Django-CSP Documentation

Release 3.5

James Socol, Mozilla

Jan 16, 2019

Contents

1 Installing django-csp **3**

2 Configuring django-csp **5**
2.1 Policy Settings 5
2.2 Other Settings 6

3 Modifying the Policy with Decorators **7**
3.1 `@csp_exempt` 7
3.2 `@csp_update` 7
3.3 `@csp_replace` 8
3.4 `@csp` 8

4 Using the generated CSP nonce **9**
4.1 `Middleware` 9
4.2 `Context Processor` 9
4.3 `Django Template Tag/Jinja Extension` 10

5 CSP Violation Reports **11**
5.1 Throttling the number of reports 11

6 Contributing **13**
6.1 Style 13
6.2 Tests 13

7 Indices and tables **15**

django-csp adds Content-Security-Policy headers to Django applications.

Version 3.5

Code https://github.com/mozilla/django-csp

License BSD; see LICENSE file

Issues https://github.com/mozilla/django-csp/issues

Contents:

CHAPTER 1

Installing django-csp

First, install django-csp via pip or from source:

```
# pip
$ pip install django-csp
```

```
# source
$ git clone https://github.com/mozilla/django-csp.git
$ cd django-csp
$ python setup.py install
```

Now edit your project's `settings` module, to add the django-csp middleware to `MIDDLEWARE`, like so:

```
MIDDLEWARE = (
    # ...
    'csp.middleware.CSPMiddleware',
    # ...
)
```

That should do it! Go on to *configuring CSP*.

CHAPTER 2

Configuring django-csp

Content-Security-Policy is a complicated header. There are many values you may need to tweak here.

Note: Note when a setting requires a tuple or list. Since Python strings are iterable, you may get very strange policies and errors.

It's worth reading the latest CSP spec and making sure you understand it before configuring django-csp.

2.1 Policy Settings

These settings affect the policy in the header. The defaults are in *italics*.

Note: The "special" source values of `'self'`, `'unsafe-inline'`, `'unsafe-eval'`, `'none'` and hash-source (`'sha256-...'`) must be quoted! e.g.: `CSP_DEFAULT_SRC = ("'self'",)`. Without quotes they will not work as intended.

CSP_DEFAULT_SRC Set the `default-src` directive. A tuple or list of values, e.g. `("'self'", 'cdn.example.net')`. *'self'*

CSP_SCRIPT_SRC Set the `script-src` directive. A tuple or list. *None*

CSP_IMG_SRC Set the `img-src` directive. A tuple or list. *None*

CSP_OBJECT_SRC Set the `object-src` directive. A tuple or list. *None*

CSP_MEDIA_SRC Set the `media-src` directive. A tuple or list. *None*

CSP_FRAME_SRC Set the `frame-src` directive. A tuple or list. *None*

CSP_FONT_SRC Set the `font-src` directive. A tuple or list. *None*

CSP_CONNECT_SRC Set the `connect-src` directive. A tuple or list. *None*

CSP_STYLE_SRC Set the `style-src` directive. A tuple or list. *None*

CSP_BASE_URI Set the `base-uri` directive. A tuple or list. *None* Note: This doesn't use default-src as a fall-back.

CSP_CHILD_SRC Set the `child-src` directive. A tuple or list. *None* Note: Deprecated in CSP v3. Use frame-src and worker-src instead.

CSP_FRAME_ANCESTORS Set the `FRAME_ANCESTORS` directive. A tuple or list. *None* Note: This doesn't use default-src as a fall-back.

CSP_FORM_ACTION Set the `FORM_ACTION` directive. A tuple or list. *None* Note: This doesn't use default-src as a fall-back.

CSP_SANDBOX Set the `sandbox` directive. A tuple or list. *None* Note: This doesn't use default-src as a fall-back.

CSP_REPORT_URI Set the `report-uri` directive. A tuple or list. Each URI can be a full or relative URI. *None* Note: This doesn't use default-src as a fall-back.

CSP_MANIFEST_SRC Set the `manifest-src` directive. A tuple or list. *None*

CSP_WORKER_SRC Set the `worker-src` directive. A tuple or list. *None*

CSP_PLUGIN_TYPES Set the `plugin-types` directive. A tuple or list. *None* Note: This doesn't use default-src as a fall-back.

CSP_REQUIRE_SRI_FOR Set the `require-sri-for` directive. A tuple or list. *None* Valid values: `script`, `style`, or both. See: require-sri-for-known-tokens Note: This doesn't use default-src as a fall-back.

CSP_UPGRADE_INSECURE_REQUESTS Include `upgrade-insecure-requests` directive. A boolean. *False* See: upgrade-insecure-requests

CSP_BLOCK_ALL_MIXED_CONTENT Include `block-all-mixed-content` directive. A boolean. *False* See: block-all-mixed-content

CSP_INCLUDE_NONCE_IN Include dynamically generated nonce in all listed directives, e.g. `CSP_INCLUDE_NONCE_IN=['script-src']` will add `'nonce-<b64-value>'` to the `script-src` directive. A tuple or list. *None*

2.1.1 Changing the Policy

The policy can be changed on a per-view (or even per-request) basis. See the *decorator documentation* for more details.

2.2 Other Settings

These settings control the behavior of django-csp. Defaults are in *italics*.

CSP_REPORT_ONLY Send "report-only" headers instead of real headers. See the spec and the chapter on *reports* for more info. A boolean. *False*

CSP_EXCLUDE_URL_PREFIXES A **tuple** of URL prefixes. URLs beginning with any of these will not get the CSP headers. *()*

> **Warning:** Excluding any path on your site will eliminate the benefits of CSP everywhere on your site. The typical browser security model for JavaScript considers all paths alike. A Cross-Site Scripting flaw on, e.g., *excluded-page/* can therefore be leveraged to access everything on the same origin.

CHAPTER 3

Modifying the Policy with Decorators

Content Security Policies should be restricted and paranoid by default. You may, on some views, need to expand or change the policy. django-csp includes four decorators to help.

3.1 @csp_exempt

Using the `@csp_exempt` decorator disables the CSP header on a given view.

```
from csp.decorators import csp_exempt

# Will not have a CSP header.
@csp_exempt
def myview(request):
    return render(...)
```

You can manually set this on a per-response basis by setting the `_csp_exempt` attribute on the response to `True`:

```
# Also will not have a CSP header.
def myview(request):
    response = render(...)
    response._csp_exempt = True
    return response
```

3.2 @csp_update

The `@csp_update` header allows you to **append** values to the source lists specified in the settings. If there is no setting, the value passed to the decorator will be used verbatim.

Note: To quote the CSP spec: "There's no inheritance; ... the default list is not used for that resource type" if it is set. E.g., the following will not allow images from 'self':

```
default-src 'self'; img-src imgsrv.com
```

The arguments to the decorator the same as the *settings* without the `CSP_` prefix, e.g. `IMG_SRC`. (They are also case-insensitive.) The values are either strings, lists or tuples.

```
from csp.decorators import csp_update

# Will allow images from imgsrv.com.
@csp_update(IMG_SRC='imgsrv.com')
def myview(request):
    return render(...)
```

3.3 @csp_replace

The `@csp_replace` decorator allows you to **replace** a source list specified in settings. If there is no setting, the value passed to the decorator will be used verbatim. (See the note under `@csp_update`.)

The arguments and values are the same as `@csp_update`:

```
from csp.decorators import csp_replace

# settings.CSP_IMG_SRC = ['imgsrv.com']
# Will allow images from imgsrv2.com, but not imgsrv.com.
@csp_replace(IMG_SRC='imgsrv2.com')
def myview(request):
    return render(...)
```

3.4 @csp

If you need to set the entire policy on a view, ignoring all the settings, you can use the `@csp` decorator. The arguments and values are as above:

```
from csp.decorators import csp

@csp(DEFAULT_SRC=["'self'"], IMG_SRC=['imgsrv.com'],
     SCRIPT_SRC=['scriptsrv.com', 'googleanalytics.com'])
def myview(request):
    return render(...)
```

CHAPTER 4

Using the generated CSP nonce

When `CSP_INCLUDE_NONCE_IN` is configured, the nonce value is returned in the CSP header. To actually make the browser do anything with this value, you will need to include it in the attributes of the tags that you wish to mark as safe.

4.1 Middleware

Installing the middleware creates a lazily evaluated property `csp_nonce` and attaches it to all incoming requests.

```
    MIDDLEWARE_CLASSES = (
    #...
    'csp.middleware.CSPMiddleware',
    #...
)
```

This value can be accessed directly on the request object in any view or template and manually appended to any script element like so -

```
<script nonce="{{request.csp_nonce}}">
        var hello="world";
</script>
```

Assuming the `CSP_INCLUDE_NONCE_IN` list contains the `script-src` directive, this will result in the above script being allowed.

4.2 Context Processor

This library contains an optional context processor, adding `csp.context_processors.nonce` to your configured context processors exposes a variable called `nonce` into the global template context. This is simple shorthand for `request.csp_nonce`, but can be useful if you have many occurences of script tags.

```
<script nonce="{{nonce}}">
    var hello="world";
</script>
```

4.3 Django Template Tag/Jinja Extension

Note: If you're making use of `csp.extensions.NoncedScript` you need to have `jinja2>=2.9.6` installed, so please make sure to either use `django-csp[jinja2]` in your requirements or define it yourself.

Since it can be easy to forget to include the `nonce` property in a script tag, there is also a `script` template tag available for both Django templates and Jinja environments.

This tag will output a properly nonced script every time. For the sake of syntax highlighting, you can wrap the content inside of the `script` tag in `<script>` html tags, which will be subsequently removed in the rendered output. Any valid script tag attributes can be specified and will be forwarded into the rendered html.

Django:

```
{% load csp %}
{% script type="application/javascript" async=False %}
        <script>
                var hello='world';
        </script>
{% endscript %}
```

Jinja:

(assumes `csp.extensions.NoncedScript` is added to the jinja extensions setting)

```
{% script type="application/javascript" async=False %}
        <script>
                var hello='world';
        </script>
{% endscript %}
```

Will output -

```
<script nonce='123456' type="application/javascript" async=false></script>
```

CHAPTER 5

CSP Violation Reports

When something on a page violates the Content-Security-Policy, and the policy defines a `report-uri` directive, the user agent may POST a report. Reports are JSON blobs containing information about how the policy was violated.

Note: django-csp no longer handles report processing itself, so you will need to stand up your own app to receive them, or else make use of a third-party report processing service.

5.1 Throttling the number of reports

To throttle the number of requests made to your `report-uri` endpoint, you can use `csp.contrib.rate_limiting.RateLimitedCSPMiddleware` instead of `csp.middleware.CSPMiddleware` and set the `CSP_REPORT_PERCENTAGE` option:

`CSP_REPORT_PERCENTAGE` Percentage of requests that should see the `report-uri` directive. Use this to throttle the number of CSP violation reports made to your `CSP_REPORT_URI`. A **float** between 0 and 1 (0 = no reports at all). Ignored if `CSP_REPORT_URI` isn't set.

CHAPTER 6

Contributing

Patches are more than welcome! You can find the issue tracker on GitHub and we'd love pull requests.

6.1 Style

Patches should follow PEP8 and should not introduce any new violations as detected by the flake8 tool.

6.2 Tests

Patches fixing bugs should include regression tests (ideally tests that fail without the rest of the patch). Patches adding new features should test those features thoroughly.

To run the tests, install the requirements (probably into a virtualenv):

```
pip install -e .
pip install -e .[tests]
```

Then just py.test to run the tests:

```
py.test
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search