Content Security Policy 1.0

W3C Candidate Recommendation 15 November 2012

This version:
http://www.w3.org/TR/2012/CR-CSP-20121115/
Latest published version:
http://www.w3.org/TR/CSP/
Previous version:
http://www.w3.org/TR/2012/WD-CSP-20120710/
Latest editor's draft:
http://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-1.0-specification.html
Editors:
Brandon Sterne, Invited Expert (formerly of Mozilla Corporation)
Adam Barth, Google, Inc.

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 http://www.w3.org/TR/.

This document describes a proposal that has been discussed by the broader community since 2010 There are experimental implementations in Firefox and Chrome, using the header names `X-Content-Security-Policy` and `X-WebKit-CSP` respectively. Internet Explorer 10 Platform Preview also contains a partial implementation, using the header name `X-Content-Security-Policy`.

In addition to the documents in the W3C Web Application Security working group, the

This version is outdated!
For the latest version, please look at https://www.w3.org/TR/CSP1/.

▲ expand

framework-reqs.

This document was published by the Web Application Security Working Group as a Candidate Recommendation. This document is intended to become a W3C Recommendation. If you wish to make comments regarding this document, please send them to public-webappsec@w3.org (subscribe, archives). W3C publishes a Candidate Recommendation to indicate that the document is believed to be stable and to encourage implementation by the developer community. This Candidate Recommendation is expected to advance to Proposed Recommendation no earlier than 30 November 2012. All feedback is welcome. A diff-marked version against the previous version of this document is available.

Publication as a Candidate Recommendation does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

The entrance criteria for this document to enter the Proposed Recommendation stage is to have a minimum of two independent and interoperable user agents that implementation all the features of this specification, which will be determined by passing the user agent tests defined in the test suite developed by the Working Group.

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.

Table of Contents

1. Introduction

This section is non-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 from where the application expects to load resources.

To mitigate XSS, for example, a web application can declare from where it expects to load scripts, allowing 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.

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 the current resource representation only. To supply a policy for an entire site, the server needs to supply a policy with each resource representation.

2. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this

specification is normative.

The key words MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this specification are to be interpreted as described in [RFC2119].

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.

A ***conformant user agent*** MUST implement all the requirements listed in this specification that are applicable to user-agents, and MAY implement those marked as "(Optional)".

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

2.1 Key Concepts and Terminology

This section defines several terms used throughout the document.

The term ***security policy***, or simply ***policy***, for the purposes of this specification refers to either:

1. a set of security preferences for restrictions within which the content can operate, or
2. a fragment of text that codifies these preferences.

The security policies defined by this document are applied by a user agent on a *per-resource representation basis*. Specifically, when a user agent receives a policy along with the representation of a given resource, that policy applies to *that resource representation only*. This document often refers to that resource representation as the ***protected resource***.

A server transmits its security policy for a particular protected resource as a collection of ***directives***, such as `default-src 'self'`, each of which declares a specific set of restrictions for that resource as instantiated by the user agent. More details are provided in the directives section.

A directive consists of a ***directive name***, which indicates the privileges controlled by the directive, and a ***directive value***, which specifies the restrictions the policy imposes on those privileges.

The term ***origin*** is defined in the Origin specification. [RFC6454]

The term ***URI*** is defined in the URI specification. [URI]

The term ***resource representation*** is defined in the HTTP 1.1 specification. [HTTP11]

The `<script>`, `<object>`, `<embed>`, `<img>`, `<video>`, `<audio>`, `<source>`, `<track>`, `<link>`, `<applet>`, `<frame>` and `<iframe>` elements 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. [CSS3FONT]

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 Augmented Backus-Naur Form (ABNF) notation used in this document is specified in RFC 5234. [ABNF]

This document also uses the ABNF extension "#rule" as defined in HTTP 1.1. [HTTP11]

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

3. Framework

This section defines the general framework for content security policies, including the delivery mechanisms and general syntax for policies. The next section contains the details of the specific directives introduced in this specification.

3.1 Policy Delivery

The server delivers the policy to the user agent via an HTTP response header.

3.1.1 `Content-Security-Policy` Header Field

The `Content-Security-Policy` header field is the preferred mechanism for delivering a CSP policy.

```
"Content-Security-Policy:" 1#policy
```

A server MAY 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.

3.1.2 `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.

```
"Content-Security-Policy-Report-Only:" 1#policy
```

For example, a 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. If their site violates this policy, instead of breaking the site, the user agent will send violation reports to a URI specified in the policy. Once a site has confidence that the policy is appropriate, they start enforcing the policy using the `Content-Security-Policy` header field.

A server MAY 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.

3.2 Syntax and Algorithms

3.2.1 Policies

A CSP ***policy*** consists of a U+003B SEMICOLON (`;`) delimited list of directives:

```
policy            = [ directive *( ";" [ directive ] ) ]
```

Each ***directive*** consists of a *directive-name* and (optionally) a *directive-value*:

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

3.2.1.1 Parsing

To ***parse a CSP 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 with name *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*.

3.2.2 Source List

Many CSP directives use a value consisting of a ***source list***.

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 `'self'` represents the set of URIs which are in the same origin as the protected resource 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
scheme-source     = scheme ":"
host-source       = [ scheme "://" ] host [ port ]
ext-host-source   = host-source "/" *( <VCHAR except ";" and ","> )
                  ; ext-host-source is reserved for future use.
keyword-source    = "'self'" / "'unsafe-inline'" / "'unsafe-eval'"
scheme            = <scheme production from RFC 3986>
host              = "*" / [ "*." ] 1*host-char *( "." 1*host-char )
host-char         = ALPHA / DIGIT / "-"
port              = ":" ( 1*DIGIT / "*" )
```

3.2.2.1 Parsing

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

1. If *source list* (with leading and trailing whitespace stripped) is a case insensitive match for the string `'none'` (including the quotation marks), return the empty set.
2. Let the *set of source expressions* be the empty set.
3. For each token returned by splitting *source list* on spaces, if the token matches the grammar for `source-expression` or `ext-host-source`, add the token to the *set of source expressions*.
4. Return the *set of source expressions*.

3.2.2.2 Matching

To check whether a URI ***matches a source expression***, the user agent MUST use an algorithm equivalent to the following:

1. If the source expression a consists of a single U+002A ASTERISK character (`*`), then return *does match*.
2. If the source expression matches the grammar for `scheme-source`:
    1. If the URI's scheme is a case-insensitive match for the source expression's `scheme`, return *does match*.
    2. Otherwise, return *does not match*.
3. If the source expression matches the grammar for `host-source` or `ext-host-source`:
    1. If the URI does not contain a host, then return *does not match*.
    2. Let *uri-scheme*, *uri-host*, and *uri-port* be the scheme, host, and port of the URI, respectively. If the URI does not have a port, then let *uri-port* be the default port for *uri-scheme*.

3. If the source expression has a `scheme` that is not a case insensitive match for *uri-scheme*, then return *does not match*.
4. If the source expression does **not** have a `scheme` and if *uri-scheme* is not a case insensitive match for the scheme of the protected resource's URI, then return *does not match*.
5. If the first character of the source expression's `host` 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 *uri-host*, then return *does not match*.
6. If *uri-host* is not a case insensitive match for the source expression's `host`, then return *does not match*.
7. If the source expression does **not** contain a `port` and *uri-port* is not the default port for *uri-scheme*, then return *does not match*.
8. If the source expression does contain a `port` that (a) does **not** contain an U+002A ASTERISK character (`*`) and (b) does **not** represent the same number as *uri-port*, then return *does not match*.
9. Otherwise, return *does match*.

4. If the source expression is a case insensitive match for `'self'` (including the quotation marks), then return *does match* if the URI has the same scheme, host, and port as the protected resource's URI (using the default port for the appropriate scheme if either or both URIs are missing ports).
5. Otherwise, return *does not match*.

A URI ***matches a source list***, if, and only if, the URI matches at least one source expression in the set of source expressions obtained by parsing the source list. Notice that no URIs match an empty set of source expressions, such as the set obtained by parsing the source list `'none'`.

3.3 Processing Model

To ***enforce*** a CSP 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 Directives, below).

Generally speaking, enforcing a directive prevents the protected resource from performing certain actions, such as loading scripts from URIs 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.

Enforcing a CSP policy SHOULD NOT interfere with the operation of user-supplied scripts such as third-party user-agent add-ons and JavaScript bookmarklets.

To ***monitor*** a CSP 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 instead reported to the developer of the web application. Monitoring a CSP 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 is using one policy but wishes to experiment with a stricter policy, the server operator 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 CSP 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 CSP 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.

Whenever a user agent runs a worker: [WEBWORKERS]

- If the user agent is enforcing a CSP policy for the *owner document*, the user agent MUST enforce the CSP policy for the worker.
- If the user agent is monitoring a CSP policy for the *owner document*, the user agent MUST monitor the CSP policy for the worker.

4. Directives

This section describes the content security policy directives introduced in this specification.

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 `'unsafe-inline'` in their CSP policies if they wish to protect themselves against XSS.

4.1 `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 directive's value as a source list.

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

- script-src
- object-src
- style-src

- img-src
- media-src
- frame-src
- font-src
- connect-src

If not specified explicitly in the policy, the directives listed above will use the *default sources*.

4.2 `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
```

If the policy contains an explicit `script-src`, let the *allowed script sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed script sources* be the *default sources*

If `'unsafe-inline'` is **not** in *allowed script sources*:

- Whenever the user agent would execute an inline script (either from a `script` element or from an inline event handler), instead the user agent MUST NOT execute script.
- Whenever the user agent would execute script contained in a `javascript` URI, instead the user agent MUST NOT execute the script. (The user agent SHOULD execute script contained in "bookmarklets" even when enforcing this restriction.)

If `'unsafe-eval'` is **not** in *allowed script sources*:

- Instead of evaluating their arguments, both operator `eval` and function `eval` MUST throw a security exception. [ECMA-262]
- When called as a constructor, the function `Function` MUST throw a security exception. [ECMA-262]
- When called with a first argument that is non-callable (e.g., not a function), the `setTimeout` function MUST return zero without creating a timer.
- When called with a first argument that is non-callable (e.g., not a function), the `setInterval` function MUST return zero without creating a timer.

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

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed script sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Requesting a script, such as when processing the `src` attribute of a `script` element or when processing the `Worker` or `SharedWorker` constructors.

- 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>` element, or the `href` attributes on `<xsl:import>` element.

4.3 `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
```

If the policy contains an explicit `object-src`, let the *allowed object sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed object sources* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed object sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Requesting data for a plugin, such as when processing the `data` attribute of an `object` element, the `src` attribute of an `embed` elements, 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 directives, such as an `object` element with a `text/html` MIME type.

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

4.4 `style-src`

The `style-src` directive restricts which styles the user applies 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
```

If the policy contains an explicit `style-src`, let the *allowed style sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed style sources* be the *default sources*

If `'unsafe-inline'` is **not** in *allowed style sources*:

- Whenever the user agent would apply style from a `style` element, instead the user agent `MUST` ignore the style.
- Whenever the user agent would apply style from a `style` attribute, instead the user agent `MUST` ignore the style.

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">`). The user agent is also not prevented from applying style from Cascading Style Sheets Object Model (CSSOM). [CSSOM]

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed style sources*, the user agent **MUST** act as if it had received an empty HTTP 400 response:

- Requesting external style sheets, such as when processing the `href` attribute of a `link` element with a `rel` attribute containing the token `stylesheet` or when processing the `@import` directive in a stylesheet.

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.

4.5 `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
```

If the policy contains an explicit `img-src`, let the *allowed image sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed image sources* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed image sources*, the user agent **MUST** act as if it had received an empty HTTP 400 response:

- Requesting data for an image, such as when processing the `src` attribute of an `img` elements, the `url()` or `image()` values on any Cascading Style Sheets (CSS) property that is capable of loading an image [*CSS3-Images*], or the `href` attribute of a `link` element with an image-related `rel` attribute, such as `icon`.

4.6 `media-src`

The `media-src` directive restricts from where the protected resource can load video and audio. 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
```

If the policy contains an explicit `media-src`, let the *allowed media sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed media sources* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed media sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Requesting data for a video or audio clip, such as when processing the `src` attribute of a `video`, `audio`, `source`, or `track` elements.

4.7 `frame-src`

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
```

If the policy contains an explicit `frame-src`, let the *allowed frame sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed frame sources* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed frame sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Requesting data for display in a nested browsing context in the protected resource created by an `iframe` or a `frame` element.
- Navigating such a nested browsing context.

4.8 `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
```

If the policy contains an explicit `font-src`, let the *allowed font sources* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed font sources* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed font sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Requesting data for display in a font, such as when processing the `@font-face`

Cascading Style Sheets (CSS) rule.

4.9 connect-src

The connect-src directive restricts which URIs 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
```

If the policy contains an explicit connect-src, let the *allowed connection targets* be the result of parsing the directive's value as a source list. Otherwise, let the *allowed connection targets* be the *default sources*

Whenever the user agent fetches a URI (including when following redirects) in the course of one of the following activities, if the URI does not match the *allowed font sources*, the user agent MUST act as if it had received an empty HTTP 400 response:

- Processing the open() method of an XMLHttpRequest object.
- Processing the WebSocket constructor.
- Processing the EventSource constructor.

4.10 sandbox (Optional)

The sandbox directive is optional.

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   = token *( 1*WSP token )
token             = <token from RFC 2616>
```

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

4.11 report-uri

The report-uri directive specifies a URI 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>
```

Let the *set of report URIs* be the value of the report-uri directive, each resolved relative to the protected resource's URI.

To ***send a violation report***, the user agent MUST use an algorithm equivalent to the

following:

1. Prepare a JSON object *violation-object* with the following keys and values: [RFC4627]
   csp-report
   A JSON object containing the following keys and values:
   document-uri
   The address of the protected resource, with any <fragment> component removed
   referrer
   The referrer attribute of the protected resource
   blocked-uri
   URI of the resource that was prevented from loading due to the policy violation, with any <fragment> component removed
   violated-directive
   The policy directive that was violated
   original-policy
   The original policy as received by the user-agent.
2. If the origin of the blocked-uri is not the same as the origin of the protected resource, then replace the blocked-uri with the ASCII serialization of the blocked-uri's origin.
3. Let the *violation report* be the JSON stringification of the *violation-object*.
4. For each *report URI* in the *set of report URIs*:
   1. Fetch the *report URI* from origin of the protected resource, with the synchronous flag *not* set, using HTTP method `POST`, with a `Content-Type` header field of `application/json` with an entity body consisting of the *violation report*. The user agent MUST NOT follow redirects when fetching this resource. (Note: The user agent ignores the fetched resource.)

5. Examples

5.1 Sample Policy Definitions

This section is non-normative.

This section provides some sample use cases and accompanying security policies.

Example 1: A server wishes to load resources only form its own origin:

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

Example 2: An auction site wishes to load images from any URI, 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.
                         script-src trustedscripts.example.com
```

Example 3: 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.

Example 4: A social network wishes to ensure that all scripts are loaded from a specific path to prevent user-generated content from being interpreted as script:

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

Unfortunately, this use case is not supported in CSP 1.0. The user agent will ignore the path and act as if the policy contained a `script-src` directive with value `https://example.com`. A future version of CSP might begin enforcing these path restrictions, however.

5.2 Sample Violation Report

This section is non-normative.

This section contains an example violation report the user agent might sent 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 CSP 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'",
    "original-policy": "default-src 'self'; report-uri http://example.org/csp-re
  }
}
```

6. Security Considerations

6.1 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 "style sheets" 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.

6.2 Violation Reports

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., `idenityprovider.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 URI, such as session identifiers or purported identities. For this reason, the user agent includes only the origin of the blocked URI.

7. 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.

8. IANA Considerations

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

8.1 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)

8.2 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)

A. References

A.1 Normative references

[ABNF]
D. Crocker and P. Overell. *Augmented BNF for Syntax Specifications: ABNF.* January 2008. Internet RFC 5234. URL: http://www.ietf.org/rfc/rfc5234.txt

[CSS3FONT]
Michel Suignard; Chris Lilley. *CSS3 module: Fonts.* 2 August 2002. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2002/WD-css3-fonts-20020802

[CSSOM]
Glenn Adams; Shane Stephens. *CSS Object Model (CSSOM).* 14 March 2012. W3C Working Draft. (Work in progress.) URL: http://dev.w3.org/csswg/cssom/

[ECMA-262]
ECMAScript Language Specification. June 2011. URL: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[EVENTSOURCE]
Ian Hickson. *Server-Sent Events.* 26 April 2012. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/eventsource/

[HTML5]

Ian Hickson; David Hyatt. *HTML5.* 29 March 2012. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/html5

[HTTP11]
R. Fielding; et al. *Hypertext Transfer Protocol - HTTP/1.1.* June 1999. Internet RFC 2616. URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC2119]
S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC4627]
D. Crockford. *The application/json Media Type for JavaScript Object Notation (JSON)* July 2006. Internet RFC 4627. URL: http://www.ietf.org/rfc/rfc4627.txt

[RFC6454]
A. Barth. *The Web Origin Concept.* December 2011. Internet RFC 6454. URL: http://www.rfc-editor.org/rfc/rfc6454.txt

[URI]
T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifiers (URI): generic syntax.* January 2005. Internet RFC 3986. URL: http://www.ietf.org/rfc/rfc3986.txt

[WEBIDL]
Cameron McCormack. *Web IDL.* 27 September 2011. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2011/WD-WebIDL-20110927/

[WEBSOCKETS]
Ian Hickson. *The WebSocket API.* 24 May 2012. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/websockets/

[WEBWORKERS]
Ian Hickson. *Web Workers.* 1 September 2011. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2011/WD-workers-20110901/

[XML11]
Eve Maler; et al. *Extensible Markup Language (XML) 1.1 (Second Edition).* 16 August 2006. W3C Recommendation. URL: http://www.w3.org/TR/2006/REC-xml11-20060816

[XMLHTTPREQUEST]
Anne van Kesteren. *The XMLHttpRequest Object.* 15 April 2008. W3C Working Draft. (Work in progress.) URL: http://www.w3.org/TR/2008/WD-XMLHttpRequest-20080415

[XSLT]
James Clark. *XSL Transformations (XSLT) Version 1.0.* 16 November 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-xslt-19991116