
django-security Documentation

Release 1.0

Pawel Krawczyk

January 07, 2015

1	Security models	3
2	Security views	5
3	Security middleware	7
4	Indices and tables	13
	Python Module Index	15

Contents:

Security models

class `security.models.CspReport` (**args*, ***kwargs*)

Content Security Policy violation report object. Each report represents a single alert raised by client browser in response to CSP received from the server.

Each alert means the browser was unable to access a web resource (image, CSS, frame, script) because server's policy prohibited it from accessing it. These alerts should be reviewed on regular basis, as they will occur in two cases: first, false positives where too restrictive CSP is blocking legitimate website features and needs tuning. Second, when real attacks were fired against the user and this raises a question how the malicious code appeared on your website.

CSP reports are available in Django admin view. To be logged into database, CSP reports view needs to be configured properly. See [csp_report](#) view for more information. Content Security Policy can be switched on for a web application using [ContentSecurityPolicyMiddleware](#) middleware.

class `security.models.PasswordExpiry` (**args*, ***kwargs*)

Associate a password expiry date with a user. For now, this date is effectively just a flag to tell us whether the user has ever changed their password, used to force users to change their initial passwords when they log in for the first time. Instances are created by `security.RequirePasswordChangeMiddleware`.

Security views

`security.views.csp_report(*args, **kwargs)`

Collect Content Security Policy reports from browsers. This view has two optional keyword arguments:

csp_save if True, reports will be saved as **CspReport** objects in database; this table is registered with Django Admin, so they can be later viewed in admin console

csp_log if True, reports will be logged through Django logging facility under `security` class

By default only logging is enabled. To collect reports, this view needs to be added to project's `urls.py`. Examples:

Default mode, only logger enable, no database logging:

```
url(r'^csp-report/$', security.views.csp_report),
```

Logger and database enabled:

```
url(r'^csp-report/$', security.views.csp_report,
    kwargs={'csp_save': True, 'csp_log': True}),
```

`security.views.require_ajax(view)`

A view decorator which ensures that the request being processed by view is an AJAX request. We return a 403 error if the request is not an AJAX request.

Security middleware

class `security.middleware.BaseMiddleware`

Abstract class containing some functionality common to all middleware that require configuration.

load_setting (*setting, value*)

Called initially for each of the keys in `REQUIRED_SETTINGS` and `OPTIONAL_SETTINGS`, and again whenever any of these settings change (from the `setting_changed` signal). Passed the setting key and the new value, which may be `None` for the keys in `OPTIONAL_SETTINGS`. If no setting keys are defined then this method is never called.

class `security.middleware.ContentNoSniff`

Sends `X-Content-Options` HTTP header to disable autodetection of MIME type of files returned by the server in Microsoft Internet Explorer. Specifically if this flag is enabled, MSIE will not load external CSS and JavaScript files unless server correctly declares their MIME type. This mitigates attacks where web page would for example load a script that was disguised as an user-supplied image.

Reference:

- [MIME-Handling Change: X-Content-Type-Options: nosniff](#)

process_response (*request, response*)

And `X-Content-Options: nosniff` to the response header.

class `security.middleware.ContentSecurityPolicyMiddleware`

Adds Content Security Policy (CSP) header to HTTP response. CSP provides fine grained instructions to the browser on location of allowed resources loaded by the page, thus mitigating attacks based on loading of untrusted JavaScript code such as Cross-Site Scripting.

The policy can be set in two modes, controlled by `CSP_MODE` options:

`CSP_MODE='enforce'` browser will enforce policy settings and log violations (*default*)

`CSP_MODE='report-only'` browser will not enforce policy, only report violations

The policy itself is a dictionary of content type keys and values containing list of allowed locations. For example, `img-src` specifies locations of images allowed to be loaded by this page:

```
'img-src' : [ 'img.example.com' ]
```

Content types and special location types (such as `none` or `self`) are defined in CSP draft (see [References](#)). The policy can be specified either as a dictionary, or a raw policy string:

Example of raw policy string (suitable for short policies):

```
CSP_STRING="allow 'self'; script-src *.google.com"
```

Example of policy dictionary (suitable for long, complex policies), with all supported content types (but not listing all supported locations):

```
CSP_DICT = {
    'default-src' : ['self', 'cdn.example.com' ],
    'script-src'  : ['self', 'js.example.com' ],
    'style-src'   : ['self', 'css.example.com' ],
    'img-src'     : ['self', 'img.example.com' ],
    'connect-src' : ['self' ],
    'font-src'    : ['fonts.example.com' ],
    'object-src'  : ['self' ],
    'media-src'   : ['media.example.com' ],
    'frame-src'   : ['self' ],
    'sandbox'     : [ '' ],
    # report URI is not array
    'report-uri'  : 'http://example.com/csp-report',
}
```

Notes:

- This middleware supports CSP header syntax for MSIE 10 (X-Content-Security-Policy), Firefox and Chrome (Content-Security-Policy) and Safari (X-WebKit-CSP).
- Enabling CSP has significant impact on browser behavior - for example inline JavaScript is disabled. Read [Default Policy Restrictions](#) to see how pages need to be adapted to work under CSP.
- Browsers will log CSP violations in JavaScript console and to a remote server configured by `report-uri` option. This package provides a view ([csp_report](#)) to collect these alerts in your application.

References:

- Content Security Policy 1.0,
- [HTML5.1 - Sandboxing](#)

process_response (*request, response*)

And Content Security Policy policy to the response header. Use either enforcement or report-only headers in all currently used variants.

class `security.middleware.DoNotTrackMiddleware`

With this middleware views can access `request.dnt` parameter to check client's preference on user tracking.

The parameter can take `True`, `False` or `None` values based on the presence of the `Do Not Track` HTTP header in client's request, which in turn depends on browser's configuration. The header indicates client's general preference to opt-out from behavioral profiling and third-party tracking.

The parameter does **not** change Django's behaviour in any way as its sole purpose is to pass the user's preference to application and it's up to the owner to implement a particular policy based on this information. Compliant website should adapt its behaviour depending on one of user's preferences:

- Explicit opt-out (`request.dnt=True`): Disable third party tracking for this request and delete all previously stored tracking data.
- Explicit opt-in (`request.dnt=False`): Website may track user.
- Header not present (`request.dnt=None`): Website may track user, but should not draw any conclusions on user's preferences.

For example, a website might respond to `request.dnt=True` by disabling template parts responsible for personalized statistics, targeted advertisements or switching to DNT aware ones.

Examples:

- [Do Not Track \(DNT\) tutorial for Django](#)
- [Do Not Track - Web Application Templates](#)

- [Opt-out of tailoring Twitter](#)

References:

- [Web Tracking Protection](#)
- [Do Not Track: A Universal Third-Party Web Tracking Opt Out](#)

process_request (*request*)

Read DNT header from browser request and create request attribute

process_response (*request, response*)

Echo DNT header in response per section 8.4 of draft-mayer-do-not-track-00

class `security.middleware.LoginRequiredMiddleware`

Middleware that requires a user to be authenticated to view any page on the site that hasn't been white listed. (The middleware also ensures the user is 'active'. Disabled users are also redirected to the login page.

Exemptions to this requirement can optionally be specified in settings via a list of regular expressions in LOGIN_EXEMPT_URLS (which you can copy from your urls.py).

Requires authentication middleware and template context processors to be loaded. You'll get an error if they aren't.

class `security.middleware.MandatoryPasswordChangeMiddleware`

Redirects any request from an authenticated user to the password change form if that user's password has expired. Must be placed after AuthenticationMiddleware in the middleware list.

Configured by dictionary MANDATORY_PASSWORD_CHANGE with the following keys:

URL_NAME name of the password change view EXEMPT_URL_NAMES list of URLs that do not trigger password change request

class `security.middleware.NoConfidentialCachingMiddleware`

Adds No-Cache and No-Store headers to confidential pages. You can either whitelist non-confidential pages and treat all others as non-confidential, or specifically blacklist pages as confidential. The behaviour is configured in NO_CONFIDENTIAL_CACHING dictionary in settings file with the following keys:

WHITELIST_ON all pages are confidential, except for pages explicitly whitelisted in WHITELIST_REGEXES

WHITELIST_REGEXES list of regular expressions defining pages exempt from the no caching policy

BLACKLIST_ON only pages defined in BLACKLIST_REGEXES will have caching disabled

BLACKLIST_REGEXES list of regular expressions defining confidential pages for which caching should be prohibited

Note: Django `cache_control` decorator allows more granular control of caching on individual view level.

Reference:

- [HTTP/1.1 Header definitions - What is Cacheable](#)

process_response (*request, response*)

Add the Cache control no-store to anything confidential. You can either whitelist non-confidential pages and treat all others as non-confidential, or specifically blacklist pages as confidential

class `security.middleware.P3PPolicyMiddleware`

Adds the HTTP header attribute specifying compact P3P policy defined in P3P_COMPACT_POLICY setting and location of full policy defined in P3P_POLICY_URL. If the latter is not defined, a default value is used (/w3c/p3p.xml). The policy file needs to be created by website owner.

Note: P3P work stopped in 2002 and the only popular browser with **limited** P3P support is MSIE.

Reference:

- [The Platform for Privacy Preferences 1.0 \(P3P1.0\) Specification - The Compact Policies](#)

process_response (*request, response*)

And P3P policy to the response header.

class `security.middleware.SessionExpiryPolicyMiddleware`

The session expiry middleware will let you expire sessions on browser close, and on expiry times stored in the cookie itself. (Expiring a cookie on browser close means you don't set the expiry value of the cookie.) The middleware will read `SESSION_COOKIE_AGE` and `SESSION_INACTIVITY_TIMEOUT` from the `settings.py` file to determine how long to keep a session alive.

We will purge a session that has expired. This middleware should be run before the `LoginRequired` middleware if you want to redirect the expired session to the login page (if required).

process_request (*request*)

Verify that the session should be considered active. We check the start time and the last activity time to determine if this is the case. We set the last activity time to `now()` if the session is still active.

class `security.middleware.StrictTransportSecurityMiddleware`

Adds Strict-Transport-Security header to HTTP response that enforces SSL connections on compliant browsers. Two parameters can be set in settings file, otherwise reasonable defaults will be used:

- `STS_MAX_AGE` time in seconds to preserve host's STS policy (default: 1 year)
- `STS_INCLUDE_SUBDOMAINS` True if subdomains should be covered by the policy as well (default: True)

Reference:

- [HTTP Strict Transport Security \(HSTS\)](#)

process_response (*request, response*)

Add Strict-Transport-Security header.

`security.middleware.XFrameOptionsDenyMiddleware`

alias of `XFrameOptionsMiddleware`

class `security.middleware.XFrameOptionsMiddleware`

Emits `X-Frame-Options` headers in HTTP response. These headers will instruct the browser to limit ability of this web page to be framed, or displayed within a `FRAME` or `IFRAME` tag. This mitigates password stealing attacks like Clickjacking and similar.

Use `X_FRAME_OPTIONS` in settings file with the following values:

- `deny` prohibit any framing of this page
- `sameorigin` allow frames from the same domain (*default*)
- `allow-from URL` allow frames from specified *URL*

Note: Frames and inline frames are frequently used by ads, social media plugins and similar widgets so test these features after setting this flag.

You can exclude certain URLs from this header by setting `X_FRAME_OPTIONS_EXCLUDE_URLS` to a list of URL regexes like so:

```
X_FRAME_OPTIONS_EXCLUDE_URLS = (
    r'^/some/url/here$',      # Note the initial slash
    r'^/another/to/exclude$',
)
```

The header will be sent unless `request.path` matches any of the above list. For more granular control, use `ContentSecurityPolicyMiddleware`.

References:

- RFC 7034: HTTP Header Field X-Frame-Options

process_response (*request, response*)

Add X-Frame-Options and Frame-Options to the response header.

class `security.middleware.XssProtectMiddleware`

Sends X-XSS-Protection HTTP header that controls Cross-Site Scripting filter on MSIE. Use `XSS_PROTECT` option in settings file with the following values:

```

sanitize enable XSS filter that tries to sanitize requests instead of blocking (de-
fault) on enable full XSS filter blocking XSS requests (may leak document.referrer
<http://homakov.blogspot.com/2013/02/hacking-with-xss-auditor.html>_) off completely disable
XSS filter

```

Reference:

- Controlling the XSS Filter

process_response (*request, response*)

Add X-XSS-Protection to the response header.

Indices and tables

- *genindex*
- *modindex*
- *search*

S

`security.middleware`, [7](#)
`security.models`, [3](#)
`security.views`, [5](#)

B

BaseMiddleware (class in security.middleware), 7

C

ContentNoSniff (class in security.middleware), 7

ContentSecurityPolicyMiddleware (class in security.middleware), 7

csp_report() (in module security.views), 5

CspReport (class in security.models), 3

D

DoNotTrackMiddleware (class in security.middleware), 8

L

load_setting() (security.middleware.BaseMiddleware method), 7

LoginRequiredMiddleware (class in security.middleware), 9

M

MandatoryPasswordChangeMiddleware (class in security.middleware), 9

N

NoConfidentialCachingMiddleware (class in security.middleware), 9

P

P3PPolicyMiddleware (class in security.middleware), 9

PasswordExpiry (class in security.models), 3

process_request() (security.middleware.DoNotTrackMiddleware method), 9

process_request() (security.middleware.SessionExpiryPolicyMiddleware method), 10

process_response() (security.middleware.ContentNoSniff method), 7

process_response() (security.middleware.ContentSecurityPolicyMiddleware method), 8

process_response() (security.middleware.DoNotTrackMiddleware method), 9

process_response() (security.middleware.NoConfidentialCachingMiddleware method), 9

process_response() (security.middleware.P3PPolicyMiddleware method), 10

process_response() (security.middleware.StrictTransportSecurityMiddleware method), 10

process_response() (security.middleware.XFrameOptionsMiddleware method), 11

process_response() (security.middleware.XssProtectMiddleware method), 11

R

require_ajax() (in module security.views), 5

S

security.middleware (module), 7

security.models (module), 3

security.views (module), 5

SessionExpiryPolicyMiddleware (class in security.middleware), 10

StrictTransportSecurityMiddleware (class in security.middleware), 10

X

XFrameOptionsDenyMiddleware (in module security.middleware), 10

XFrameOptionsMiddleware (class in security.middleware), 10

XssProtectMiddleware (class in security.middleware), 11