Django Two-Factor Authentication Documentation

Release 1.0.0

Bouke Haarsma

Jul 08, 2020

Contents

Complete Two-Factor Authentication for Django. Built on top of the one-time password framework django-otp and Django's built-in authentication framework `django.contrib.auth` for providing the easiest integration into most Django projects. Inspired by the user experience of Google's Two-Step Authentication, allowing users to authenticate through call, text messages (SMS) or by using a token generator app like Google Authenticator.

Contents:

CHAPTER 1

Requirements

1.1 Django

Django version 1.4 and above are supported. The minimal version of Django is 1.4.2 as it includes the `six` compatibility layer used for transitioning to Python 3.

1.2 Python

The following Python versions are supported: 2.6, 2.7, 3.2, 3.3 and 3.4. As support for older Django versions is dropped, the minimum version might be raised. See also What Python version can I use with Django?.

1.3 django-otp

This project is used for generating one-time passwords. Version 0.2 and above are supported.

CHAPTER 2

Installation

You can install from **PyPI_** using `pip` to install `django-two-factor-auth` and its dependencies:

```
``pip install django-two-factor-auth``
```

Add the following apps to the `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'django_otp',
    'django_otp.plugins.otp_static',
    'django_otp.plugins.otp_totp',
    'two_factor',
)
```

Add the `django-otp` middleware to your `MIDDLEWARE_CLASSES`. Make sure it comes after `AuthenticationMiddleware`:

```
MIDDLEWARE_CLASSES = (
    ...
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django_otp.middleware.OTPMiddleware',
    ...
)
```

Point to the new login pages in your settings:

```
from django.core.urlresolvers import reverse_lazy

LOGIN_URL = reverse_lazy('two_factor:login')

# this one is optional
LOGIN_REDIRECT_URL = reverse_lazy('two_factor:profile')
```

Add the routes to your url configuration:

```
urlpatterns = patterns(
    '',
    url(r'', include('two_factor.urls', 'two_factor')),
    ...
)
```

Warning: Be sure to remove any other login routes, otherwise the two-factor authentication might be circumvented. The admin interface should be automatically patched to use the new login method.

2.1 Yubikey

In order to support Yubikeys, you have to install a plugin for *django-otp*:

```
pip install django-otp-yubikey
```

Add the following app to the `INSTALLED_APPS`:

```
INSTALLED_APPS = (
    ...
    'otp_yubikey',
)
```

This plugin also requires adding a validation service, through wich YubiKeys will be verified. Normally, you'd use the YubiCloud for this. In the Django admin, navigate to `YubiKey validation services` and add an item. Django Two-Factor Authentication will identify the validation service with the name `default`. The other fields can be left empty, but you might want to consider requesting an API ID along with API key and using SSL for communicating with YubiCloud.

You could also do this using this snippet:

```
manage.py shell
>>> from otp_yubikey.models import ValidationService
>>> ValidationService.objects.create(name='default', use_ssl=True,
...     param_sl='', param_timeout='')
<ValidationService: default>
```

CHAPTER 3

Configuration

3.1 General Settings

TWO_FACTOR_PATCH_ADMIN (default: True) Whether the Django admin is patched to use the default login view.

> **Warning:** The admin currently does not enforce one-time passwords being set for admin users.

TWO_FACTOR_CALL_GATEWAY (default: None) Which gateway to use for making phone calls. Should be set to a module or object providing a `make_call` method. Currently two gateways are bundled:

- `two_factor.gateways.twilio.gateway.Twilio` for making real phone calls using Twilio.
- `two_factor.gateways.fake.Fake` for development, recording tokens to the default logger.

TWO_FACTOR_SMS_GATEWAY (default: None) Which gateway to use for sending text messages. Should be set to a module or object providing a `send_sms` method. Currently two gateways are bundled:

- `two_factor.gateways.twilio.gateway.Twilio` for sending real text messages using Twilio.
- `two_factor.gateways.fake.Fake` for development, recording tokens to the default logger.

LOGIN_URL Should point to the login view provided by this application. This login view handles password authentication followed by a one-time password exchange if enabled for that account.

See also LOGIN_URL.

LOGIN_REDIRECT_URL This application provides a basic page for managing one's account. This view is entirely optional and could be implemented in a custom view.

See also LOGIN_REDIRECT_URL.

TWO_FACTOR_QR_FACTORY The default generator for the QR code images is set to SVG. This does not require any further dependencies, however it does not work on IE8 and below. If you have PIL, Pillow or pyimaging installed you may wish to use PNG images instead.

- `qrcode.image.pil.PilImage` may be used for PIL/Pillow

- `qrcode.image.pure.PymagingImage` may be used for pyimaging

For more QR factories that are available see python-qrcode.

3.2 Twilio Gateway

To use the Twilio gateway, you need first to install the Twilio client:

```
pip install twilio
```

Next, you also need to include the Twilio urlpatterns. As these urlpatterns are all defined using a single Django namespace, these should be joined with the base urlpatterns, like so:

```
# urls.py
from two_factor.urls import urlpatterns as tf_urls
from two_factor.gateways.twilio.urls import urlpatterns as tf_twilio_urls

urlpatterns = patterns('',
    url(r'', include(tf_urls + tf_twilio_urls, 'two_factor')),
)
```

3.3 Fake Gateway

CHAPTER 4

Implementing

Users can opt-in to enhanced security by enabling two-factor authentication. There is currently no enforcement of a policy, it is entirely optional. However, you could override this behaviour to enforce a custom policy.

4.1 Limiting access to certain views

For increased security views can be limited to two-factor-enabled users. This allows you to secure certain parts of the website. Doing so requires a decorator, class mixin or a custom inspection of a user's session.

4.1.1 Decorator

You can use django-otp's built-in *otp_required()* decorator to limit access to two-factor-enabled users:

```
from django_otp.decorators import otp_required

@otp_required
def my_view(request):
    pass
```

4.1.2 Mixin

The mixin `OTPRequiredMixin` can be used to limit access to class-based views (CBVs):

```
class ExampleSecretView(OTPRequiredMixin, TemplateView):
    template_name = 'secret.html'
```

4.1.3 Custom

The method `is_verified()` is added through django-otp's `OTPMiddleware` which can be used to check if the user was logged in using two-factor authentication:

```
def my_view(request):
    if request.user.is_verified():
        # user logged in using two-factor
        pass
    else:
        # user not logged in using two-factor
        pass
```

4.2 Enforcing two-factor

Forcing users to enable two-factor authentication is not implemented. However, you could create your own custom policy.

4.3 Admin Site

By default the admin login is patched to use the login views provided by this application. Patching the admin is required as users would otherwise be able to circumvent OTP verification. See also `TWO_FACTOR_PATCH_ADMIN`. Be aware that certain packages include their custom login views, for example *django.contrib.admindocs*. When using said packages, OTP verification can be circumvented. Thus however the normal admin login view is patched, OTP might not always be enforced on the admin views. See the next paragraph on how to do this.

In order to only allow verified users (enforce OTP) to access the admin pages, you have to use a custom admin site. You can either use `AdminSiteOTPRequired` or `AdminSiteOTPRequiredMixin`. See also the Django documentation on Hooking AdminSite instances into your URLconf.

4.4 Signals

When a user was successfully verified using a OTP, the signal *`user_verified`* is sent. The signal includes the user, the device used and the request itself. You can use this signal for example to warn a user when one of his backup tokens was used:

```
from django.contrib.sites.models import get_current_site
from django.dispatch import receiver
from two_factor.signals import user_verified


@receiver(user_verified)
def test_receiver(request, user, device, **kwargs):
    current_site = get_current_site(request)
    if device.name == 'backup':
        message = 'Hi %(username)s,\n\n'\
                  'You\'ve verified yourself using a backup device '\
                  'on %(site_name)s. If this wasn\'t you, your '\
                  'account might have been compromised. You need to '\
                  'change your password at once, check your backup '\
```

(continues on next page)

(continued from previous page)

```
                'phone numbers and generate new backup tokens.'\
                % {'username': user.get_username(),
                   'site_name': current_site.name}
        user.email_user(subject='Backup token used', message=message)
```

CHAPTER 5

Management Commands

5.1 Status

5.2 Disable

CHAPTER 6

Class Reference

6.1 Admin Site

6.2 Decorators

`django_otp.decorators.`**`otp_required`**(*view=None*, *redirect_field_name='next'*, *login_url=None*, *if_configured=False*)

Similar to `login_required()`, but requires the user to be verified. By default, this redirects users to **:setting:'OTP_LOGIN_URL'**.

Parameters **`if_configured`** (*bool*) – If `True`, an authenticated user with no confirmed OTP devices will be allowed. Default is `False`.

6.3 Models

6.4 Middleware

6.5 Signals

`two_factor.signals.`**`user_verified`**

Sent when a user is verified against a OTP device. Provides the following arguments:

`sender` The class sending the signal (`'two_factor.views.core'`).

`user` The user that was verified.

`device` The OTP device that was used.

`request` The `HttpRequest` in which the user was verified.

6.6 Template Tags

6.7 Views

6.8 View Mixins

CHAPTER 7

Release Notes

7.1 1.0.0

- Fixed #68 – Cannot generate QR code for unicode characters
- Fixed #66 – Admin not secure when using admindocs (clarified documentation)

7.2 1.0.0-beta3

- Fixed #62 – Don't leak sensitive post parameters
- Fixed #63 – Login wizard should handle changing passwords

7.3 1.0.0-beta2

- Fixed #60 – Always cast the token to an int before verification

7.4 1.0.0-beta1

- Support for Django 1.7
- New translations: German, Spanish, French, Swedish and Portuguese (Brazil)
- #39 – Added support for custom user model (Django 1.5+)
- Added management commands
- Added support for YubiKeys
- Fire signal when user is verified

- #44 – Don't require re-login after setup
- #49 – Advise to add backup devices after setup
- #52 – Add URL encoding to otpauth URL
- #54 – Mitigate voicemail hack
- #55 – Use two_factor:login instead of LOGIN_URL

7.5 0.5.0

- #32 – Make the auth method label capitalization more consistent
- #31 – Set an error code for phone_number_validator
- #30 – Don't transmit token seed through GET parameters
- #29 – Generate QR codes locally
- #27 – South migrations to support custom user model

7.6 0.4.0

- Fixed #26 – Twilio libraries are required

7.7 0.3.1

- Fixed #25 – Back-up tokens cannot be used for login

7.8 0.3.0

- #18 – Optionally enforce OTP for admin views.
- New translation: Simplified Chinese.

7.9 0.2.3

- Two new translations: Hebrew and Arabic.

7.10 0.2.2

- Allow changing Twilio call language.

7.11 0.2.1

- Allow overriding instructions in the template.
- Allow customization of the redirect query parameter.
- Faster backup token generating.

7.12 0.2.0

This is a major upgrade, as the package has been rewritten completely. Upgrade to this version with care and make backups of your database before running the South migrations. See installation instructions for installing the new version; update your template customizations and run the database migrations.

I would love to hear your feedback on this application. If you run into problems, please file an issue on GitHub, or contribute to the project by forking the repository and sending some pull requests.

This application is currently translated into English, Dutch, Hebrew, Arabic, German, Chinese and Spanish. Please contribute your own language using Transifex.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

t

Index