Package 'firesafety'

May 8, 2026

Title A Collection of Security Related Plugins for 'fiery'

Version 0.1.0

Description Provide a range of plugins for 'fiery' web servers that handle different aspects of server-side web security. Be aware that security cannot be handled blindly, and even though these plugins will raise the security of your server you should not build critical infrastructure without the aid of a security expert.

License MIT + file LICENSE

Encoding UTF-8

URL https://github.com/thomasp85/firesafety

BugReports https://github.com/thomasp85/firesafety/issues

Imports cli, R6, rlang (>= 1.1.0), routr (>= 1.0.0)

RoxygenNote 7.3.2

Suggests fiery (>= 1.3.0), testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Thomas Lin Pedersen [aut, cre] (ORCID: <https://orcid.org/0000-0002-5147-4711>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Thomas Lin Pedersen <thomas.pedersen@posit.co>

Repository CRAN

Date/Publication 2025-09-10 08:20:02 UTC

Contents

CORS *Plugin for setting up CORS in a fiery server*

Description

Cross-Origin Resource Sharing (CORS) is a mechanism for servers to indicate from where it may be accessed and allows browsers to block requests that are not permitted. For security reasons, browsers limits requests initiated from JavaScript to only those for the same site. To allow requests from other sites the server needs to send the right CORS headers with the response. Read more about CORS at MDN

Details

CORS is opt-in. The security measure is already in place in browsers to limit cross-origin requests, and CORS is a way to break out of this in a controlled manner where you can indicate exactly who can make a request and what requests can be made. In general it works like this:

1. A request is being initiated from a website, either through JavaScript or another venue, to a site different than the one it originates from.
2. The browser identifies that the request is cross-origin and sends an `OPTIONS` request to the server with information about the request it intends to send (this is called a pre-flight request).
3. The server responds with a 204 response giving the allowed types of requests that can be made for the resource.
4. If the original request conforms to the response the browser will then send the actual request.
5. The server responds to the actual request.
6. The client gets the response, but the browser will limit what information in the response it can access based on the information provided by the server in the pre-flight response.

As can be seen, a CORS request is slightly more complex than the standard request-response you normally think about. However, the pre-flight request can be cached by the browser and so, will not happen every time a ressource is accessed. While a site may employ a CORS policy the same way across all its endpoints it does not need to. It is fine to only turn on CORS for a subset of paths. In general it is a good rule of thumb to set up resource isolation for the paths that do not have CORS enabled.

Initialization

A new 'CORS'-object is initialized using the `new()` method on the generator and pass in any settings deviating from the defaults

Usage

```
cors <- CORS$new(...)
```

Fiery plugin

A CORS object is a fiery plugin and can be used by passing it to the `attach()` method of the fiery server object. Once attached all requests will be passed through the plugin and the policy applied to it

Active bindings

`name` The name of the plugin

Methods

Public methods:

- `CORS$new()`
- `CORS$add_path()`
- `CORS$on_attach()`
- `CORS$clone()`

Method `new()`**:** Initialize a CORS object

Usage:

```
CORS$new(
  path = "/*",
  origin = "*",
  methods = c("get", "head", "put", "patch", "post", "delete"),
  allowed_headers = NULL,
  exposed_headers = NULL,
  allow_credentials = FALSE,
  max_age = NULL
)
```

Arguments:

`path` The path that the policy should apply to. routr path syntax applies, meaning that wilcards and path parameters are allowed.

`origin` The origin allowed for the path. Can be one of:

- A boolean. If `TRUE` then all origins are permitted and the preflight response will have the `Access-Control-Allow-Origin` header reflect the origin of the request. If `FALSE` then all origins are denied
- The string `"*"` which will allow all origins and set `Access-Control-Allow-Origin` to `*`. This is different than setting it to `TRUE` because `*` instructs browsers that any origin is allowed and it may use this information when searching the cache
- A character vector giving allowed origins. If the request origin matches any of these then the `Access-Control-Allow-Origin` header in the response will reflect the origin of the request
- A function taking the request and returning `TRUE` if the origin is permitted and `FALSE` if it is not. If permitted the `Access-Control-Allow-Origin` header will reflect the request origin

`methods` The HTTP methods allowed for the `path`

`allowed_headers` A character vector of request headers allowed when making the request. If the request contains headers not permitted, then the response will be blocked by the browser. `NULL` will allow any header by reflecting the `Access-Control-Request-Headers` header value from the request into the `Access-Control-Allow-Headers` header in the response.

`exposed_headers` A character vector of response headers that should be made available to the client upon a succesful request

`allow_credentials` A boolean indicating whether credentials are allowed in the request. Credentials are cookies or HTTP authentication headers, which are normally stripped from `fetch()` requests by the browser. If this is `TRUE` then `origin` cannot be `*` according to the spec

`max_age` The duration browsers are allowed to keep the preflight response in the cache

Method `add_path()`**:** Add CORS settings to a path

Usage:

```
CORS$add_path(
  path = "/*",
  origin = "*",
  methods = c("get", "head", "put", "patch", "post", "delete"),
  allowed_headers = NULL,
  exposed_headers = NULL,
  allow_credentials = FALSE,
  max_age = NULL
)
```

Arguments:

`path` The path that the policy should apply to. routr path syntax applies, meaning that wilcards and path parameters are allowed.

`origin` The origin allowed for the path. Can be one of:

- A boolean. If `TRUE` then all origins are permitted and the preflight response will have the `Access-Control-Allow-Origin` header reflect the origin of the request. If `FALSE` then all origins are denied
- The string `"*"` which will allow all origins and set `Access-Control-Allow-Origin` to `*`. This is different than setting it to `TRUE` because `*` instructs browsers that any origin is allowed and it may use this information when searching the cache
- A character vector giving allowed origins. If the request origin matches any of these then the `Access-Control-Allow-Origin` header in the response will reflect the origin of the request
- A function taking the request and returning `TRUE` if the origin is permitted and `FALSE` if it is not. If permitted the `Access-Control-Allow-Origin` header will reflect the request origin

`methods` The HTTP methods allowed for the `path`

`allowed_headers` A character vector of request headers allowed when making the request. If the request contains headers not permitted, then the response will be blocked by the browser. `NULL` will allow any header by reflecting the `Access-Control-Request-Headers` header value from the request into the `Access-Control-Allow-Headers` header in the response.

`exposed_headers` A character vector of response headers that should be made available to the client upon a succesful request

`allow_credentials` A boolean indicating whether credentials are allowed in the request. Credentials are cookies or HTTP authentication headers, which are normally stripped from `fetch()` requests by the browser. If this is `TRUE` then `origin` cannot be `*` according to the spec

`max_age` The duration browsers are allowed to keep the preflight response in the cache

Method `on_attach()`**:** Method for use by `fiery` when attached as a plugin. Should not be called directly.

Usage:

```
CORS$on_attach(app, ...)
```

Arguments:

`app` The fiery server object

`...` Ignored

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CORS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Setup CORS for a sub path allowing access from www.trustworthy.com
# Tell the browser to cache the preflight for a day
cors <- CORS$new(
  path = "/shared_assets/*",
  origin = "https://www.trustworthy.com",
  methods = c("get", "head", "post"),
  max_age = 86400
)


# Use it in a fiery server
app <- fiery::Fire$new()

app$attach(cors)
```

---

csp *Construct settings for the* `Content-Security-Policy` *header*

Description

This helper function exists mainly to document the possible values and prevent misspelled directives. It returns a bare list. See the header reference and the CSP section of the MDN security guide for more information on the header

Usage

```
csp(
  default_src = NULL,
  script_src = NULL,
  script_src_elem = NULL,
  script_src_attr = NULL,
  style_src = NULL,
  style_src_elem = NULL,
  style_src_attr = NULL,
  img_src = NULL,
  font_src = NULL,
  media_src = NULL,
  object_src = NULL,
  child_src = NULL,
  frame_src = NULL,
  worker_src = NULL,
  connect_src = NULL,
  fenced_frame_src = NULL,
  manifest_src = NULL,
  prefetch_src = NULL,
  base_uri = NULL,
  sandbox = FALSE,
  form_action = NULL,
  frame_ancestors = NULL,
  report_to = NULL,
  require_trusted_types_for = FALSE,
  trusted_types = NULL,
  upgrade_insecure_requests = FALSE
)
```

Arguments

`default_src`	Fallback for all other `*_src` values
`script_src`	Fallback for `script_src_*` values
`script_src_elem`	Valid sources for `<script>` elements
`script_src_attr`	Valid sources for inline event handlers
`style_src`	Fallback for `style_src_*` values
`style_src_elem`	Valid sources for `<style>` elements
`style_src_attr`	Valid sources for inline styling of elements
`img_src`	Valid sources for images and favicons
`font_src`	Valid sources for fonts loaded with `@font-face`
`media_src`	Valid sources for `<audio>`, `<video>`, and `<track>` elements
`object_src`	Valid sources for `<object>` and `<embed>` elements

child_src	Fallback for frame_src and worker_src
frame_src	Valid sources for <frame> and <iframe> elements
worker_src	Valid sources for Worker, SharedWorker, and ServiceWorker scripts
connect_src	Valid sources for URLs loaded from within scripts
fenced_frame_src	Valid sources for <fencedframe> elements
manifest_src	Valid sources for application manifest files
prefetch_src	Valid sources to be prefetched and prerendered
base_uri	Valid sources that can be put in a <base> element
sandbox	Logical. Enable sandboxing of the requested document/ressource
form_action	Valid URLs to be targeted by form submissions
frame_ancestors	Valid parents that may embed this document in an <frame>, <iframe>, <object>, or <embed> element.
report_to	A URL to report violations to. Setting this will also add a report-uri directive along with a Reporting-Endpoints header for maximum compitability.
require_trusted_types_for	Logical. Enforces Trusted Types
trusted_types	Specifies an allow list of Trusted Types
upgrade_insecure_requests	Logical. Automatically treat all HTTP urls in the document as if they were HTTPS

Value

A bare list with the input arguments

Examples

```
# Default setting
csp(
  default_src = "self",
  script_src = "self",
  script_src_attr = "none",
  style_src = c("self", "https:", "unsafe-inline"),
  img_src = c("self", "data:"),
  font_src = c("self", "https:", "data:"),
  object_src = "none",
  base_uri = "self",
  form_action = "self",
  frame_ancestors = "self",
  upgrade_insecure_requests = TRUE
)
```

`ResourceIsolation` *Fetch metadata based resource isolation plugin*

Description

This plugin uses the information provided in the `Sec-Fetch-*` request headers to block unwanted requests to your server coming from other sites. Setting up a strict control with which requests are allowed is an important part of preventing some cross-site leaks as well as cross-site request forgery attacks.

Details

Compared to the other security measures in firesafety, the reource isolation plugin is a server-side blocker of requests. Both CORS and CORP sends back a full response and it is then up to the browser to determine if the response becomes available to the site. In contrast, this plugin will return a 403 response if the request fails to be accepted. This is not to say that resource isolation is *better* than CORS, CORP or other measures. They all target different situations (or the same situation from different angles) and works best in unison. You can read more about this type of defence at MDN and XS-Leaks Wiki

How it works:

Resource isolation takes advantage of the `Sec-Fetch-*` headers that browser send along with requests. These headers informs the server about the nature of the request. Where it comes from, what action initiated it, and how it will be used. Based on this information the server may chose to allow a request to proceed or deny it altogether. This plugin runs a request through a range of tests and if it passes *any* of them it proceeds:

1. Does the request have the `Sec-Fetch-*` headers
2. Is `allow_cors == TRUE` and is `Sec-Fetch-Mode` set to `cors`
3. Is `Sec-Fetch-Site` set to `allowed_site` or a more restrictive value
4. Is the request method `GET`, the `Sec-Fetch-Mode` `navigation`, and the `Sec-Fetch-Dest` not one of those given by `forbidden_navigation`

You can have different permissions for different paths. The default during initialization is to add it to `/*` so that all all paths will share the same policy, but you can strengthen or loosen up specific paths as needed. A good rule of thumb is to make the policy as restrictive as possible while allowing your application to still work as intented. Further, if you have paths that do not have a resource isolation policy in place these should have CORS enabled.

Initialization

A new 'ResourceIsolation'-object is initialized using the `new()` method on the generator and pass in any settings deviating from the defaults

Usage

```
resource_isolation <- ResourceIsolation$new(...)
```

Fiery plugin

A ResourceIsolation object is a fiery plugin and can be used by passing it to the `attach()` method of the fiery server object. Once attached all requests will be passed through the plugin and the policy applied to it

Active bindings

`name` The name of the plugin

Methods

Public methods:

- `ResourceIsolation$new()`
- `ResourceIsolation$add_path()`
- `ResourceIsolation$on_attach()`
- `ResourceIsolation$clone()`

Method `new()`**:** Initialize a new ResourceIsolation object

Usage:

```
ResourceIsolation$new(
  path = "/*",
  allowed_site = "same-site",
  forbidden_navigation = c("object", "embed"),
  allow_cors = TRUE
)
```

Arguments:

`path` The path that the policy should apply to. routr path syntax applies, meaning that wilcards and path parameters are allowed.

`allowed_site` The allowance level to permit. Either `cross-site`, `same-site`, or `same-origin`.

`forbidden_navigation` A vector of destinations not allowed for navigational requests. See the `Sec-Fetch-Dest` documentation for a description of possible values. The special value `"all"` is also permitted which is the equivalent of passing all values.

`allow_cors` Should `Sec-Fetch-Mode: cors` requests be allowed

Method `add_path()`**:** Add a policy to a path

Usage:

```
ResourceIsolation$add_path(
  path,
  allowed_site,
  forbidden_navigation = c("object", "embed"),
  allow_cors = TRUE
)
```

Arguments:

`path` The path that the policy should apply to. routr path syntax applies, meaning that wilcards and path parameters are allowed.

`allowed_site` The allowance level to permit. Either `cross-site`, `same-site`, or `same-origin`.

`forbidden_navigation` A vector of destinations not allowed for navigational requests. See the `Sec-Fetch-Dest` documentation for a description of possible values. The special value `"all"` is also permitted which is the equivalent of passing all values.

`allow_cors` Should `Sec-Fetch-Mode: cors` requests be allowed

Method `on_attach()`**:** Method for use by `fiery` when attached as a plugin. Should not be called directly.

Usage:

```
ResourceIsolation$on_attach(app, ...)
```

Arguments:

`app` The fiery server object

`...` Ignored

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ResourceIsolation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create resource isolation policy denying all navigation requests
resource_isolation <- ResourceIsolation$new(forbidden_navigation = "all")

# Allow cross-site requests on a subpath
resource_isolation$add_path(
  path = "/all_is_welcome/*",
  allowed_site = "cross-site"
)


# Use it in a fiery server
app <- fiery::Fire$new()

app$attach(resource_isolation)
```

SecurityHeaders *Plugin for setting security related headers*

Description

This plugin is inspired by Helmet.js and aids you in setting response headers relevant for security of your fiery server. All defaults are taken from Helmet.js as well, except for the `max-age` of the `Strict-Transport-Security` header that has been doubled to 2 years which is the recommendation.

Details

Web security is a complicated subject and it is impossible for this document to stay current and true at all times as well as be able to learn the user of all the intricacies of web security. It is **strongly** advised that you familiarise yourself with this subject if you plan on exposing a fiery webserver to the public. A good starting point is MDN's guide on web security.

This plugin concerns 14 different headers that are in one way or another implicated in security. Some of them are only relevant if you serve HTML content on the web and have no effect on e.g. a server providing a REST api. These have been marked with **UI** below. While you may turn these off for a pure API server (by setting them to `NULL`), it is advised that you only steer away from the defaults if you have a good grasp of the implications. The headers are set very efficiently so removing some unneeded ones will only have an effect on the size of the response, not the handling time.

Headers:

`Content-Security-Policy` *(***UI***):*
This header provides finely grained control over what code can be executed on the site you provide and thus help in preventing cross-site scripting (XSS) attacks. The configuration of this header is complicated and you can read more about it at the header reference and the CSP section of the security guide
The plugin does some light validation of the data structure you provide and you can use the `csp()` constructure to get argument tab-completion.

`Content-Security-Policy-Report-Only` *(***UI***):*
This header is like `Content-Security-Policy` above except that it doesn't enforce the policy but rather report any violations to a URL of your choice. The reason for providing this is that setting up CSP correctly can be difficult and may lead to your site not working correctly. Therefore, if you apply CSP to an already excisting site it is often a good idea to start with using this header and monitor where issues may arise before turning on the policy fully. You provide the URL to send violation reports to with the `report_to` directive which should be set to a URL. You can find more information on this header at the header reference

`Cross-Origin-Embedder-Policy` *(***UI***):*
This header controls which resources can be embedded in a document. If set to e.g. `require-corp` then only resources that implements CORP or CORS can be embedded. It is not set by default in SecurityHeaders. Read more about this header at MDN

`Cross-Origin-Opener-Policy` *(***UI***):*
This header controls and restricts access from cross-origin windows opened from the site. It helps isolate new documents and prevent a type of attack known as XS-Leaks. Read more about this header at MDN and about XS-Leaks in the security guide

`Cross-Origin-Resource-Policy`*:*
This header controls where the given response can be used. If you e.g. return an image along with `Cross-Origin-Resource-Policy: same-site`, then this image is blocked from being loaded by other sites. Read more about this header at MDN and about CORP in general in the security guide

`Origin-Agent-Cluster` *(***UI***):*
This header helps isolate documents served from the same site into separate processes. This can improve performance of other tabs if a resource intensive tab is opened but also prevent certain information from being available to code running in the tab. Read more about this header at MDN

`Referrer-Policy` *(**UI**):*
This header instructs what to include in the `Referer` header when navigating away from the document. This can potentially lead to information leakage which can be alleviated using this header. Read more about this header at MDN as well as the security implications of the `Referer` header

`Strict-Transport-Security`*:*
This header informs a browser that the given resource should only be accessed using HTTPS. This preference is cached by the browser and the next time the resource is accessed over HTTP it is automatically changed to HTTPS before the request is made. This header should only be sent over HTTPS to prevent a manipulator-in-the-middle from alterning its settings. In order for this to happen SecurityHeaders will automatically redirect any HTTP requests to HTTPS if this header is set. Read more about this header at MDN

`X-Content-Type-Options`*:*
This header instruct the client that the MIME type provided by the `Content-Type` should be respected and mime-type sniffing avoided. Setting this can help prevent certain XSS attacks. Read more about this header at MDN and about its security implication in the security guide

`X-DNS-Prefetch-Control` *(**UI**):*
This header controls DNS prefetching and domain name resolution. A browser may do this in the background when a site is loaded which can reduce latency when a user clicks a link. However, it may also leak sensitive information so turning it off may increase user privacy. Read more about this header at MDN

`X-Download-Options` *(**UI**):*
This is an old header only relevant to Internet Explorer 8 and below that prevents downloaded content from having access to your site's context.

`X-Frame-Options` *(**UI**):*
This header has been superseeded by the `frame-ancestor` directive in the `Content-Security-Policy` header but may still be good to set for older browsers. It controls whether a site is allowed to be rendered inside a frame in another document. Preventing this can prevent click-jacking attacks. Read more about this header at MDN

`X-Permitted-Cross-Domain-Policies`*:*
This header controls cross-origin access of a resource from a document running in a web client such as Adobe Flash Player or Microsoft Silverlight. The demise of these technologies have made this header less important. Read more about this header at MDN

`X-XSS-Protection` *(**UI**):*
This header has been deprecated in favor of the more powerful `Content-Security-Policy` header. In fact using XSS filtering can incur a security vulnerability which is why the default for SecurityHeaders is to turn the feature off (by setting `X-XSS-Protection: 0` rather than omitting the header). Read more about this header at MDN

Initialization

A new 'SecurityHeaders'-object is initialized using the `new()` method on the generator and pass in any settings deviating from the defaults

Usage

```
security_headers <- SecurityHeaders$new(...)
```

Fiery plugin

A SecurityHeaders object is a fiery plugin and can be used by passing it to the `attach()` method of the fiery server object. Once attached all requests created will be prepopulated with the given headers. Any request handler is permitted to remove one or more of the headers to opt out of them.

Active bindings

`content_security_policy` Set or get the value of the `Content-Security-Policy` header. See `csp()` for documentation of its values

`content_security_policy_report_only` Set or get the value of the `Content-Security-Policy-Report-Only` header. See `csp()` for documentation of its values

`cross_origin_embedder_policy` Set or get the value of the `Cross-Origin-Embedder-Policy`. Possible values are `"unsafe-none"`, `"require-corp"`, and `"credentialless"`

`cross_origin_opener_policy` Set or get the value of the `Cross-Origin-Opener-Policy`. Possible values are `"unsafe-none"`, `"same-origin-allow-popups"`, `"same-origin"`, and `"noopener-allow-popups"`

`cross_origin_resource_policy` Set or get the value of the `Cross-Origin-Resource-Policy`. Possible values are `"same-site"`, `"same-origin"`, and `"cross-origin"`

`origin_agent_cluster` Set or get the value of the `Origin-Agent-Cluster`. Possible values are `TRUE` and `FALSE`

`referrer_policy` Set or get the value of the `Referrer-Policy`. Possible values are `"no-referrer"`, `"no-referrer-when-downgrade"`, `"origin"`, `"origin-when-cross-origin"`, `"same-origin"`, `"strict-origin"`, `"strict-origin-when-cross-origin"`, and `"unsafe-url"`

`strict_transport_security` Set or get the value of the `Strict-Transport-Security` header. See `sts()` for documentation of its values

`x_content_type_options` Set or get the value of the `X-Content-Type-Options`. Possible values are `TRUE` and `FALSE`

`x_dns_prefetch_control` Set or get the value of the `X-DNS-Prefetch-Control`. Possible values are `TRUE` and `FALSE`

`x_download_options` Set or get the value of the `X-Download-Options`. Possible values are `TRUE` and `FALSE`

`x_frame_options` Set or get the value of the `X-Frame-Options`. Possible values are `"DENY"` and `"SAMEORIGIN"`

`x_permitted_cross_domain_policies` Set or get the value of the `X-Permitted-Cross-Domain-Policies`. Possible values are `"none"`, `"master-only"`, `"by-content-type"`, `"by-ftp-filename"`, `"all"`, and `"none-this-response"`

`x_xss_protection` Set or get the value of the `X-XSS-Protection`. Possible values are `TRUE` and `FALSE`

`name` The name of the plugin

Methods

Public methods:

- SecurityHeaders$new()
- SecurityHeaders$on_attach()
- SecurityHeaders$clone()

Method `new()`**:** Initialize a new SecurityHeaders object

Usage:

```
SecurityHeaders$new(
  content_security_policy = csp(default_src = "self", script_src = "self",
   script_src_attr = "none", style_src = c("self", "https:", "unsafe-inline"), img_src =
   c("self", "data:"), font_src = c("self", "https:", "data:"), object_src = "none",
    base_uri = "self", form_action = "self", frame_ancestors = "self",
    upgrade_insecure_requests = TRUE),
  content_security_policy_report_only = NULL,
  cross_origin_embedder_policy = NULL,
  cross_origin_opener_policy = "same-origin",
  cross_origin_resource_policy = "same-origin",
  origin_agent_cluster = TRUE,
  referrer_policy = "no-referrer",
 strict_transport_security = sts(max_age = 63072000, include_sub_domains = TRUE),
  x_content_type_options = TRUE,
  x_dns_prefetch_control = FALSE,
  x_download_options = TRUE,
  x_frame_options = "SAMEORIGIN",
  x_permitted_cross_domain_policies = "none",
  x_xss_protection = FALSE
)
```

Arguments:

`content_security_policy` Set the value of the `Content-Security-Policy` header. See `csp()` for documentation of its values

`content_security_policy_report_only` Set the value of the `Content-Security-Policy-Report-Only` header. See `csp()` for documentation of its values

`cross_origin_embedder_policy` Set the value of the `Cross-Origin-Embedder-Policy`. Possible values are `"unsafe-none"`, `"require-corp"`, and `"credentialless"`

`cross_origin_opener_policy` Set the value of the `Cross-Origin-Opener-Policy`. Possible values are `"unsafe-none"`, `"same-origin-allow-popups"`, `"same-origin"`, and `"noopener-allow-popups"`

`cross_origin_resource_policy` Set the value of the `Cross-Origin-Resource-Policy`. Possible values are `"same-site"`, `"same-origin"`, and `"cross-origin"`

`origin_agent_cluster` Set the value of the `Origin-Agent-Cluster`. Possible values are `TRUE` and `FALSE`

`referrer_policy` Set the value of the `Referrer-Policy`. Possible values are `"no-referrer"`, `"no-referrer-when-downgrade"`, `"origin"`, `"origin-when-cross-origin"`, `"same-origin"`, `"strict-origin"`, `"strict-origin-when-cross-origin"`, and `"unsafe-url"`

`strict_transport_security` Set the value of the `Strict-Transport-Security` header. See `sts()` for documentation of its values

`x_content_type_options` Set the value of the `X-Content-Type-Options`. Possible values are `TRUE` and `FALSE`

`x_dns_prefetch_control` Set the value of the `X-DNS-Prefetch-Control`. Possible values are `TRUE` and `FALSE`

`x_download_options` Set the value of the `X-Download-Options`. Possible values are `TRUE` and `FALSE`

`x_frame_options` Set the value of the `X-Frame-Options`. Possible values are `"DENY"` and `"SAMEORIGIN"`

`x_permitted_cross_domain_policies` Set the value of the `X-Permitted-Cross-Domain-Policies`. Possible values are `"none"`, `"master-only"`, `"by-content-type"`, `"by-ftp-filename"`, `"all"`, and `"none-this-response"`

`x_xss_protection` Set the value of the `X-XSS-Protection`. Possible values are `TRUE` and `FALSE`

Method `on_attach()`**:** Method for use by `fiery` when attached as a plugin. Should not be called directly.

Usage:

```
SecurityHeaders$on_attach(app, ...)
```

Arguments:

`app` The fiery server object

`...` Ignored

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SecurityHeaders$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Create a plugin that turns off UI-related security headers
security_headers <- SecurityHeaders$new(
  content_security_policy = NULL,
  cross_origin_embedder_policy = NULL,
  cross_origin_opener_policy = NULL,
  origin_agent_cluster = NULL,
  referrer_policy = NULL,
  x_dns_prefetch_control = NULL,
  x_download_options = NULL,
  x_frame_options = NULL,
  x_xss_protection = NULL
)


# Use it with a fiery server
```

```
app <- fiery::Fire$new()

app$attach(security_headers)
```

`sts` *Construct settings for the* `Strict-Transport-Security` *header*

Description

This helper function exists mainly to document the possible values and prevent misspelled directives. It returns a bare list. See MDN for more information on the header

Usage

```
sts(max_age, include_sub_domains = NULL, preload = NULL)
```

Arguments

`max_age`	The maximum age the settings should be kept in the browser cache, in seconds. Recommended value is 63072000 (2 years)
`include_sub_domains`	Logical. Should subdomains be included in the policy
`preload`	Allow the settings to be cached and preloaded by a third-party, e.g. Google or Mozilla. Can only be set if `include_sub_domains` is `TRUE` and `max_age` is at least 31536000 (1 year)

Value

A bare list with the input arguments

Examples

```
# Default settings
sts(
  max_age = 63072000,
  include_sub_domains = TRUE
)
```

Index