django-security-logger Documentation

Release 1.4

Luboš Mátl

Apr 02, 2020
Django-security-logger is library for logging input, output request and Django commands. Library can be used with django-reversion to log which data was changed in a request. The library provides throttling security mechanism.
CHAPTER 1

Project Home

https://github.com/druids/django-security
2.1 Content

2.1.1 Installation

Using PIP

You can install django-security-logger via pip:

```
$ pip install django-security-logger
```

2.1.2 Configuration

After installation you must go through these steps:

**Required Settings**

The following variables have to be added to or edited in the project’s `settings.py`:

For using the library you just add `security` to `INSTALLED_APPS` variable:

```
INSTALLED_APPS = (  
    ...  
    'security',  
    ...  
)
```
Next you must add `security.middleware.LogMiddleware` to list of middlewares, the middleware should be added after authentication middleware:

```python
MIDDLEWARE = (  
    ...  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'security.middleware.LogMiddleware',  
    ...  
)
```

**Setup**

**SECURITY_DEFAULT_THROTTLING_VALIDATORS_PATH**
Path to the file with configuration of throttling validators. Default value is 'security.default_validators'.

**SECURITY_THROTTLING_FAILURE_VIEW**
Path to the view that returns throttling failure. Default value is 'security.views.throttling_failure_view'.

**SECURITY_LOG_REQUEST_IGNORE_URL_PATHS**
Set of URL paths that are omitted from logging.

**SECURITY_LOG_REQUEST_IGNORE_IP**
Tuple of IP addresses that are omitted from logging.

**SECURITY_LOG_REQUEST_BODY_LENGTH**
Maximal length of logged request body. More chars than defined are truncated. Default value is 1000. If you set None value the request body will not be truncated.

**SECURITY_LOG_RESPONSE_BODY_LENGTH**
Maximal length of logged response body. More chars than defined are truncated. Default value is 1000. If you set None value the response body will not be truncated.

**SECURITY_LOG_RESPONSE_BODY_CONTENT_TYPES**
Tuple of content types which request/response body are logged for another content types body are removed. Default value is ('application/json', 'application/xml', 'text/xml', 'text/csv', 'text/html', 'application/xhtml+xml').

**SECURITY_LOG_JSON_STRING_LENGTH**
If request/response body are in JSON format and body is longer than allowed the truncating is done with a smarter way. String JSON values longer than value of this setting are truncated. Default value is 250. If you set None value this method will not be used.

**SECURITY_COMMAND_LOG_EXCLUDED_COMMANDS**
Because logger supports Django command logging too this setting contains list of commands that are omitted from logging. Default value is ('runserver', 'makemigrations', 'migrate', 'sqlmigrate', 'showmigrations', 'shell', 'shell_plus', 'test', 'help', 'reset_db', 'compilemessages', 'makemessages', 'dumpdata', 'loaddata').

**SECURITY_HIDE_SENSITIVE_DATA PATTERNS**
Setting contains patterns for regex function that goes through body and headers and replaces sensitive data with defined replacement.

**SECURITY_SENSITIVE_DATA_REPLACEMENT**
Setting contains sensitive data replacement value. Default value is '[Filtered]'.

**SECURITY_APPEND_SLASH**
Setting same as Django setting APPEND_SLASH. Default value is True.
SECURITY_CELERY_STALE_TASK_TIME_LIMIT_MINUTES
Default wait timeout to set not triggered task to the failed state. Default value is 60.

SECURITY_LOG_OUTPUT_REQUESTS
Enable logging of output requests via logging module. Default value is True.

SECURITY_AUTO_GENERATE_TASKS_FOR_DJANGO_COMMANDS
List or set of Django commands which will be automatically transformed into celery tasks.

2.1.3 Models

class security.models.InputLoggedRequest
Model for storing log of input logged requests

    host
    CharField, contains host of logged request.

    method
    CharField, HTTP method of logged request.

    path
    CharField, URL path of logged request.

    queries
    JSONField, HTTP query dictionary.

    is_secure
    BooleanField, contains True if was used HTTPS.

    slug
    SlugField, slug where is stored view name.

    request_timestamp
    DateTimeField, date and time when request was received.

    request_headers
    JSONField request headers in dictionary format.

    request_body
    TextField, HTTP request body.

    response_timestamp
    DateTimeField, date and time when response was sent.

    response_code
    PositiveSmallIntegerField, response HTTP status code.

    response_headers
    JSONField response headers in dictionary format.

    response_body
    TextField, HTTP response body.

    status
    PositiveSmallIntegerField, status of request in choices (incomplete, info, warning, error, debug, critical).

    error_description
    TextField, value contains traceback of exception that was raised during request.

    exception_name
    CharField, value contains name of the exception that was raised during request.
user
    ForeignKey, foreign key to the logged user.

ip
    GenericIPAddressField, IP address of the client.

type
    PositiveSmallIntegerField, type of the request (common, throttled, successful login, unsuccessful login)

class security.models.OutputLoggedRequest
    Model for storing log of output logged requests

    host
        CharField, contains host of logged request.

    method
        CharField, HTTP method of logged request.

    path
        CharField, URL path of logged request.

    queries
        JSONField, HTTP query dictionary.

    is_secure
        BooleanField, contains True if was used HTTPS.

    slug
        SlugField, slug where is stored view name.

    request_timestamp
        DateTimeField, date and time when request was received.

    request_headers
        JSONField request headers in dictionary format.

    request_body
        TextField, HTTP request body.

    response_timestamp
        DateTimeField, date and time when response was sent.

    response_code
        PositiveSmallIntegerField, response HTTP status code.

    response_headers
        JSONField response headers in dictionary format.

    response_body
        TextField, HTTP response body.

    status
        PositiveSmallIntegerField, status of request in choices (incomplete, info, warning, error, debug, critical).

    error_description
        TextField, value contains traceback of exception that was raised during request.

    exception_name
        CharField, value contains name of the exception that was raised during request.
input_logged_request
    ForeignKey, foreign key to the input request during which was output request performed.

class security.models.OutputLoggedRequestRelatedObjects
    You can relate a model objects with output logged request.

    output_logged_request
    Relation to the output logged request.

class security.models.CommandLog
    Represents a log of a command run.

    start
    Date and time when command was started.

    stop
    Date and time when command finished.

    name
    Name of the command.

    input
    Arguments/options the command was run with.

    executed_from_command_line
    Flag that indicates if command was run from the command line.

    output
    Standard and error output of the command.

    is_successful
    Flag that indicates if command finished successfully.

class security.models.CeleryTaskLog
    Represents a log of a celery task initiation.

    celery_task_id
    Identifier of celery task. Computed as random uuid value.

    name
    Name of the task.

    queue_name
    Name of the task queue.

    input
    Input args and kwargs of the celery task.
task_args
List of task args which was serialized into JSONField.

task_kwargs
Dict of task kwargs which was serialized into JSONField.

estimated_time_of_first_arrival
Celery task estimated time of first arrival. Which was computed from celery task etc or countdown value.

expires
Time of a task expiration. Waiting task will not be run if the time is a thing of the past.

stale
Time when a task will be marked as stale and will be automatically set as expired.

is_set_as_stale
Boolean value that identifies if task is expired.

get_start()
Date and time when task was started.

get_stop()
Date and time when task finished.

get_state()
State of the task (WAITING, ACTIVE, SUCCEEDED, FAILED, RETRIED, EXPIRED).

class security.models.CeleryTaskRunLog
Represents a log of celery task run.

celery_task_id
Identifier of celery task. Computed as random uuid value. There can be more logs with the same task number. But with the different retries value (retried tasks have same celery task ID).

start
Date and time when task run was started.

stop
Date and time when task run finished.

name
Name of the task run.

state
State of the task run (ACTIVE, SUCCEEDED, FAILED, RETRIED).

error_message
Exception message when task fails.

queue_name
Name of the task queue.

input
Input args and kwargs of the celery task.

output
Standard and error output of the celery task.

task_args
List of task args which was serialized into JSONField.

task_kwargs
Dict of task kwargs which was serialized into JSONField.
retries
Task attempt number is the task was retried.

estimated_time_of_next_retry
Celery task estimated time of arrival of retried task. Which was computed from celery task etc or countdown value.

2.1.4 Commands

purgelogs
Remove old request, command or celery logs that are older than defined value, parameters:

- expiration - timedelta from which logs will be removed. Units are h - hours, d - days, w - weeks, m - months, y - years
- noinput - tells Django to NOT prompt the user for input of any kind
- backup - tells Django where to backup removed logs in JSON format
- type - tells Django what type of requests should be removed (input-request/output-request/command/celery)

setstaletaskstoerrorstate
Set tasks which is in WAITING state and was not started more than SECURITY_CELERY_STALE_TASK_TIME_LIMIT_MINUTES (by default 60 minutes) to the failed state.

runcelery
Run celery worker or beater with autoreload, parameters:

- type - type of the startup (beat or worker)
- celerysettings - path to the celery configuration file
- autoreload - tells Django to use the auto-reloader
- extra - extra celery startup arguments

celery_health_check
Check Celery queue health. Either by count of tasks with state WAITING (--max-tasks-count) or by time waiting in queue (--max-created-at-diff, in seconds) or both at once. Default queue name is default. You can change queue name with argument --queue-name.

2.1.5 Logger

Input requests
Input requests are logged automatically with security.middleware.LogMiddleware. The middleware creates security.models.InputLoggedRequest object before sending request to next middleware. Response data to the logged requests are completed in the end. You can found logged request in the Django request objects with that way request.input_logged_request.
Decorators

There are several decorators for views and generic views that can be used for view logging configuration:

- `security.decorators.hide_request_body` - decorator for view that removes request body from logged request
- `security.decorators.hide_request_body_all` - decorator for generic view class that removes request body from logged request
- `security.decorators.log_exempt` - decorator for view that exclude all requests to this view from logging
- `security.decorators.log_exempt_all` - decorator for generic view class that exclude all requests to this view from logging

Output requests

Logging of output requests is a little bit complicated and is related to the way how output requests are performed. You can enable logging of output requests to stdout via `SECURITY_LOG_OUTPUT_REQUESTS` (default True) in following format: "\{request_timestamp\}" "\{response_timestamp\}" "\{response_time\}" "\{http_code\}" "\{http_host\}" "\{http_path\}" "\{http_method\}" "\{slug\}". Security provides two ways how to log output requests:

requests

The first method is used for logging simple HTTP requests using requests library. The only change necessary is to import from `security.transport import security_requests as requests` instead of `import requests`. Same methods (get, post, put, ..) are available as in the requests library. Every method has two extra optional parameters:

- `slug` - text slug that is stored with the logged request to tag concrete logged value
- `related_objects` - list or tuple of related objects that will be related with output logged requests

suds

For SOAP based clients there are extensions to the suds library. You must only use `security.transport.security_suds.Client` class without standard suds client or `security.transport.security_suds.SecurityRequestsTransport` with standard suds client object. As init data of `security.transport.security_suds.SecurityRequestsTransport` you can send `slug` and `related_objects`. The `security.transport.security_suds.Client` has `slug` as initial parameter but related objects must be added via `add_related_objects(self, *related_objects)` method.

Decorators/context processors

`security.decorators.atomic_log` - because logged requests are stored in models, they are subject to rollback, if you are using transactions. To solve this problem you can use this decorator before Django `transaction.atomic` decorator. The logs are stored on the end of the transaction (even with raised exception). Decorator can be nested, logs are saved only with the last decorator. If you want to join a object with output request log you can use this decorator too. In the example user is logged with output request:
from security.decorators import atomic_log
from security.transport import security_requests as requests

user = User.objects.first()
with atomic_log(output_requests_slug='github-request', output_requests_related_objects=[user]):
    requests.get('https:///github.com/druids/')

Sensitive data

Because some sensitive data inside requests and responses should not be stored (for example password, authorization token, etc.) django-security-logger uses regex to find these cases and replace these values with information about hidden value. Patterns are set with SECURITY_HIDE_SENSITIVE_DATA_PATTERNS which default setting is:

```python
SECURITY_HIDE_SENSITIVE_DATA_PATTERNS = {
    'BODY': (
        r'^password\s*:\s*\((?:\:\\|\\[^\"]*\")\)*',
        r'<password>{{[^<]*}}',
        r'password=([^\&]*)',
        r'csrfmiddlewaretoken=([^&]*)',
        r'content-disposition: form-data; name="password"\r\n\n.*',
        r'^access_key': "([^\"]*)",
    ),
    'HEADERS': (
        r'Authorization',
        r'X_Authorization',
        r'Cookie',
        r'.*token.*',
    ),
    'QUERIES': (
        r'.*token.*',
    ),
}
```

Patterns are split to two groups BODY, HEADERS and QUERIES. There are names of HTTP headers and queries, whose values will be replaced by the replacement. The search is case insensitive. BODY is a little bit complicated. If regex groups are used in the pattern only these groups will be replaced with the replacement. If no groups are used, the whole pattern will be replaced.

Commands log

If you want to log commands you must only modify your manage.py file:

```python
if __name__ == '__main__':
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'settings')

    # Used function for security to log commands
    from security.management import execute_from_command_line

    sys.path.append(os.path.join(PROJECT_DIR, 'libs'))
    execute_from_command_line(sys.argv)
```
If you want to call command from code, you should use `security.management.call_command` instead of standard Django `call_command` function.

**Celery tasks log**

If you want to log celery tasks you must firstly install celery library (celery==4.3.0). Then you must define your task as in example:

```python
from security.tasks import LoggedTask

@celery_app.task(
    base=LoggedTask,
    bind=True,
    name='sum_task')
def sum_task(self, task_id, a, b):
    return a + b
```

Task result will be automatically logged to the `security.models.CeleryTaskLog`

You can use predefined celery task `security.tasks.call_django_command` to run arbitrary django command. For example:

```python
from security.tasks import call_django_command
call_django_command.apply_async(args=('check',))
```

```python
class security.tasks.LoggedTask
    There are several methods of LoggedTask which you can use for your advanced tasks logic.
    on_apply_task(task_log, args, kwargs, options)
        This method is called before the task is queued. You can override this method.
    on_start_task(task_run_log, args, kwargs)
        This method is called when the task was started.
    on_success_task(task_run_log, args, kwargs, retval)
        This method is called when the task was successfully completed.
    on_failure_task(task_run_log, args, kwargs, exc)
        This method is called when the task raised an exception and is not retried.
    on_retry_task(task_run_log, args, kwargs, exc)
        This method is called when the task raised an exception and is retried.
    apply_async_on_commit(args=None, kwargs=None, related_objects=None, **options)
        This method has the same behaviour as apply_async but runs task with on_commit django signal. Therefore it is initialized at the end of the django atomic block. You can relate task with a django model instances by using related_object argument.
    delay_on_commit(*args, **kwargs)
        This method is same as delay method only uses for task initialization apply_async_on_commit.
    apply_async_and_get_result(args=None, kwargs=None, timeout=None, propagate=True, related_objects=None, **options)
        This method applies task in an asynchronous way, wait defined timeout and get AsyncResult or TimeoutError. Timeout value None means endless waiting time.
    is_processing(related_objects=None)
        This method can be used for checking if task is waiting or active. If task is related with objects you can use related_objects to filter only these tasks.
```
2.1.6 Throttling

In terms of django-security-logger throttling is a process responsible for regulating the rate of incoming HTTP requests. There are many ways how to restrict number of requests that may depend on a concrete view. The simplest throttling is to restrict maximum number of request from one IP address per unit of time.

Default configuration

Default throttling configuration is set with SECURITY_DEFAULT_THROTTLING_VALIDATORS_PATH. The setting contains path to the file with throttling configuration. Default configuration is 'security.default_validators' and the config file content is:

```python
from .throttling import PerRequestThrottlingValidator

default_validators = {
    PerRequestThrottlingValidator(3600, 1000), # 1000 per an hour
    PerRequestThrottlingValidator(60, 20), # 20 per an minute
}
```

Validators

There are only three predefined throttling validators:

- `security.throttling.PerRequestThrottlingValidator` - init parameters are timeframe throttling timedelta in seconds, throttle_at number of request per one IP address per timeframe and error message.
- `security.throttling.UnsuccessfulLoginThrottlingValidator` - validator with same input parameters as previous validator but counts only unsuccessful login request.
- `security.throttling.SuccessfulLoginThrottlingValidator` - validator with same input parameters as previous validator but counts only requests from anonymous (not logged in) user.

Custom validator

Creating custom validator is very simple, you only create class with validate method that receives request and if request must be regulated the method raises `security.exception.ThrottlingException`:

```python
class CustomValidator:
    def validate(self, request):
        if should_regulate(request):
            raise ThrottlingException('Your custom message')
```

Decorators

Because throttling can be different per view, there are decorators for changing default validators for concrete view:

- `security.decorators.throttling_exempt()` - marks a view function as being exempt from the throttling protection.
- `security.decorators.throttling_exempt_all()` - marks a view class as being exempt from the throttling protection.
• `security.decorators.throttling(*validators, keep_default=True)` - add throttling validators for view function. You can remove default throttling validators with set `keep_default` to the `False` value.

• `security.decorators.throttling_all(*validators, keep_default=True)` - add throttling validators for view class. You can remove default throttling validators with set `keep_default` to the `False` value.

View

If `security.exception.ThrottlingException` is raised the specific error view is returned. You can change it with only overriding template named 429.html in your templates. With setting `SECURITY_THROTTLING_FAILURE_VIEW` you can change view function which default code is:

```python
from django.shortcuts import render
from django.utils.encoding import force_text

def throttling_failure_view(request, exception):
    response = render(request, '429.html', {'description': force_text(exception)})
    response.status_code = 429
    return response
```

2.1.7 Extra

Django-security-logger provides extra features to improve your logged data.

security.contrib.reversion_log

If you have installed `django-reversion` it is possible to relate input logged requests with concrete object change. Firstly you must add extension to your `INSTALLED_APPS` setting:

```python
INSTALLED_APPS = (
    ...
    'security.contrib.reversion_log',
    ...
)
```

For `django-reversion` version older than 2.x you must add middleware `security.contrib.reversion_log.middleware.RevisionLogMiddleware` too:

```python
MIDDLEWARE = (
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'security.middleware.LogMiddleware',
    'security.contrib.reversion_log.middleware.RevisionLogMiddleware',
    ...
)
```

Input logged requests and reversion revision objects are related via m2m model `security.contrib.reversion_log.models.InputRequestRevision`
security.contrib.debug_toolbar_log

If you are using django-debug-toolbar you can log toolbar results with logged request. You only add extension to your INSTALLED_APPS setting:

```python
INSTALLED_APPS = (
    ...
    'security.contrib.reversion_log',
    ...
)
```

And add `security.contrib.debug_toolbar_log.middleware.DebugToolbarLogMiddleware` on the first place:

```python
MIDDLEWARE = (
    'security.contrib.debug_toolbar_log.middleware.DebugToolbarLogMiddleware',
    ...
)
```

Finally you can start log debug toolbar settings with your logged requests by turning on settings:

```python
SECURITY_DEBUG_TOOLBAR = True
```

Do not forget turn on django DEBUG.

To show results in django-is-core you must set setting:

```python
SECURITY_SHOW_DEBUG_TOOLBAR = True
```
### Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Method/Attribute</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>apply_async_and_get_result() (security.tasks.LoggedTask method)</td>
<td>14</td>
</tr>
<tr>
<td>A</td>
<td>apply_async_on_commit() (security.tasks.LoggedTask method)</td>
<td>14</td>
</tr>
<tr>
<td>C</td>
<td>celery_task_id (security.models.CeleryTaskLog attribute)</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>celery_task_id (security.models.CeleryTaskRunLog attribute)</td>
<td>10</td>
</tr>
<tr>
<td>C</td>
<td>celery_task_run_log (security.models.OutputLoggedRequestRelatedObjects attribute)</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>command_log (security.models.OutputLoggedRequestRelatedObjects attribute)</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>content_object (security.models.OutputLoggedRequestRelatedObjects attribute)</td>
<td>9</td>
</tr>
<tr>
<td>C</td>
<td>content_type (security.models.OutputLoggedRequestRelatedObjects attribute)</td>
<td>9</td>
</tr>
<tr>
<td>D</td>
<td>delay_on_commit() (security.tasks.LoggedTask method)</td>
<td>14</td>
</tr>
<tr>
<td>E</td>
<td>error_description (security.models.InputLoggedRequest attribute)</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>error_description (security.models.OutputLoggedRequest attribute)</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>error_message (security.models.CeleryTaskRunLog attribute)</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>estimated_time_of_first_arrival (security.models.CeleryTaskLog attribute)</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td>estimated_time_of_next_retry (security.models.CeleryTaskRunLog attribute)</td>
<td>11</td>
</tr>
<tr>
<td>E</td>
<td>exception_name (security.models.InputLoggedRequest attribute)</td>
<td>7</td>
</tr>
<tr>
<td>E</td>
<td>exception_name (security.models.OutputLoggedRequest attribute)</td>
<td>8</td>
</tr>
<tr>
<td>E</td>
<td>executed_from_command_line (security.models.CommandLog attribute)</td>
<td>9</td>
</tr>
<tr>
<td>F</td>
<td>expires (security.models.CeleryTaskLog attribute)</td>
<td>10</td>
</tr>
<tr>
<td>G</td>
<td>get_start() (security.models.CeleryTaskLog method)</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>get_state() (security.models.CeleryTaskLog method)</td>
<td>10</td>
</tr>
<tr>
<td>G</td>
<td>get_stop() (security.models.CeleryTaskLog method)</td>
<td>10</td>
</tr>
<tr>
<td>H</td>
<td>host (security.models.InputLoggedRequest attribute)</td>
<td>7</td>
</tr>
<tr>
<td>H</td>
<td>host (security.models.OutputLoggedRequest attribute)</td>
<td>8</td>
</tr>
<tr>
<td>I</td>
<td>input (security.models.CeleryTaskLog attribute)</td>
<td>9</td>
</tr>
<tr>
<td>I</td>
<td>input (security.models.CeleryTaskRunLog attribute)</td>
<td>10</td>
</tr>
<tr>
<td>I</td>
<td>inputLogged_request (security.models.OutputLoggedRequest attribute)</td>
<td>8</td>
</tr>
<tr>
<td>I</td>
<td>ip (security.models.InputLoggedRequest attribute)</td>
<td>8</td>
</tr>
<tr>
<td>I</td>
<td>is_processing() (security.tasks.LoggedTask method)</td>
<td>14</td>
</tr>
<tr>
<td>I</td>
<td>is_secure (security.models.InputLoggedRequest attribute)</td>
<td>7</td>
</tr>
<tr>
<td>I</td>
<td>is_secure (security.models.OutputLoggedRequest attribute)</td>
<td>8</td>
</tr>
</tbody>
</table>
is_set_as_stale (security.models.CeleryTaskLog attribute), 10
is_successful (security.models.CommandLog attribute), 9

M

method (security.models.InputLoggedRequest attribute), 7
method (security.models.OutputLoggedRequest attribute), 8

N

name (security.models.CeleryTaskLog attribute), 9
name (security.models.CeleryTaskRunLog attribute), 10
name (security.models.CommandLog attribute), 9

O

object_id (security.models.OutputLoggedRequestRelatedObjects attribute), 9
on_apply_task () (security.tasks.LoggedTask method), 14
on_failure_task () (security.tasks.LoggedTask method), 14
on_retry_task () (security.tasks.LoggedTask method), 14
on_start_task () (security.tasks.LoggedTask method), 14
on_success_task () (security.tasks.LoggedTask method), 14
output (security.models.CeleryTaskRunLog attribute), 10
output (security.models.CommandLog attribute), 9
outputLogged_request (security.models.OutputLoggedRequestRelatedObjects attribute), 9

P

path (security.models.InputLoggedRequest attribute), 7
path (security.models.OutputLoggedRequest attribute), 8

Q

queries (security.models.InputLoggedRequest attribute), 7
queries (security.models.OutputLoggedRequest attribute), 8
queue_name (security.models.CeleryTaskLog attribute), 9
queue_name (security.models.CeleryTaskRunLog attribute), 10

R

request_body (security.models.InputLoggedRequest attribute), 7
request_headers (security.models.InputLoggedRequest attribute), 8
request_timestamp (security.models.InputLoggedRequest attribute), 7
response_body (security.models.InputLoggedRequest attribute), 8
response_code (security.models.InputLoggedRequest attribute), 7
response_headers (security.models.InputLoggedRequest attribute), 8
response_timestamp (security.models.InputLoggedRequest attribute), 7
retries (security.models.CeleryTaskRunLog attribute), 8

S

security.models.CeleryTaskLog (built-in class), 9
security.models.CeleryTaskRunLog (built-in class), 10
security.models.CommandLog (built-in class), 9
security.models.InputLoggedRequest (built-in class), 7
security.models.OutputLoggedRequest (built-in class), 8
security.models.OutputLoggedRequestRelatedObjects (built-in class), 9
security.tasks.LoggedTask (*built-in class*), 14
SECURITY_APPEND_SLASH, 6
SECURITY_AUTO_GENERATE_TASKS_FOR_DJANGO_COMMANDS, 7
SECURITY_CELERY_STALE_TASK_TIME_LIMIT_MINUTES, 6
SECURITY_COMMAND_LOG_EXCLUDED_COMMANDS, 6
SECURITY_DEFAULT_THROTTLING_VALIDATORS_PATH, 6
SECURITY_HIDE_SENSITIVE_DATA PATTERNS, 6
SECURITY_LOG_JSON_STRING_LENGTH, 6
SECURITY_LOG_OUTPUT_REQUESTS, 7
SECURITY_LOG_REQUEST_BODY_LENGTH, 6
SECURITY_LOG_REQUEST_IGNORE_IP, 6
SECURITY_LOG_REQUEST_IGNORE_URL_PATHS, 6
SECURITY_LOG_RESPONSE_BODY_CONTENT_TYPES, 6
SECURITY_LOG_RESPONSE_BODY_LENGTH, 6
SECURITY_SENSITIVE_DATA_REPLACEMENT, 6
SECURITY_THROTTLING_FAILURE_VIEW, 6
slug (security.models.InputLoggedRequest attribute), 7
slug (security.models.OutputLoggedRequest attribute), 8
stale (security.models.CeleryTaskLog attribute), 10
start (security.models.CeleryTaskRunLog attribute), 10
start (security.models.CommandLog attribute), 9
state (security.models.CeleryTaskRunLog attribute), 10
status (security.models.InputLoggedRequest attribute), 7
status (security.models.OutputLoggedRequest attribute), 8
stop (security.models.CeleryTaskRunLog attribute), 10
stop (security.models.CommandLog attribute), 9
T
task_args (security.models.CeleryTaskLog attribute), 9
task_args (security.models.CeleryTaskRunLog attribute), 10
task_kwargs (security.models.CeleryTaskLog attribute), 10
task_kwargs (security.models.CeleryTaskRunLog attribute), 10
type (security.models.InputLoggedRequest attribute), 8
U
user (security.models.InputLoggedRequest attribute), 7