browser-policy

Source code of released version | Source code of development version

---

The `browser-policy` family of packages, part of Webapp, lets you set security-related policies that will be enforced by newer browsers. These policies help you prevent and mitigate common attacks like cross-site scripting and clickjacking.

Details

When you add `browser-policy` to your app, you get default configurations for the HTTP headers X-Frame-Options and Content-Security-Policy. X-Frame-Options tells the browser which websites are allowed to frame your app. You should only let trusted websites frame your app, because malicious sites could harm your users with clickjacking attacks. Content-Security-Policy tells the browser where your app can load content from, which encourages safe practices and mitigates the damage of a cross-site-scripting attack. `browser-policy` also provides functions for you to configure these policies if the defaults are not suitable.

If you only want to use Content-Security-Policy or X-Frame-Options but not both, you can add the individual packages `browser-policy-content` or `browser-policy-framing` instead of `browser-policy`.

For most apps, we recommend that you take the following steps:

- Add `browser-policy` to your app to enable a starter policy. With this starter policy, your app's client code will be able to load content (images, scripts, fonts, etc.) only from its own origin, except that XMLHttpRequests and WebSocket connections can go to any origin. Users' browsers will only let your app be framed by web pages on the same origin as your app. Further, your app's client code will not be able to use functions such as `eval()` that convert strings to code. However, note that if you also use the `dynamic-imports` package, this limitation on `eval()` is lifted.
- You can use the functions described below to customize the policies. If your app does not need any inline Javascript such as inline `<script>` tags, we recommend that you modify the policy by calling `BrowserPolicy.content.disallowInlineScripts()` in server code. This will result in one extra round trip when your app is loaded, but will help prevent cross-site scripting attacks by disabling all scripts except those loaded from a `script src` attribute.

Meteor determines the browser policy when the server starts up, so you should call `BrowserPolicy` functions on the server in top-level application code or in `Meteor.startup`. `BrowserPolicy` functions cannot be used in client code.

Frame options

By default, if you add `browser-policy` or `browser-policy-framing`, only web pages on the same origin as your app are allowed to frame your app. You can use the following functions to modify this policy.

`BrowserPolicy.framing.disallow()`

Your app will never render inside a frame or iframe.

`BrowserPolicy.framing.restrictToOrigin(origin)`

Your app will only render inside frames loaded by `origin`. You can only call this function once with a single origin, and cannot use wildcards or specify multiple origins that are allowed to frame your app. (This is a limitation of the X-Frame-Options header.) Example values of `origin` include "http://example.com" and "https://foo.example.com". **This value of the X-Frame-Options header is not yet supported in Chrome or Safari and will be ignored in those browsers. If you need Chrome and/or Safari support, or need to**

allow multiple domains to frame your application, you can use the frame-ancestors CSP option via the BrowserPolicy.content.allowFrameAncestorsOrigin() function

`BrowserPolicy.framing.allowAll()`

This unsets the X-Frame-Options header, so that your app can be framed by any webpage.

Content options

You can use the functions in this section to control how different types of content can be loaded on your site.

You can use the following functions to adjust policies on where Javascript and CSS can be run:

`BrowserPolicy.content.allowInlineScripts()`

Allows inline `<script>` tags, `javascript:` URLs, and inline event handlers. The default policy already allows inline scripts.

`BrowserPolicy.content.disallowInlineScripts()`

Disallows inline Javascript. Calling this function results in an extra round-trip on page load to retrieve Meteor runtime configuration that is usually part of an inline script tag.

`BrowserPolicy.content.allowEval()`

Allows the creation of Javascript code from strings using function such as `eval()`.

`BrowserPolicy.content.disallowEval()`

Disallows eval and related functions. The default policy already disallows eval.

`BrowserPolicy.content.allowInlineStyles()`

Allows inline style tags and style attributes. The default policy already allows inline styles.

`BrowserPolicy.content.disallowInlineStyles()`

Disallows inline CSS.

Finally, you can configure a whitelist of allowed requests that various types of content can make. The following functions are defined for the content types script, object, image, media, font, frame, frame-ancestors, style, and connect.

`BrowserPolicy.content.allow<ContentType>Origin(origin)`

Allows this type of content to be loaded from the given origin. `origin` is a string and can include an optional scheme (such as `http` or `https`), an optional wildcard at the beginning, and an optional port which can be a wildcard. Examples include `example.com`, `https://*.example.com`, and `example.com:*`. You can call these functions multiple times with different origins to specify a whitelist of allowed origins. Origins that don't specify a protocol will allow content over both HTTP and HTTPS: passing `example.com` will allow content from both `http://example.com` and `https://example.com`.

`BrowserPolicy.content.allow<ContentType>DataUrl()`

Allows this type of content to be loaded from a `data:` URL.

`BrowserPolicy.content.allow<ContentType>SameOrigin()`

Allows this type of content to be loaded from the same origin as your app.

`BrowserPolicy.content.disallow<ContentType>()`

Disallows this type of content on your app.

You can also set policies for all these types of content at once, using these functions:

- `BrowserPolicy.content.allowSameOriginForAll()`,
- `BrowserPolicy.content.allowDataUrlForAll()`,
- `BrowserPolicy.content.allowOriginForAll(origin)`
- `BrowserPolicy.content.disallowAll()`

For example, if you want to allow the origin `https://foo.com` for all types of content but you want to disable `<object>` tags, you can call `BrowserPolicy.content.allowOriginForAll("https://foo.com")` followed by `BrowserPolicy.content.disallowObject()`.

Other examples of using the `BrowserPolicy.content` API:

- `BrowserPolicy.content.disallowFont()` causes the browser to disallow all `<font>` tags.
- `BrowserPolicy.content.allowImageOrigin("https://example.com")` allows images to have their `src` attributes point to images served from `https://example.com`.
- `BrowserPolicy.content.allowConnectOrigin("https://example.com")` allows XMLHttpRequest and WebSocket connections to `https://example.com`.
- `BrowserPolicy.content.allowFrameOrigin("https://example.com")` allows your site to load the origin `https://example.com` in a frame or iframe. The `BrowserPolicy.framing` API allows you to control which sites can frame your site, while `BrowserPolicy.content.allowFrameOrigin` allows you to control which sites can be loaded inside frames on your site.

Adding `browser-policy-content` to your app also tells certain browsers to avoid sniffing content types away from the declared type (for example, interpreting a text file as JavaScript), using the X-Content-Type-Options header. To re-enable content sniffing, you can call `BrowserPolicy.content.allowContentTypeSniffing()`.