creates [an iframe srcdoc document](#) in a browsing context nested in the protected resource, if the user agent is monitoring any policies for the protected resource, the user agent MUST [monitor](#) those policies on the [iframe](#) srcdoc document as well.

6. Script Interfaces

6.1. SecurityPolicyViolationEvent Interface

[**Constructor**(DOMString *type*, optional [SecurityPolicyViolationEventInit](#) *eventInitDict*);
interface **SecurityPolicyViolationEvent** : [Event](#) {

```

    readonly    attribute DOMString documentURI;
    readonly    attribute DOMString referrer;
    readonly    attribute DOMString blockedURI;
    readonly    attribute DOMString violatedDirective;
    readonly    attribute DOMString effectiveDirective;
    readonly    attribute DOMString originalPolicy;
    readonly    attribute DOMString sourceFile;
    readonly    attribute DOMString statusCode;
    readonly    attribute long      lineNumber;
    readonly    attribute long      columnNumber;
};

```

***documentURI*, of type [DOMString](#), readonly**

Refer to the [document-uri](#) property of violation reports for a description of this property.

***referrer*, of type [DOMString](#), readonly**

Refer to the [referrer](#) property of violation reports for a description of this property.

***blockedURI*, of type [DOMString](#), readonly**

Refer to the [blocked-uri](#) property of violation reports for a description of this property.

***violatedDirective*, of type [DOMString](#), readonly**

Refer to the [violated-directive](#) property of violation reports for a description of this property.

***effectiveDirective*, of type [DOMString](#), readonly**

Refer to the [effective-directive](#) property of violation reports for a description of this property.

***originalPolicy*, of type [DOMString](#), readonly**

Refer to the [original-policy](#) property of violation reports for a description of this property.

***statusCode*, of type [DOMString](#), readonly**

Refer to the [status-code](#) property of violation reports for a description of this property.

***sourceFile*, of type [DOMString](#), readonly**

Refer to the [source-file](#) property of violation reports for a description of this property.

***lineNumber*, of type [long](#), readonly**

Refer to the [line-number](#) property of violation reports for a description of this property.

***columnNumber*, of type [long](#), readonly**

Refer to the [column-number](#) property of violation reports for a description of this property.

6.2. [SecurityPolicyViolationEventInit](#) Interface

```

dictionary SecurityPolicyViolationEventInit : EventInit {
    DOMString documentURI;
    DOMString referrer;

```

```

    DOMString blockedURI;
    DOMString violatedDirective;
    DOMString effectiveDirective;
    DOMString originalPolicy;
    DOMString sourceFile;
    long      lineNumber;
    long      columnNumber;
};

```

***documentURI*, of type [DOMString](#)**

Refer to the [document-uri](#) property of violation reports for a description of this property.

***referrer*, of type [DOMString](#)**

Refer to the [referrer](#) property of violation reports for a description of this property.

***blockedURI*, of type [DOMString](#)**

Refer to the [blocked-uri](#) property of violation reports for a description of this property.

***violatedDirective*, of type [DOMString](#)**

Refer to the [violated-directive](#) property of violation reports for a description of this property.

***effectiveDirective*, of type [DOMString](#)**

Refer to the [effective-directive](#) property of violation reports for a description of this property.

***originalPolicy*, of type [DOMString](#)**

Refer to the [original-policy](#) property of violation reports for a description of this property.

***sourceFile*, of type [DOMString](#)**

Refer to the [source-file](#) property of violation reports for a description of this property.

***lineNumber*, of type [long](#)**

Refer to the [line-number](#) property of violation reports for a description of this property.

***columnNumber*, of type [long](#)**

Refer to the [column-number](#) property of violation reports for a description of this property.

6.3. Firing Violation Events

To ***fire a violation event***, the user agent MUST use an algorithm equivalent to the following:

1. Let *report object* be the result of [generating a violation report object](#).
2. [Queue a task](#) to [fire an event](#) named `securitypolicyviolation` using the [SecurityPolicyViolationEvent](#) interface with the following initializations:
 - `blockedURI` MUST be initialized to the value of *report object*'s `blocked-uri` key.
 - `documentURI` MUST be initialized to the value of *report object*'s `document-uri` key.

- `effectiveDirective` MUST be initialized to the value of *report object*'s `effective-directive` key.
- `originalPolicy` MUST be initialized to the value of *report object*'s `original-policy` key.
- `referrer` MUST be initialized to the value of *report object*'s `referrer` key.
- `violatedDirective` MUST be initialized to the value of *report object*'s `violated-directive` key.
- `sourceFile` MUST be initialized to the value of *report object*'s `source-file` key.
- `lineNumber` MUST be initialized to the value of *report object*'s `line-number` key.
- `columnNumber` MUST be initialized to the value of *report object*'s `column-number` key.

The [task source](#) for these [tasks](#) is the [Content Security Policy task source](#).

7. Directives

This section describes the content security policy directives introduced in this specification. Directive names are case insensitive.

In order to protect against Cross-Site Scripting (XSS), web application authors SHOULD include:

- both the [script-src](#) and [object-src](#) directives, or
- include a [default-src](#) directive, which covers both scripts and plugins.

In either case, authors SHOULD NOT include either `'unsafe-inline'` or `data:` as valid sources in their policies. Both enable XSS attacks by allowing code to be included directly in the document itself; they are best avoided completely.

7.1. base-uri

The ***base-uri*** directive restricts the URLs that can be used to specify the [document base URL](#). The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "base-uri"  
directive-value   = source-list
```

The term ***allowed base URLs*** refers to the result of [parsing the base-uri directive's value as a source list](#).

Note: `base-uri` does not fall back to the [default sources](#).

Step 4 of the algorithm defined in HTML5 to obtain a *document's base URL* MUST be changed to:

4. If the previous step was not successful, or the result of the previous step does not [match](#) the [allowed base URLs](#) for the [protected resource](#), then the [document base URL](#) is *fallback base URL*. Otherwise, it is the result of the previous step.

7.2. `child-src`

The ***child-src*** directive governs the creation of [nested browsing contexts](#) as well as Worker execution contexts. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name      = "child-src"
directive-value      = source-list
```

The term ***allowed child sources*** refers to the result of [parsing the `child-src` directive's value as a `source list`](#) if a `child-src` directive is explicitly specified, and otherwise to the [default sources](#).

7.2.1. Nested Browsing Contexts

To enforce the `child-src` directive the user agent MUST enforce the [frame-src](#) directive.

7.2.2. Workers

Whenever the user agent [fetches](#) a URL while processing the Worker or SharedWorker constructors [\[WORKERS\]](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#) if the URL does not [match](#) the [allowed child sources](#) for the [protected resource](#).

7.3. `connect-src`

The ***connect-src*** directive restricts which URLs the protected resource can load using script interfaces. The syntax for the name and value of the directive are described by the following ABNF grammar:

directive-name = "connect-src"
directive-value = [source-list](#)

The term ***allowed connection targets*** refers to the result of [parsing the connect-src directive's value as a source list](#) if the policy contains an explicit connect-src directive, or otherwise to the [default sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed connection targets](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Processing the [send\(\) method](#) of an XMLHttpRequest object.
- Processing the [WebSocket constructor](#).
- Processing the [EventSource constructor](#).
- Pinging an endpoint during [hyperlink auditing](#).
- Sending a beacon via the [sendBeacon\(\)](#) method [\[BEACON\]](#)

7.3.1. Usage

This section is not normative.

JavaScript offers a few mechanisms that directly connect to an external server to send or receive information. EventSource maintains an open HTTP connection to a server in order to receive push notifications, WebSockets open a bidirectional communication channel between your browser and a server, and XMLHttpRequest makes arbitrary HTTP requests on your behalf. These are powerful APIs that enable useful functionality, but also provide tempting avenues for data exfiltration.

The connect-src directive allows you to ensure that these sorts of connections are only opened to origins you trust. Sending a policy that defines a list of source expressions for this directive is straightforward. For example, to limit connections to only example.com, send the following header:

Content-Security-Policy: [connect-src](#) example.com

All of the following will fail with the preceding directive in place:

- `new WebSocket("wss://evil.com/");`
- `(new XMLHttpRequest()).open("GET", "https://evil.com/", true);`
- `new EventSource("https://evil.com");`

7.4. `default-src`

The **`default-src`** directive sets a default source list for a number of directives. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name      = "default-src"  
directive-value     = source-list
```

Let the ***default sources*** be the result of [parsing the `default-src` directive's value as a source list](#) if a `default-src` directive is explicitly specified, and otherwise the U+002A ASTERISK character (*).

To enforce the `default-src` directive, the user agent MUST enforce the following directives:

- [child-src](#)
- [connect-src](#)
- [font-src](#)
- [img-src](#)
- [media-src](#)
- [object-src](#)
- [script-src](#)
- [style-src](#)

If not specified explicitly in the policy, the directives listed above will use the [default sources](#) as their source list.

7.4.1. Usage

This section is not normative.

`default-src`, as the name implies, serves as a default source list which the other source list-style directives will use as a fallback if they're not otherwise explicitly set. That is, consider the following policy declaration:

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

Under this policy, fonts, frames, images, media, objects, scripts, and styles will all only load from the same origin as the protected resource, and connections will only be made to the same origin. Adding a more specific declaration to the policy would completely override the default source list for that resource type.

Content-Security-Policy: [default-src](#) 'self'; [script-src](#) example.com

Under this new policy, fonts, frames, and etc. continue to be load from the same origin, but scripts will *only* load from example.com. There's no inheritance; the [script-src](#) directive sets the allowed sources of script, and the default list is not used for that resource type.

Given this behavior, one good way of building a policy for a site would be to begin with a [default-src](#) of 'none', and to build up a policy from there that contains only those resource types which are actually in use for the page you'd like to protect. If you don't use webfonts, for instance, there's no reason to specify a source list for [font-src](#); specifying only those resource types a page uses ensures that the possible attack surface for that page remains as small as possible.

7.5. [font-src](#)

The **[font-src](#)** directive restricts from where the protected resource can load fonts. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "font-src"  
directive-value   = source-list
```

The term ***allowed font sources*** refers to the result of [parsing the font-src directive's value as a source list](#) if the policy contains an explicit font-src, or otherwise to the [default sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed font sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting data for display in a font, such as when processing the <<@font-face>> Cascading Style Sheets (CSS) rule.

7.6. [form-action](#)

The **[form-action](#)** restricts which URLs can be used as the action of HTML [form](#) elements. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "form-action"  
directive-value   = source-list
```

The term ***allowed form actions*** refers to the result of [parsing the form-action directive's value as a source list](#).

Whenever the user agent [fetches](#) a URL in the course of processing an HTML [form](#) element, if the URL does not [match](#) the [allowed form actions](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#).

Note: form-action does not fall back to the [default sources](#) when the directive is not defined. That is, a policy that defines [default-src](#) 'none' but not form-action will still allow form submissions to any target.

7.7. frame-ancestors

The ***frame-ancestors*** directive indicates whether the user agent should allow embedding the resource using a [frame](#), [iframe](#), [object](#), [embed](#) or [applet](#) element, or equivalent functionality in non-HTML resources. Resources can use this directive to avoid many UI Redressing [[UIREDRESS](#)] attacks by avoiding being embedded into potentially hostile contexts.

The syntax for the name and value of the directive are described by the following ABNF grammar:

```
ancestor-source-list = [ ancestor-source *( 1*WSP ancestor-source ) ] / "'none'"
ancestor-source      = scheme-source / host-source
```

```
directive-name  = "frame-ancestors"
directive-value = ancestor-source-list
```

The term ***allowed frame ancestors*** refers to the result of [parsing the frame-ancestors directive's value as a source list](#). If a frame-ancestors directive is not explicitly included in the policy, then [allowed frame ancestors](#) is `"*"`.

To enforce the frame-ancestors directive, whenever the user agent would load the protected resource into a [nested browsing context](#), the user agent MUST perform the following steps:

1. Let *nestedContext* be the nested browsing context into which the protected resource is being loaded.
2. Let *ancestorList* be the list of all [ancestors](#) of *nestedContext*.
3. For each *ancestorContext* in *ancestorList*:
 1. Let *document* be *ancestorContext*'s [active document](#).
 2. If *document*'s URL does not [match](#) the [allowed frame ancestors](#) for the [protected resource](#), the user agent MUST:
 1. Abort loading the protected resource.

2. Take one of the following actions:
 1. Act as if it received an empty [HTTP 200 response](#).
 2. Redirect the user to a friendly error page which provides the option of opening the blocked page in a new [top-level browsing context](#).
 3. [Parse a sandboxing directive](#) using the empty string as the *input* and the newly created document's [forced sandboxing flag set](#) as the *output*.
 4. [Report a violation](#).
 5. Abort these steps.

Steps 3.2.2 and 3.2.3 ensure that the blocked frame appears to be a normal cross-origin document's load. If these steps are ignored, leakage of a document's policy state is possible.

The `frame-ancestors` directive MUST be ignored when [monitoring](#) a policy, and when a contained in a policy defined via a [meta](#) element.

Note: [frame-ancestors](#) does not fall back to the [default sources](#) when the directive is not defined. That is, a policy that defines [default-src](#) 'none' but not `frame-ancestors` will still allow the resource to be framed from anywhere.

When generating a violation report for a `frame-ancestors` violation, the user agent MUST NOT include the value of the embedding ancestor as a `blocked-uri` value unless it is same-origin with the protected resource, as disclosing the value of cross-origin ancestors is a violation of the Same-Origin Policy.

7.7.1. Relation to X-Frame-Options

This directive is similar to the `X-Frame-Options` header that several user agents have implemented. The 'none' source expression is roughly equivalent to that header's `DENY`, 'self' to `SAMEORIGIN`, and so on. The major difference is that many user agents implement `SAMEORIGIN` such that it only matches against the top-level document's location. This directive checks each ancestor. If any ancestor doesn't match, the load is cancelled. [\[RFC7034\]](#)

The `frame-ancestors` directive *obsoletes* the `X-Frame-Options` header. If a resource has both policies, the `frame-ancestors` policy SHOULD be enforced and the `X-Frame-Options` policy SHOULD be ignored.

7.7.2. Multiple Host Source Values

This section is not normative.

Multiple source-list expressions are allowed in a single policy (in contrast to X-Frame-Options, which allows only one) to enable scenarios involving embedded application components that are multiple levels below the top-level browsing context.

Many common scenarios for permissioned embedding (e.g. embeddable payment, sharing or social apps) involve potentially many hundreds or thousands of valid source-list expressions, but it is strongly recommended against accommodating such scenarios with a static frame-ancestors directive listing multiple values. In such cases it is beneficial to generate this value dynamically, based on an HTTP Referer header or an explicitly passed-in value, to allow only the sources necessary for each given embedding of the resource.

Consider a service providing a payments application at `https://payments/makeEmbedded`. The service allows this resource to be embedded by both merchant Alice and merchant Bob, who compete with each other. Sending:

Content-Security-Policy: [frame-ancestors](#) https://alice https://bob

would allow Bob to re-frame Alice's resource and create fraudulent clicks, perhaps discrediting Alice with her customers or the payments service. If the payments service used additional information (e.g. as part of a URL like `https://payments/makeEmbedded?merchant=alice`) to send individually-tailored headers listing only the source-list expressions needed by each merchant, this attack would be eliminated.

7.8. frame-src

The ***frame-src*** directive is *deprecated*. Authors who wish to govern nested browsing contexts SHOULD use the `child-src` directive instead.

The `frame-src` directive restricts from where the protected resource can embed frames. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "frame-src"
directive-value    = source-list
```

The term ***allowed frame sources*** refers to the result of [parsing the frame-src directive's value as a source list](#) if the policy contains an explicit `frame-src`, or otherwise to the list of [allowed child sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL

does not [match](#) the [allowed frame sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting data for display in a [nested browsing context](#) in the protected resource created by an [iframe](#) or a [frame](#) element.
- [Navigated](#) such a [nested browsing context](#).

7.9. `img-src`

The **`img-src`** directive restricts from where the protected resource can load images. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "img-src"
directive-value   = source-list
```

The term ***allowed image sources*** refers to the result of [parsing the `img-src` directive's value as a source list](#) if the policy contains an explicit `img-src`, or otherwise to the list of [default sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed image sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting data for an image, such as when processing the [src](#) or `srcset` attributes of an [img](#) element, the [src](#) attribute of an [input](#) element with a type of [image](#), the [poster](#) attribute of a [video](#) element, the [url\(\)](#), [image\(\)](#) or [image-set\(\)](#) values on any Cascading Style Sheets (CSS) property that is capable of loading an image [\[CSS4-IMAGES\]](#), or the [href](#) attribute of a [link](#) element with an image-related [rel](#) attribute, such as [icon](#).

7.10. `media-src`

The **`media-src`** directive restricts from where the protected resource can load video, audio, and associated text tracks. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "media-src"
directive-value   = source-list
```

The term ***allowed media sources*** refers to the result of [parsing the `media-src` directive's value as a source list](#) if the policy contains an explicit `media-src`, or otherwise to the list of [default sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed media sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting data for a video or audio clip, such as when processing the [src](#) attribute of a [video](#), [audio](#), [source](#), or [track](#) element.

7.11. object-src

The ***object-src*** directive restricts from where the protected resource can load plugins. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name      = "object-src"
directive-value     = source-list
```

The term ***allowed object sources*** refers to the result of [parsing the object-src directive's value as a source list](#) if the policy contains an explicit `object-src`, or otherwise to the list of [default sources](#).

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed object sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting data for a plugin, such as when processing the [data](#) attribute of an [object](#) element, the [src](#) attribute of an [embed](#) element, or the [code](#) or [archive](#) attributes of an [applet](#) element.
- Requesting data for display in a [nested browsing context](#) in the protected resource created by an [object](#) or an [embed](#) element.
- Navigating such a [nested browsing context](#).

It is not required that the consumer of the element's data be a [plugin](#) in order for the `object-src` directive to be enforced. Data for any [object](#), [embed](#), or [applet](#) element MUST match the [allowed object sources](#) in order to be fetched. This is true even when the element data is semantically equivalent to content which would otherwise be restricted by one of the other [§7 Directives](#), such as an [object](#) element with a `text/html` MIME type.

Whenever the user agent would load a [plugin](#) without an associated URL (e.g., because the [object](#) element lacked a [data](#) attribute), if the protected resource's URL does not [match](#) the [allowed object sources](#) for the [protected resource](#), the user agent MUST NOT load the plugin.

7.12. plugin-types

The ***plugin-types*** directive restricts the set of plugins that can be invoked by the protected resource by limiting the types of resources that can be embedded. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "plugin-types"  
directive-value   = media-type-list
```

The term ***allowed plugin media types*** refers to the result of [parsing the plugin-types directive's value as a media type list](#).

Whenever the user agent would instantiate a [plugin](#) to handle *resource* while enforcing the plugin-types directive, the user agent MUST instead act as though the plugin reported an error *and* [report a violation](#) if any of the following conditions hold:

- The plugin is embedded into the protected resource via an [object](#) or [embed](#) element that does not explicitly declare a [MIME type](#) via a [type](#) attribute.
- *resource*'s media type does not [match](#) the list of [allowed plugin media types](#).
- The plugin is embedded into the protected resource via an [object](#) or [embed](#) element, and the media type declared in the element's [type](#) attribute is not an [ASCII case-insensitive match](#) for the *resource*'s media type.
- The plugin is embedded into the protected resource via an [applet](#) element, and *resource*'s media type is not an [ASCII case-insensitive match](#) for application/x-java-applet.

Note: In any of these cases, acting as though the plugin reported an error will cause the user agent to display the [fallback content](#).

Whenever the user agent creates a [plugin document](#) as the [active document](#) of a [child browsing context](#) of the [protected resource](#), if the user agent is enforcing any plugin-types directives for the protected resource, the user agent MUST [enforce](#) those plugin-types directives on the plugin document as well.

Whenever the user agent creates a [plugin document](#) as the [active document](#) of a [child browsing context](#) of the [protected resource](#), if the user agent is monitoring any plugin-types directives for the protected resource, the user agent MUST [monitor](#) those plugin-types directives on the plugin document as well.

7.12.1. Usage

This section is not normative.

The `plugin-types` directive whitelists a certain set of MIME types that can be embedded in a protected resource. For example, a site might want to ensure that PDF content loads, but that no other plugins can be instantiated. The following directive would satisfy that requirement:

Content-Security-Policy: [plugin-types](#) application/pdf

Resources embedded via an [embed](#) or [object](#) element delivered with an `application/pdf` content type would be rendered in the appropriate plugin; resources delivered with some other content type would be blocked. Multiple types can be specified, in any order. If the site decided to additionally allow Flash at some point in the future, it could do so with the following directive:

Content-Security-Policy: [plugin-types](#) application/pdf application/x-shockwave-flash

Note: Wildcards are not accepted in the `plugin-types` directive. Only the resource types explicitly listed in the directive will be allowed.

7.12.2. Predeclaration of expected media types

This section is not normative.

Enforcing the `plugin-types` directive requires that [object](#) and [embed](#) elements declare the expected media type of the resource they include via the [type](#) attribute. If an author expects to load a PDF, she could specify this as follows:

```
<object data="resource" type="application/pdf"></object>
```

If *resource* isn't actually a PDF file, it won't load. This prevents certain types of attacks that rely on serving content that unexpectedly invokes a plugin other than that which the author intended.

Note: *resource* will not load in this scenario even if its media type is otherwise whitelisted: resources will only load when their media type is whitelisted *and* matches the declared type in their containing element.

7.13. [report-uri](#)

The **`report-uri`** directive specifies a URL to which the user agent sends reports about policy violation. The syntax for the name and value of the directive are described by the following ABNF grammar:


```
directive-name      = "report-uri"
directive-value     = uri-reference *( 1*WSP uri-reference )
uri-reference      = <URI-reference from RFC 3986>
```

The *set of report URLs* is the value of the report-uri directive, each resolved relative to the protected resource's URL.

The process of sending violation reports to the URLs specified in this directive's value is defined in this document's [§4.4 Reporting](#) section.

Note: The report-uri directive will be ignored if contained within a [meta element](#).

7.14. sandbox

The *sandbox* directive specifies an HTML sandbox policy that the user agent applies to the protected resource. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name      = "sandbox"
directive-value     = "" / sandbox-token *( 1*WSP sandbox-token )
sandbox-token      = <token from RFC 7230>
```

When enforcing the sandbox directive, the user agent MUST [parse a sandboxing directive](#) using the directive-value as the *input* and protected resource's [forced sandboxing flag set](#) as the output. [\[HTML5\]](#)

The sandbox directive will be ignored when [monitoring](#) a policy, and when contained in a policy defined via a [meta element](#). Moreover, this directive has no effect when [monitored](#), and has no reporting requirements.

7.14.1. Sandboxing and Workers

When delivered via an HTTP header, a Content Security Policy may indicate that sandboxing flags ought to be applied to a JavaScript execution environment that is not a [Document](#). Of particular interest is the script content intended for use as a Worker, Shared Worker, or Service Worker. Many of the sandboxing flags do not apply to such environments, but [allow-scripts](#) and [allow-same-origin](#) have special requirements.

When a resource is loaded while executing the [runs a Worker](#) algorithm, the user agent MUST act as if there was a fatal network error and no resource could be obtained if either of the following conditions

holds:

1. The [sandbox](#) directive delivered with the resource does *not* contain the [allow-scripts](#) flag.
2. The [sandbox](#) directive delivered with the resource does *not* contain the [allow-same-origin](#) flag, *and* the creation of the new execution context requires it to be same-origin with its creating context.

7.14.2. Usage

This section is not normative.

HTML5 defines a [sandbox](#) attribute for [iframe](#) elements, intended to allow web authors to reduce the risk of including potentially untrusted content by imposing restrictions on that content's abilities.

When the attribute is set, the content is forced into a unique origin, prevented from submitting forms, running script, creating or navigating other browsing contexts, and prevented from running plugins. These restrictions can be loosened by setting certain flags as the attribute's value.

The `sandbox` directive allows any resource, framed or not, to ask for the same sorts of restrictions to be applied to itself.

For example, a message board or email system might provide downloads of arbitrary attachments provided by other users. Attacks that rely on tricking a client into rendering one of these attachments could be mitigated by requesting that resources only be rendered in a very restrictive sandbox.

Sending the `sandbox` directive with an empty value establishes such an environment:

Content-Security-Policy: [sandbox](#)

More trusted resources might be allowed to run in an environment with fewer restrictions by adding `allow-*` flags to the directive's value. For example, you can allow a page that you trust to run script, while ensuring that it isn't treated as same-origin with the rest of your site. This can be accomplished by sending the `sandbox` directive with the [allow-scripts](#) flag:

Content-Security-Policy: [sandbox](#) [allow-scripts](#)

The set of flags available to the CSP directive should match those available to the [iframe](#) attribute. Currently, those include:

- [allow-forms](#)
- [allow-pointer-lock](#)
- [allow-popups](#)

- [allow-same-origin](#)
- [allow-scripts](#), and
- [allow-top-navigation](#)

Note: Like the rest of Content Security Policy, the `sandbox` directive is meant as a defense-in-depth. Web authors would be well-served to use it *in addition to* standard sniffing-mitigation and privilege-reduction techniques.

7.15. `script-src`

The **`script-src`** directive restricts which scripts the protected resource can execute. The directive also controls other resources, such as XSLT style sheets [\[XSLT\]](#), which can cause the user agent to execute script. The syntax for the name and value of the directive are described by the following ABNF grammar:

```
directive-name    = "script-src"
directive-value    = source-list
```

The term ***allowed script sources*** refers to the result of [parsing the `script-src` directive's value as a source list](#) if the policy contains an explicit `script-src`, or otherwise to the [default sources](#).

If `'unsafe-inline'` is **not** in the list of [allowed script sources](#), or if at least one [nonce-source](#) or [hash-source](#) is present in the list of [allowed script sources](#):

- Whenever the user agent would execute an inline script from a [script](#) element that lacks a [valid nonce](#) and lacks a [valid hash](#) for the [allowed script sources](#), instead the user agent MUST NOT execute script, and MUST [report a violation](#).
- Whenever the user agent would execute an inline script from an inline event handler, instead the user agent MUST NOT execute script, and MUST [report a violation](#).
- Whenever the user agent would execute script contained in a javascript URL, instead the user agent MUST NOT execute the script, and MUST [report a violation](#).

If `'unsafe-eval'` is **not** in [allowed script sources](#):

- Instead of evaluating their arguments, both operator `eval` and function `eval` [\[ECMA-262\]](#) MUST throw an `EvalError` exception.
- When called as a constructor, the function `Function` [\[ECMA-262\]](#) MUST throw an `EvalError` exception.

- When called with a first argument that is not [callable](#) (a string, for example), the [setTimeout\(\)](#) function MUST return zero without creating a timer.
- When called with a first argument that is not [callable](#) (a string, for example), the [setInterval\(\)](#) function MUST return zero without creating a timer.

Whenever the user agent [fetches](#) a URL (including when following redirects) in the course of one of the following activities, if the URL does not [match](#) the [allowed script sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, and [report a violation](#):

- Requesting a script while processing the [src](#) attribute of a [script](#) element that lacks a [valid nonce](#) for the [allowed script sources](#).
- Requesting a script while invoking the `importScripts` method on a `WorkerGlobalScope` object. [\[WORKERS\]](#)
- Requesting an HTML component, such as when processing the `href` attribute of a [link](#) element with a `rel` attribute containing the token `import`. [\[HTML-IMPORTS\]](#)
- Requesting an Extensible Stylesheet Language Transformations (XSLT) [\[XSLT\]](#), such as when processing the `<?xml-stylesheet?>` processing directive in an XML document [\[XML11\]](#), the [href](#) attributes on `<xsl:include>` and `<xsl:import>` elements.

7.15.1. Nonce usage for [script](#) elements

This section is not normative.

The [script-src](#) directive lets developers specify exactly which script elements on a page were intentionally included for execution. Ideally, developers would avoid inline script entirely and whitelist scripts by URL. However, in some cases, removing inline scripts can be difficult or impossible. For those cases, developers can whitelist scripts using a randomly generated nonce.

Usage is straightforward. For *each* request, the server generates a unique value at random, and includes it in the Content-Security-Policy header:

```
Content-Security-Policy: default-src 'self';  
                        script-src 'self' https://example.com 'nonce-$RANDOM'
```

This same value is then applied as a nonce attribute to each [script](#) element that ought to be executed. For example, if the server generated the random value `Nc3n83cnSAd3wc3Sasdfn939hc3`, the server would send the following policy:

Content-Security-Policy: [default-src](#) 'self';
[script-src](#) 'self' https://example.com 'nonce-Nc3n83cnSAd3w'

Script elements can then execute either because their [src](#) URLs are whitelisted or because they have a [valid nonce](#):

```
<script>  
alert("Blocked because the policy doesn't have 'unsafe-inline'.")  
</script>
```

```
<script nonce="EDNnf03nceI0fn39fn3e9h3sdfa">  
alert("Still blocked because nonce is wrong.")  
</script>
```

```
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3">  
alert("Allowed because nonce is valid.")  
</script>
```

```
<script src="https://example.com/allowed-because-of-src.js"></script>
```

```
<script nonce="EDNnf03nceI0fn39fn3e9h3sdfa"  
  src="https://elsewhere.com/blocked-because-nonce-is-wrong.js"></script>
```

```
<script nonce="Nc3n83cnSAd3wc3Sasdfn939hc3"  
  src="https://elsewhere.com/allowed-because-nonce-is-valid.js"></script>
```

Note that the nonce's value is *not* a hash or signature that verifies the contents of the script resources. It's quite simply a random string that informs the user agent which scripts were intentionally included in the page.

Script elements with the proper nonce execute, regardless of whether they're inline or external. Script elements without the proper nonce don't execute unless their URLs are whitelisted. Even if an attacker is able to inject markup into the protected resource, the attack will be blocked by the attacker's inability to guess the random value.

7.15.2. Hash usage for [script](#) elements

This section is not normative.

The [script-src](#) directive lets developers whitelist a particular inline script by specifying its hash as an allowed source of script.

Usage is straightforward. The server computes the hash of a particular script block's contents, and includes the base64 encoding of that value in the Content-Security-Policy header:

```
Content-Security-Policy: default-src 'self';
                        script-src 'self' https://example.com 'sha256-base64 encod
```

Each inline script block's contents are hashed, and compared against the whitelisted value. If there's a match, the script is executed. For example, the SHA-256 digest of `alert('Hello, world.');` is `qznLcsR0x4GACP2dm0UCKCzCG+HiZ1guq6ZZDob/Tng=`.

You can obtain the digest of a string on the command line simply via the `openssl` program. For example:

```
echo -n "alert('Hello, world.');" | openssl dgst -sha256 -binary | openssl enc -l
```

If the server sent the following header:

```
Content-Security-Policy: script-src 'sha512-YWIZOWNiNzJjNDRlYzc4MTgwMDhmZDlkOWI0NTA;
```

Then the following script tag would result in script execution:

```
<script>alert('Hello, world.');
```

Whitespace is significant. The following scripts blocks would not hash to the same value, and would therefore *not* execute:

```
<script> alert('Hello, world.');
```

```
<script>alert('Hello, world.');
```

```
<script> alert('Hello, world.');
```

```
<script>
alert('Hello, world.');
```

Note also that the hash applies *only* to inline script. An externalized script containing the value `alert('Hello, world.');` would *not* execute if its origin was not whitelisted as a valid source of script.

7.16. `style-src`

The **`style-src`** directive restricts which styles the user may apply to the protected resource. The syntax for the name and value of the directive are described by the following ABNF grammar:

directive-name = "style-src"
directive-value = [source-list](#)

The term *allowed style sources* refers to the result of [parsing the style-src directive's value as a source list](#) if the policy contains an explicit style-src, or otherwise to the [default sources](#).

If 'unsafe-inline' is **not** in the list of [allowed style sources](#), or if at least one [nonce-source](#) or [hash-source](#) is present in the list of [allowed style sources](#):

- Whenever the user agent would apply style from a [style](#) element that lacks a [valid nonce](#) *and* lacks a [valid hash](#) for the [allowed style sources](#), instead the user agent MUST ignore the style, *and* MUST [report a violation](#).
- Whenever the user agent would apply style from a [style](#) attribute, instead the user agent MUST ignore the style, *and* MUST [report a violation](#).

Note: These restrictions on inline do not prevent the user agent from applying style from an external stylesheet (e.g., found via <link rel="stylesheet" ...>).

If 'unsafe-eval' is **not** in [allowed style sources](#), then:

- Whenever the user agent would invoke the Cascading Style Sheets Object Model algorithms [insert a CSS rule](#), [parse a CSS rule](#), [parse a CSS declaration block](#), or [parse a group of selectors](#) instead the user agent MUST throw a [SecurityError](#) exception *and* terminate the algorithm. This would include, for example, all invocations of CSSOM's various `cssText` setters and `insertRule` methods. [\[CSSOM\]](#) [\[HTML5\]](#)

Whenever the user agent [fetches](#) a URL in the course of one of the following activities, if the URL does not [match](#) the [allowed style sources](#) for the [protected resource](#), the user agent MUST act as if there was a fatal network error and no resource was obtained, *and* [report a violation](#):

- Requesting an external stylesheet when processing the [href](#) of a [link](#) element whose [rel](#) attribute contains the token [stylesheet](#).
- Requesting an external stylesheet when processing the <<@import>> directive.
- Requesting an external stylesheet when processing a Link HTTP response header field [\[RFC5988\]](#).

Note: As this stylesheet might be prefetched before a [Document](#) actually exists, user agents will need to carefully consider how to instantiate a meaningful [policy](#) against which to compare this request. See [§10.1 Processing Complications](#) for more detail.

Note: The `style-src` directive does not restrict the use of XSLT. XSLT is restricted by the [script-src](#) directive because the security consequences of including an untrusted XSLT stylesheet are similar to those incurred by including an untrusted script.

7.16.1. Nonce usage for [style](#) elements

This section is not normative.

See the [script-src nonce usage information](#) for detail; the application of nonces to [style](#) elements is similar enough to avoid repetition here.

7.16.2. Hash usage for [style](#) elements

This section is not normative.

See the [script-src hash usage information](#) for detail; the application of hashes to [style](#) elements is similar enough to avoid repetition here.

8. Examples

8.1. Sample Policy Definitions

This section provides some sample use cases and supporting [policies](#).

A server wishes to load resources only from its own origin:

Content-Security-Policy: [default-src](#) 'self'

An auction site wishes to load images from any URL, plugin content from a list of trusted media providers (including a content distribution network), and scripts only from a server under its control hosting sanitized ECMAScript:

Content-Security-Policy:

```
default-src 'self'; img-src *;  
object-src media1.example.com media2.example.com *.cdn.example.com;  
script-src trustedscripts.example.com
```

An online banking site wishes to ensure that all of the content in its pages is loaded over TLS to prevent attackers from eavesdropping on insecure content requests:

Content-Security-Policy: [default-src](#) https: 'unsafe-inline' 'unsafe-eval'

This policy allows inline content (such as inline [script](#) elements), use of `eval`, and loading resources over https. Note: This policy does not provide any protection from cross-site scripting vulnerabilities.

A website that relies on inline [script](#) elements wishes to ensure that script is only executed from its own origin, and those elements it intentionally inserted inline:

Content-Security-Policy: [script-src](#) 'self' 'nonce-*\$RANDOM*';

The inline [script](#) elements would then only execute if they contained a matching [nonce](#) attribute:

```
<script nonce="$RANDOM">...</script>
```

8.2. Sample Violation Report

This section contains an example violation report the user agent might send to a server when the protected resource violations a sample policy.

In the following example, the user agent rendered a representation of the resource `http://example.org/page.html` with the following policy:

```
default-src 'self'; report-uri http://example.org/csp-report.cgi
```

The protected resource loaded an image from `http://evil.example.com/image.png`, violating the policy.

```
{
  "csp-report": {
    "document-uri": "http://example.org/page.html",
    "referrer": "http://evil.example.com/haxor.html",
    "blocked-uri": "http://evil.example.com/image.png",
    "violated-directive": "default-src 'self'",
    "effective-directive": "img-src",
    "original-policy": "default-src 'self'; report-uri http://example.org/csp-report"
  }
}
```

9. Security Considerations

9.1. Cascading Style Sheet (CSS) Parsing

The [style-src](#) directive restricts the locations from which the protected resource can load styles. However, if the user agent uses a lax CSS parsing algorithm, an attacker might be able to trick the user agent into accepting malicious "stylesheets" hosted by an otherwise trustworthy origin.

These attacks are similar to the [CSS cross-origin data leakage](#) attack described by Chris Evans in 2009. User agents SHOULD defend against both attacks using the same mechanism: stricter CSS parsing rules for style sheets with improper MIME types.

9.2. Redirect Information Leakage

The violation reporting mechanism in this document has been designed to mitigate the risk that a malicious web site could use violation reports to probe the behavior of other servers. For example, consider a malicious web site that white lists `https://example.com` as a source of images. If the malicious site attempts to load `https://example.com/login` as an image, and the `example.com` server redirects to an identity provider (e.g., `identityprovider.example.net`), CSP will block the request. If violation reports contained the full blocked URL, the violation report might contain sensitive information contained in the redirected URL, such as session identifiers or purported identities. For this reason, the user agent includes only the origin of the blocked URL.

The mitigations are not complete, however: redirects which are blocked will produce side-effects which may be visible to JavaScript (via `img.naturalHeight`, for instance). An earlier version of this specification defined a [CSP request header](#) which servers could use (in conjunction with the `referrer` and `origin` headers) to determine whether or not it was completely safe to redirect a user. This header

caused some issues with CORS processing (tracked in [whatwg/fetch#52](#)), and has been punted to the next version of this document.

10. Implementation Considerations

The [Content-Security-Policy](#) header is an end-to-end header. It is processed and enforced at the client and, therefore, SHOULD NOT be modified or removed by proxies or other intermediaries not in the same administrative domain as the resource.

The originating administrative domain for a resource might wish to apply a [Content-Security-Policy](#) header outside of the immediate context of an application. For example, a large organization might have many resources and applications managed by different individuals or teams but all subject to a uniform organizational standard. In such situations, a [Content-Security-Policy](#) header might be added or combined with an existing one at a network-edge security gateway device or web application firewall. To enforce multiple policies, the administrator SHOULD combine the policy into a single header. An administrator might wish to use different combination algorithms depending on his or her intended semantics.

One sensible policy combination algorithm is to start by allowing a default set of sources and then letting individual upstream resource owners expand the set of allowed sources by including additional origins. In this approach, the resultant policy is the union of all allowed origins in the input policies.

Another sensible policy combination algorithm is to intersect the given policies. This approach enforces that content comes from a certain whitelist of origins, for example, preventing developers from including third-party scripts or content in violation of organizational standards and practices. In this approach, the combination algorithm forms the combined policy by removing disallowed hosts from the policies supplied by upstream resource owners.

Interactions between the [default-src](#) and other directives SHOULD be given special consideration when combining policies. If none of the policies contains a [default-src](#) directive, adding new src directives results in a more restrictive policy. However, if one or more of the input policies contain a [default-src](#) directive, adding new src directives might result in a less restrictive policy, for example, if the more specific directive contains a more permissive set of allowed origins.

Using a more restrictive policy than the input policy authored by the resource owner might prevent the resource from rendering or operating as intended.

Note: Migration to HTTPS from HTTP may require updates to the policy in order to keep things running as before. Source expressions like `http://example.com` do *not* match HTTPS resources. For example, administrators **SHOULD** carefully examine existing policies before rolling out [HTTP Strict Transport Security](#) headers for an application. [\[RFC6797\]](#)

Server administrators **MAY** wish to send multiple policies if different reporting options are desired for subsets of an overall policy. For instance, the following headers:

Content-Security-Policy: [frame-ancestors](#) https://example.com/

Content-Security-Policy: [default-src](#) https;; report-uri https://example.com/

would send violation reports for http resources, but would not send violation reports for [frame-ancestors](#) violations. Note also that combining them via `'` into the single header

Content-Security-Policy: [frame-ancestors](#) https://example.com/, [default-src](#) https

would have the same effect, as the comma splits the header during parsing.

10.1. Processing Complications

Many user agents implement some form of optimistic resource fetching algorithm to speed up page loads. In implementing these features, user agents **MUST** ensure that these optimizations do not alter the behavior of the page's security policy.

Here, we'll note a few potential complications that could cause bugs in implementations:

1. The [frame-ancestor](#) directive **MUST** take effect before a document is loaded into a [nested browsing context](#), and certainly before script is potentially executed. One way to approach this constraint is to perform the ancestor check defined in [§7.7 frame-ancestors](#) while parsing the document's headers. This might mean that no document object is available at all, which can complicate checks against `'self'`, and [scheme-](#) or [port-](#)relative source expressions.
2. Likewise, the `Link` HTTP response header could generate requests for stylesheet resources before a document is available. User agents **MUST** ensure that any policy contained in the response headers is parsed and effective *before* these requests are generated. For example, a response returning the following headers:

```
Content-Security-Policy: style-src 'none'  
Link: <awesome.css>; rel=stylesheet
```

MUST have the same behavior as a response returning the following headers:

```
Link: <awesome.css>; rel=stylesheet
Content-Security-Policy: style-src 'none'
```

namely, both must block requests for the stylesheet. To fulfil this requirement user agents **MUST** wait until all headers have been processed before beginning to prefetch resources.

11. IANA Considerations

The permanent message header field registry should be updated with the following registrations:
[\[RFC3864\]](#)

11.1. Content-Security-Policy

Header field name

Content-Security-Policy

Applicable protocol

http

Status

standard

Author/Change controller

W3C

Specification document

This specification (See [Content-Security-Policy](#) Header Field)

11.2. Content-Security-Policy-Report-Only

Header field name

Content-Security-Policy-Report-Only

Applicable protocol

http

Status

standard

Author/Change controller

W3C

Specification document

This specification (See [Content-Security-Policy-Report-Only](#) Header Field)

12. Acknowledgements

In addition to the documents in the W3C Web Application Security working group, the work on this document is also informed by the work of the [IETF websec working group](#), particularly that working group's requirements document: [draft-hodges-websec-framework-reqs](#).

A portion of the [frame-ancestors](#) directive was originally developed as X-Frame-Options. [\[RFC7034\]](#)

Brian Smith, Neil Matatall, Anne van Kesteren, and Sigbjørn Vik provided particularly insightful feedback to keep this specification sane.


Conformance

Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.


All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [\[RFC2119\]](#)

Examples in this specification are introduced with the words "for example" or are set apart from the normative text with `class="example"`, like this:



This is an example of an informative example.

Informative notes begin with the word "Note" and are set apart from the normative text with `class="note"`, like this:



Note, this is an informative note.

Conformant Algorithms

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("must", "should", "may", etc) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant. Implementers are encouraged to optimize.

Conformance Classes

A *conformant user agent* must implement all the requirements listed in this specification that are applicable to user agents.

A *conformant server* must implement all the requirements listed in this specification that are applicable to servers.

Index

Terms defined by this specification

- [allowed base URLs](#), in §7.1
- [allowed child sources](#), in §7.2
- [allowed connection targets](#), in §7.3
- [allowed font sources](#), in §7.5
- [allowed form actions](#), in §7.6
- [allowed frame ancestors](#), in §7.7
- [allowed frame sources](#), in §7.8
- [allowed image sources](#), in §7.9
- [allowed media sources](#), in §7.10
- [allowed object sources](#), in §7.11
- [allowed plugin media types](#), in §7.12
- [allowed script sources](#), in §7.15
- [allowed style sources](#), in §7.16

- [ALPHA](#), in §2.4
- [ancestor-source](#), in §7.7
- [ancestor-source-list](#), in §7.7
- [base64-value](#), in §4.2
- [base-uri](#), in §7.1
- [blockedURI](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [callable](#), in §2.3
- [callers](#), in §2.3
- [child-src](#), in §7.2
- [columnNumber](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [conformant server](#), in §Unnumbered section
- [conformant user agent](#), in §Unnumbered section
- [connect-src](#), in §7.3
- [Content-Security-Policy](#), in §3.1
- [Content-Security-Policy-Report-Only](#), in §3.2
- [Content Security Policy task source](#), in §4.4
- [default sources](#), in §7.4
- [default-src](#), in §7.4
- [digest of element's content](#), in §4.2.5
- [DIGIT](#), in §2.4
- [directive](#), in §2.1
- [directive name](#), in §2.1
- [directive-name](#), in §4.1
- [directive-token](#), in §4.1
- [directive-value](#), in §4.1

- [directive value](#), in §2.1
- [documentURI](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [effectiveDirective](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [element's content](#), in §4.2.5
- [enforce](#), in §5
- [eventInitDict](#), in §6.1
- [fire a violation event](#), in §6.3
- [font-src](#), in §7.5
- [form-action](#), in §7.6
- [frame-ancestors](#), in §7.7
- [frame-src](#), in §7.8
- [generate a violation report object](#), in §4.4
- [generating a violation report object](#), in §4.4
- [globally unique identifier](#), in §2.2
- [hash-algo](#), in §4.2
- [hash-source](#), in §4.2
- [hash-value](#), in §4.2
- [host-char](#), in §4.2
- [host-part](#), in §4.2
- [host-source](#), in §4.2
- [HTTP 200 response](#), in §2.2
- [img-src](#), in §7.9
- [JSON object](#), in §2.2
- [JSON stringification](#), in §2.2
- [keyword-source](#), in §4.2

- [lineNumber](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [match a media type list](#), in §4.3.2
- [match a source expression](#), in §4.2.2
- [match a source list](#), in §4.2.2
- [media-src](#), in §7.10
- [media type](#), in §4.3
- [media-type](#), in §4.3
- [media type list](#), in §4.3
- [media-type-list](#), in §4.3
- [monitor](#), in §5
- [nonce](#)
 - [attribute for HTMLScriptElement](#), in §4.2.3
 - [element-attr for script](#), in §4.2.3
 - [attribute for HTMLStyleElement](#), in §4.2.3
 - [element-attr for style](#), in §4.2.3
- [nonce-source](#), in §4.2
- [nonce-value](#), in §4.2
- [object-src](#), in §7.11
- [origin](#), in §2.2
- [originalPolicy](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [parse a media type list](#), in §4.3.1
- [parse a source list](#), in §4.2.1
- [parse the policy](#), in §4.1.1
- [path-part](#), in §4.2
- [plugin-types](#), in §7.12
- [policy](#), in §2.1

- [policy-token](#), in §4.1
- [port-part](#), in §4.2
- [protected resource](#), in §2.1
- [referrer](#)
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [report a violation](#), in §4.4
- [report-uri](#), in §7.13
- [representation](#), in §2.2
- [resource representation](#), in §2.2
- [runs a worker](#), in §2.3
- [sandbox](#), in §7.14
- [sandbox-token](#), in §7.14
- [scheme-part](#), in §4.2
- [scheme-source](#), in §4.2
- [script-src](#), in §7.15
- [security policy](#), in §2.1
- [security policy directive](#), in §2.1
- [security policy directive name](#), in §2.1
- [security policy directive value](#), in §2.1
- [SecurityPolicyViolationEvent](#), in §6.1
- [SecurityPolicyViolationEventInit](#), in §6.2
- [SecurityPolicyViolationEvent\(type, eventInitDict\)](#), in §6.1
- [send violation reports](#), in §4.4
- [set of report URLs](#), in §7.13
- [SHA-256](#), in §2.2
- [SHA-384](#), in §2.2
- [SHA-512](#), in §2.2
- [source-expression](#), in §4.2

- [source expression](#), in §4.2
- [source-file](#), in §4.4
- `sourceFile`
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [source-list](#), in §4.2
- [source list](#), in §4.2
- [statusCode](#), in §6.1
- [stripped for reporting](#), in §4.4
- [strip uri for reporting](#), in §4.4
- [style-src](#), in §7.16
- [type](#), in §6.1
- [uri-reference](#), in §7.13
- [URL](#), in §2.2
- [valid hash](#), in §4.2.5
- [valid nonce](#), in §4.2.4
- [VCHAR](#), in §2.4
- `violatedDirective`
 - [attribute for SecurityPolicyViolationEvent](#), in §6.1
 - [dict-member for SecurityPolicyViolationEventInit](#), in §6.2
- [WSP](#), in §2.4

Terms defined by reference

- [\[css-images-3\]](#) defines the following terms:
 - [image-set\(\)](#)
- [\[css-images-4\]](#) defines the following terms:
 - [image\(\)](#)
- [\[WebIDL\]](#) defines the following terms:
 - [DOMString](#)

References

Normative References

[ABNF]

Dave Crocker; Paul Overell. [Augmented BNF for Syntax Specifications: ABNF](#). RFC. URL: <http://www.ietf.org/rfc/rfc5234.txt>

[BEACON]

Jatinder Mann; Alois Reitbauer. [Beacon](#). WD. URL: <https://www.w3.org/TR/beacon/>

[ECMA-262]

Allen Wirfs-Brock. [ECMA-262 6th Edition, The ECMAScript 2015 Language Specification](#). June 2015. Standard. URL: <http://www.ecma-international.org/ecma-262/6.0/>

[FIPS180]

[Secure Hash Standard](#). URL: <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>

[HTML-IMPORTS]

Dmitri Glazkov; Hajime Morrita. [HTML Imports](#). WD. URL: <https://www.w3.org/TR/html-imports/>

[HTML5]

Ian Hickson; et al. [HTML5](#). REC. URL: <https://www.w3.org/TR/html5/>

[RFC3492]

Adam M. Costello. [Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications \(IDNA\)](#). REC. URL: <http://www.ietf.org/rfc/rfc3492.txt>

[RFC3864]

Graham Klyne; Mark Nottingham; Jeffrey C. Mogul. [Registration Procedures for Message Header Fields](#). RFC. URL: <http://www.ietf.org/rfc/rfc3864.txt>

[RFC4627]

Douglas Crockford. [The 'application/json' Media Type for JavaScript Object Notation \(JSON\)](#). RFC. URL: <http://www.ietf.org/rfc/rfc4627.txt>

[RFC6454]

Adam Barth. [The Web Origin Concept](#). RFC. URL: <http://www.ietf.org/rfc/rfc6454.txt>

[RFC7034]

David Ross; Tobias Gondrom. [HTTP Header Field X-Frame-Options](#). RFC. URL: <http://www.ietf.org/rfc/rfc7034.txt>

[RFC7230]

Roy T. Fielding; Julian F. Reschke. [HTTP/1.1 Message Syntax and Routing](#). RFC. URL: <http://www.ietf.org/rfc/rfc7230.txt>

[RFC7231]

Roy T. Fielding; Julian F. Reschke. [HTTP/1.1 Semantics and Content](#). RFC. URL:

<http://www.ietf.org/rfc/rfc7231.txt>

[URL]

Anne van Kesteren. [URL Standard](#). Living Standard. URL: <https://url.spec.whatwg.org/>

Note: URLs can be used in numerous different manners, in many differing contexts. For the purpose of producing strict URLs one may wish to consider [\[RFC3986\]](#) [\[RFC3987\]](#).

URLs can be used in numerous different manners, in many differing contexts. For the purpose of producing strict URLs one may wish to consider [\[RFC3986\]](#) [\[RFC3987\]](#).

As a word of caution, there are notable differences in the manner in which Web browsers and other software stacks outside the HTML context handle URLs. While no changes would be accepted to URL processing that would break existing Web content, some important parts of URL processing should therefore be considered as implementation-defined (e.g. parsing file: URLs or operating on URLs that would be syntax errors under the [\[RFC3986\]](#) [\[RFC3987\]](#) syntax).

[WebIDL]

Cameron McCormack. [Web IDL Level 1](#). 08 March 2016. CR. URL: <https://www.w3.org/TR/WebIDL-1/>

[XMLHttpRequest]

Anne van Kesteren; et al. [XMLHttpRequest Level 1](#). 30 January 2014. WD. URL: <https://www.w3.org/TR/XMLHttpRequest/>

[CSS-IMAGES-3]

CSS Image Values and Replaced Content Module Level 3 URL: <https://www.w3.org/TR/css3-images/>

[CSS-IMAGES-4]

CSS Image Values and Replaced Content Module Level 4 URL: <https://www.w3.org/TR/css4-images/>

[CSS3-FONTS]

John Daggett. [CSS Fonts Module Level 3](#). 3 October 2013. CR. URL: <https://www.w3.org/TR/css-fonts-3/>

[CSS4-IMAGES]

Elika Etemad; Tab Atkins Jr.. [CSS Image Values and Replaced Content Module Level 4](#). 11 September 2012. WD. URL: <https://www.w3.org/TR/css4-images/>

[CSSOM]

Simon Pieters; Glenn Adams. [CSS Object Model \(CSSOM\)](#). 5 December 2013. WD. URL: <https://www.w3.org/TR/cssom/>

[EVENTSOURCE]

Ian Hickson. [Server-Sent Events](https://www.w3.org/TR/eventsources/). 3 February 2015. REC. URL: <https://www.w3.org/TR/eventsources/>

[RFC2119]

S. Bradner. [Key words for use in RFCs to Indicate Requirement Levels](https://tools.ietf.org/html/rfc2119). March 1997. Best Current Practice. URL: <https://tools.ietf.org/html/rfc2119>

[RFC3986]

T. Berners-Lee; R. Fielding; L. Masinter. [Uniform Resource Identifier \(URI\): Generic Syntax](https://tools.ietf.org/html/rfc3986). January 2005. Internet Standard. URL: <https://tools.ietf.org/html/rfc3986>

[RFC5988]

M. Nottingham. [Web Linking](https://tools.ietf.org/html/rfc5988). October 2010. Proposed Standard. URL: <https://tools.ietf.org/html/rfc5988>

[WEBSOCKETS]

Ian Hickson. [The WebSocket API](https://www.w3.org/TR/websockets/). 20 September 2012. CR. URL: <https://www.w3.org/TR/websockets/>

[WORKERS]

Ian Hickson. [Web Workers](https://www.w3.org/TR/workers/). 1 May 2012. CR. URL: <https://www.w3.org/TR/workers/>

[XML11]

Tim Bray; et al. [Extensible Markup Language \(XML\) 1.1 \(Second Edition\)](https://www.w3.org/TR/xml11/). 16 August 2006. REC. URL: <https://www.w3.org/TR/xml11/>

[XSLT]

James Clark. [XSL Transformations \(XSLT\) Version 1.0](https://www.w3.org/TR/xslt). 16 November 1999. REC. URL: <https://www.w3.org/TR/xslt>

Informative References

[RFC6797]

Jeff Hodges; Collin Jackson; Adam Barth. [HTTP Strict Transport Security \(HSTS\)](http://www.ietf.org/rfc/rfc6797.txt). RFC. URL: <http://www.ietf.org/rfc/rfc6797.txt>

[UIREDRESS]

Giorgio Maone; et al. [User Interface Security Directives for Content Security Policy](https://www.w3.org/TR/UISecurity/). WD. URL: <https://www.w3.org/TR/UISecurity/>

IDL Index

```
partial interface HTMLScriptElement {  
  attribute DOMString nonce;  
};
```

```
partial interface HTMLStyleElement {  
  attribute DOMString nonce;  
};
```

```
[Constructor(DOMString type, optional SecurityPolicyViolationEventInit eventInitDict);  
interface SecurityPolicyViolationEvent : Event {  
  readonly attribute DOMString documentURI;  
  readonly attribute DOMString referrer;  
  readonly attribute DOMString blockedURI;  
  readonly attribute DOMString violatedDirective;  
  readonly attribute DOMString effectiveDirective;  
  readonly attribute DOMString originalPolicy;  
  readonly attribute DOMString sourceFile;  
  readonly attribute DOMString statusCode;  
  readonly attribute long lineNumber;  
  readonly attribute long columnNumber;  
};
```

```
dictionary SecurityPolicyViolationEventInit : EventInit {  
  DOMString documentURI;  
  DOMString referrer;  
  DOMString blockedURI;  
  DOMString violatedDirective;  
  DOMString effectiveDirective;  
  DOMString originalPolicy;  
  DOMString sourceFile;  
  long lineNumber;  
  long columnNumber;  
};
```

