django-security Documentation

Release 1.0

Pawel Krawczyk

March 19, 2013

CONTENTS

Contents:

CHAPTER

ONE

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 databse, 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.

CHAPTER

TWO

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 proccessed by view is an AJAX request. We return a 403 error if the request is not an AJAX request.

CHAPTER

THREE

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 <http://msdn.microsoft.com/en-us/library/ie/gg622941(v=vs.85).aspx>_*

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 signification impact on browser behavior - for example inline JavaScript is disabled. Read http://developer.chrome.com/extensions/contentSecurityPolicy.html 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 <http://www.w3.org/TR/CSP/>_*, *HTML5.1 - Sandboxing <http://www.w3.org/html/wg/drafts/html/master/single-page.html#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**
Sets request.dnt to True or False based on the presence of the Do Not Track HTTP header in request received from the client. The header indicates client’s general preference to opt-out from behavioral profiling and third-party tracking. Compliant website should adapt its behaviour depending on one of user’s implied 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`): Server may track user.

One form of tracking that DNT controls is using cookies, especially permanent or third-party cookies.

Reference: *Do Not Track: A Universal Third-Party Web Tracking Opt Out <http://tools.ietf.org/html/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 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 behaviouri is configured in NO_CONFIDENTIAL_CACHING dictionary in settings file with the following keys:

WHITELIST_ON all pages are confifendialt, 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 <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1>_*

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 <http://www.w3.org/TR/P3P/#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 middelware 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:

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) <https://datatracker.ietf.org/doc/rfc6797/>_*

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. For more granular control use **ContentSecurityPolicyMiddleware_**.

References: *Clickjacking Defense <http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx>_*

process_response(*request*, *response*)
And 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 (*default*) 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 <http://blogs.msdn.com/b/ieinternals/archive/2011/01/31/controlling-the-internet-explorer-xss-filter-with-the-x-xss-protection-http-header.aspx>_*

process_response(*request*, *response*)
Add X-XSS-Protection to the reponse header.

CHAPTER

FOUR

INDICES AND TABLES

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

PYTHON MODULE INDEX

S

security.middleware, **??**
security.models, **??**
security.views, **??**