django-sesame Documentation

Release 1.7

Aymeric Augustin

Jul 20, 2019

Contents

django-sesame provides frictionless authentication with "Magic Links" for your Django project.

It generates URLs containing an authentication token such as: https://example.com/?url_auth_token=AAAAARchl18CIQUlImmbV9q7PZk%3A89AEU34b0JLSrkT8Ty2RPISio5

Then it authenticates users based on tokens found in URLs.

Known use cases for django-sesame include:

1. Login by email, an increasingly attractive option on mobile where typing passwords is uncomfortable. This technique is prominently deployed by Slack.

   If you're doing this, you should define a small `SESAME_MAX_AGE`, perhaps 10 minutes, and consider enabling `SESAME_ONE_TIME`.

2. Authenticated links, typically if you're generating an export or a report and emailing a link to access it. An authenticated link will work even if the user isn't logged in on the device where they're opening it.

   Likewise, you should configure an appropriate `SESAME_MAX_AGE`, probably no more than a few days.

3. Non-critical private websites, for example for a family or club site, where users don't expect to manage a personal account with a password. Authorized users can bookmark personalized authenticated URLs.

   Here you can rely on the default settings because that's the original use case for which django-sesame was built.

CHAPTER 1

(In)security

Before using django-sesame in your project, please review the following advice carefully. (Also, please don't use security-sensitive libraries published by strangers on the Internet without checking what they do.)

The major security weakness in django-sesame is a direct consequence of the feature it implements: **whoever obtains an authentication token will be able to authenticate to your website.**

URLs end up in countless insecure places: emails, referer headers, proxy logs, browser history, etc. You can't avoid that. At best you can mitigate it by creating short-lived or single-use tokens, as described below.

Otherwise, a reasonable attempt has been made to provide a secure solution. django-sesame uses Django's signing framework to create signed tokens. It offers configurable options for token expiration or invalidation.

CHAPTER 2

Requirements

django-sesame is tested with:

- Django 1.11 (LTS), 2.1 and 2.2;
- all supported Python versions.

It builds upon `django.contrib.auth`.

It supports custom user models, provided:

- They have an integer or UUID primary key — other types could be added if there's a use case;
- They have `password` and `last_login` fields — most custom user models inherit them from `AbstractBaseUser`.

CHAPTER 3

Getting started

1. Install django-sesame and ua-parser:

```
$ pip install django-sesame
$ pip install ua-parser  # optional, see "Safari issues" below
```

2. Add `sesame.backends.ModelBackend` to `AUTHENTICATION_BACKENDS`:

```
AUTHENTICATION_BACKENDS += ['sesame.backends.ModelBackend']
```

3. Add `sesame.middleware.AuthenticationMiddleware` to `MIDDLEWARE`:

```
MIDDLEWARE += ['sesame.middleware.AuthenticationMiddleware']
```

   The best position for `sesame.middleware.AuthenticationMiddleware` is just after `django.contrib.auth.middleware.AuthenticationMiddleware`.

4. Generate authentication tokens with `sesame.utils.get_query_string(user)`.

That's all!

CHAPTER 4

Generating URLs

django-sesame provides two functions to generate authenticated URLs.

1. `sesame.utils.get_query_string(user)` returns a complete query string that you can append to any URL to enable one-click login.
2. `sesame.utils.get_parameters(user)` returns a dictionary of GET parameters to add to the query string, if you're already building one.

Share resulting URLs with your users while ensuring adequate confidentiality.

By default, the URL parameter is called `url_auth_token`. You can set the `SESAME_TOKEN_NAME` setting to a shorter name that doesn't conflict with query string parameters used by your application.

CHAPTER 5

Tokens lifecycle

By default, tokens don't expire but are tied to the password of the user. Changing the password invalidates the token. When the authentication backend uses salted passwords — that's been the default in Django for a long time — the token is invalidated even if the new password is identical to the old one.

If you want tokens to expire after a given amount of time, set the `SESAME_MAX_AGE` setting to a duration in seconds. Then each token will contain the time it was generated at and django-sesame will check if it's still valid at each login attempt.

If you want tokens to be usable only once, set the `SESAME_ONE_TIME` setting to `True`. In that case tokens are only valid if the last login date hasn't changed since they were generated. Since logging in changes the last login date, such tokens are usable at most once.

If you don't want tokens to be invalidated by password changes, set the `SESAME_INVALIDATE_ON_PASSWORD_CHANGE` setting to `False`. **This is strongly discouraged because it becomes impossible to invalidate tokens** short of changing the `SECRET_KEY` setting. If you're doing it anyway, you should set `SESAME_MAX_AGE` to a short value to minimize risks. This option may be useful for generating tokens during a signup process, when you don't know if the token will be used before or after initializing the password.

Finally, if the `is_active` attribute of a user is set to `False`, django-sesame rejects authentication tokens for this user.

CHAPTER 6

Per-view authentication

The configuration described in the "Getting started" section enables a middleware that looks for a token in every request and, if there is a valid token, logs the user in. It's as if they had submitted their username and password in a login form.

Sometimes this behavior is too blunt. For example, you may want to build a Magic Link that gives access to a specific view but doesn't log the user in permanently.

To achieve this, you can remove `sesame.middleware.AuthenticationMiddleware` from the `MIDDLEWARE` setting and authenticate the user with django-sesame in a view as follows:

```
from django.core.exceptions import PermissionDenied
from django.http import HttpResponse

from sesame.utils import get_user

def hello(request):
    user = get_user(request)
    if user is None:
        raise PermissionDenied
    return HttpResponse("Hello {}!".format(user))
```

When `SESAME_ONE_TIME` is enabled, `get_user()` updates the user's last login date in order to invalidate the token. When `SESAME_ONE_TIME` isn't enabled, it doesn't, because making a database write for every call to `get_user()` could degrade performance. You can override this behavior with the `update_last_login` keyword argument:

```
get_user(request, update_last_login=True)   # update last_login
get_user(request, update_last_login=False)  # don't update last_login
```

`get_user()` is a thin wrapper around the low-level `authenticate()` function from `django.contrib.auth`. If you use `authenticate()` to verify an authentication token, the `sesame.backends.ModelBackend` authentication backend expects an `url_auth_token` argument:

```
from django.contrib.auth import authenticate

user = authenticate(url_auth_token=...)
```

If you rely on `authenticate()`, you must update `user.last_login` to ensure one-time tokens are invalidated. Indeed, in `django.contrib.auth`, `authenticate()` is a low-level function and the higher-level `login()` function is responsible for updating `user.last_login`.

CHAPTER 7

Safari issues

django-sesame removes the token from the URL with a HTTP 302 Redirect after authenticating a user successfully. Unfortunately, in some scenarios, this triggers Safari's "Protection Against First Party Bounce Trackers". In that case, Safari clears cookies and the user is logged out.

To avoid this problem, django-sesame doesn't perform the redirect when it detects that the browser is Safari. This relies on the ua-parser package, which is an optional dependency. If it isn't installed, django-sesame always redirects.

CHAPTER 8

Stateless authentication

Technically, django-sesame can provide stateless authenticated navigation without `django.contrib.sessions`, provided all internal links include the authentication token, but that increases the security issues explained above.

If `django.contrib.sessions.middleware.SessionMiddleware` and `django.contrib.auth.middleware.AuthenticationMiddleware` aren't enabled, `sesame.middleware.AuthenticationMiddleware` sets `request.user` to the currently logged-in user or `AnonymousUser()`.

CHAPTER 9

Changelog

9.1 1.7

- Fixed invalidation of one-time tokens in `get_user()`.

9.2 1.6

- Fixed detection of Safari on iOS.

9.3 1.5

- Added support for single use tokens with the `SESAME_ONE_TIME` setting.
- Added support for not invalidating tokens on password change with the `SESAME_INVALIDATE_ON_PASSWORD_CHANGE` setting.
- Added compatibility with custom user models where the primary key is a UUID.
- Added the `get_user()` function to obtain a user instance from a request.
- Improved error message for pre-existing tokens when changing the `SESAME_MAX_AGE` setting.
- Fixed authentication on Safari by disabling the redirect which triggers ITP.

9.4 1.4

- Added a redirect to the same URL with the query string parameter removed.

9.5 1.3

- Added compatibility with Django 2.0.

9.6 1.2

- Added the ability to rename the query string parameter with the `SESAME_TOKEN_NAME` setting.
- Added compatibility with Django 1.8.

9.7 1.1

- Added support for expiring tokens with the `SESAME_MAX_AGE` setting.

9.8 1.0

- Initial release.