Django Two-Factor Authentication Documentation

Release 1.13.2

Bouke Haarsma

Jan 18, 2022

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

Modern Django versions are supported. Currently this list includes Django 2.2, 3.0, 3.1, 3.2, and 4.0.

1.2 Python

The following Python versions are supported: 3.5, 3.6, 3.7, 3.8 and 3.9 with a limit to what Django itself supports. 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 0.8.x and above are supported.

1.4 django-formtools

Formerly known as `django.contrib.formtools`, it has been separated from Django 1.8 into a new package. Version 1.0 is 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
```

This project uses `django-phonenumber-field` which requires either `phonenumbers` or `phonenumberslite` to be installed. Either manually install a supported version using `pip` or install `django-two-factor-auth` with the extras specified as in the below examples:

```
$ pip install django-two-factor-auth[phonenumbers]

OR

$ pip install django-two-factor-auth[phonenumberslite]
```

2.1 Setup

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`. Make sure it comes after `AuthenticationMiddleware`:

```
MIDDLEWARE = (
    ...
```

(continues on next page)

(continued from previous page)

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

Point to the new login pages in your `settings.py`:

```
LOGIN_URL = 'two_factor:login'

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

Add the routes to your project url configuration:

```
from two_factor.urls import urlpatterns as tf_urls
urlpatterns = [
   path('', include(tf_urls)),
    ...
]
```

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.

2.2 Yubikey Setup

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 which 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 Django's *manage.py shell*:

```
$ python 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 as described in setup. This login view handles password authentication followed by a one-time password exchange if enabled for that account. This can be a URL path or URL name as defined in the Django documentation.

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. This can be a URL path or URL name as defined in the Django documentation.

See also LOGIN_REDIRECT_URL.

LOGOUT_REDIRECT_URL Should point to a view that the user is redirected to after logging out. It was added in Django 1.10, and also adapted by this application. This can be a URL path or URL name as defined in the Django documentation.

See also LOGOUT_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.

TWO_FACTOR_TOTP_DIGITS (default: `6`) The number of digits to use for TOTP tokens, can be set to 6 or 8. This setting will be used for tokens delivered by phone call or text message and newly configured token generators. Existing token generator devices will not be affected.

Warning: The Google Authenticator app does not support 8 digit codes (see the upstream ticket). Don't set this option to 8 unless all of your users use a 8 digit compatible token generator app.

TWO_FACTOR_LOGIN_TIMEOUT (default `600`) The number of seconds between a user successfully passing the "authentication" step (usually by entering a valid username and password) and them having to restart the login flow and re-authenticate. This ensures that users can't sit indefinitely in a state of having entered their password successfully but not having passed two factor authentication. Set to `0` to disable.

PHONENUMBER_DEFAULT_REGION (default: `None`) The default region for parsing phone numbers. If your application's primary audience is a certain country, setting the region to that country allows entering phone numbers without that country's country code.

TWO_FACTOR_PHONE_THROTTLE_FACTOR (default: `1`) This controls the rate of throttling. The sequence of 1, 2, 4, 8… seconds is multiplied by this factor to define the delay imposed after 1, 2, 3, 4… successive failures. Set to `0` to disable throttling completely.

3.2 Twilio Gateway

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

```
$ pip install twilio
```

Next, add additional urls to your config:

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

urlpatterns = [
    path('', include(tf_twilio_urls)),
    ...
]
```

Additionally, you need to enable the `ThreadLocals` middleware:

```
MIDDLEWARE = (
    ...

    # Always include for two-factor auth
    'django_otp.middleware.OTPMiddleware',

    # Include for Twilio gateway
    'two_factor.middleware.threadlocals.ThreadLocals',
)
```

class `two_factor.gateways.twilio.gateway.`**`Twilio`**

Gateway for sending text messages and making phone calls using Twilio.

All you need is your Twilio Account SID and Token, as shown in your Twilio account dashboard.

`TWILIO_ACCOUNT_SID` Should be set to your account's SID.

`TWILIO_AUTH_TOKEN` Should be set to your account's authorization token.

`TWILIO_CALLER_ID` Should be set to a verified phone number. Twilio differentiates between numbers verified for making phone calls and sending text messages.

`TWILIO_MESSAGING_SERVICE_SID` Can be set to a Twilio Messaging Service for SMS. This service can wrap multiple phone numbers and choose one depending on the destination country. When left empty the `TWILIO_CALLER_ID` will be used as sender ID.

3.3 Fake Gateway

class `two_factor.gateways.fake.`**`Fake`**

Prints the tokens to the logger. You will have to set the message level of the `two_factor` logger to `INFO` for them to appear in the console. Useful for local development. You should configure your logging like this:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'two_factor': {
            'handlers': ['console'],
            'level': 'INFO',
        }
    }
}
```

3.4 Remember Browser

During a successful login with a token, the user may choose to remember this browser. If the same user logs in again on the same browser, a token will not be requested, as the browser serves as a second factor.

The option to remember a browser is deactived by default. Set *TWO_FACTOR_REMEMBER_COOKIE_AGE* to activate.

The browser will be remembered as long as:

- the cookie that authorizes the browser did not expire,
- the user did not reset the password, and
- the device initially used to authorize the browser is still valid.

The browser is remembered by setting a signed 'remember cookie'.

In order to invalidate remebered browsers after password resets, the package relies on the *password* field of the *User* model. Please consider this in case you do not use the *password* field e.g. [django-auth-ldap](https://github.com/django-auth-ldap/django-auth-ldap)

`TWO_FACTOR_REMEMBER_COOKIE_AGE` Age in seconds to remember the browser. The remember cookie will expire after the given time interval and the server will not accept this cookie to remember this browser, user, and device any longer.

If this is set to a positive *int* the user is presented the option to remember the browser when entering the token. If the age is *None*, the user must authenticate with a token option during each login, if a device is setup.

Default: *None*

`TWO_FACTOR_REMEMBER_COOKIE_PREFIX` Prefix of the remember cookie. It prefixes a uuid4 to allow multiple remember cookies on the same browser for multiple users.

Default: *'remember-cookie_'*

`TWO_FACTOR_REMEMBER_COOKIE_DOMAIN` The domain to be used when setting the remember cookie.

Only relevant if *TWO_FACTOR_REMEMBER_COOKIE_AGE* is not *None*.

Default: *None*

`TWO_FACTOR_REMEMBER_COOKIE_PATH` The path of the remember cookie.

Only relevant if *TWO_FACTOR_REMEMBER_COOKIE_AGE* is not *None*.

Default: *'/'*

`TWO_FACTOR_REMEMBER_COOKIE_SECURE` Whether the remember cookie should be secure (https:// only).

Only relevant if *TWO_FACTOR_REMEMBER_COOKIE_AGE* is not *None*.

Default: *False*

`TWO_FACTOR_REMEMBER_COOKIE_HTTPONLY` Whether to use the non-RFC standard httpOnly flag (IE, FF3+, others)

Only relevant if *TWO_FACTOR_REMEMBER_COOKIE_AGE* is not *None*.

Default: *True*

`TWO_FACTOR_REMEMBER_COOKIE_SAMESITE` Whether to set the flag restricting cookie leaks on cross-site requests. This can be 'Lax', 'Strict', or None to disable the flag.

Only relevant if *TWO_FACTOR_REMEMBER_COOKIE_AGE* is not *None*.

Default: *'Lax'*

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.

If you want to enforce two factor authentication in the admin and use the default admin site (e.g. because 3rd party packages register to `django.contrib.admin.site`) you can monkey patch the default `AdminSite` with this. In your `urls.py`:

```
from django.contrib import admin
from two_factor.admin import AdminSiteOTPRequired

admin.site.__class__ = AdminSiteOTPRequired

urlpatterns = [
    path('admin/', admin.site.urls),
    ...
]
```

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.shortcuts 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 '\
                  '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

class two_factor.management.commands.two_factor_status.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Command to check two-factor authentication status for certain users.

The command accepts any number of usernames, and will list if OTP is enabled or disabled for those users.

Example usage:

```
manage.py two_factor_status bouke steve
bouke: enabled
steve: disabled
```

5.2 Disable

class two_factor.management.commands.two_factor_disable.**Command**(*stdout=None*, *stderr=None*, *no_color=False*, *force_color=False*)

Command for disabling two-factor authentication for certain users.

The command accepts any number of usernames, and will remove all OTP devices for those users.

Example usage:

```
manage.py two_factor_disable bouke steve
```

CHAPTER 6

Class Reference

6.1 Admin Site

class `two_factor.admin.`**`AdminSiteOTPRequired`**(*name='admin'*)
AdminSite enforcing OTP verified staff users.

class `two_factor.admin.`**`AdminSiteOTPRequiredMixin`**
Mixin for enforcing OTP verified staff users.

Custom admin views should either be wrapped using `admin_view()` or use `has_permission()` in order to secure those views.

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 `OTP_LOGIN_URL`.

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

`two_factor.views.utils.`**`class_view_decorator`**(*function_decorator*)
Converts a function based decorator into a class based decorator usable on class based Views.

Can't subclass the *View* as it breaks inheritance (super in particular), so we monkey-patch instead.

From: http://stackoverflow.com/a/8429311/58107

6.3 Models

class `two_factor.models.`**`PhoneDevice`**(**args*, ***kwargs*)
Model with phone number and token seed linked to a user.

class `django_otp.plugins.otp_static.models.`**`StaticDevice`**(**args*, ***kwargs*)
A static `Device` simply consists of random tokens shared by the database and the user.

These are frequently used as emergency tokens in case a user's normal device is lost or unavailable. They can be consumed in any order; each token will be removed from the database as soon as it is used.

This model has no fields of its own, but serves as a container for *`StaticToken`* objects.

`token_set`
The RelatedManager for our tokens.

class `django_otp.plugins.otp_static.models.`**`StaticToken`**(**args*, ***kwargs*)
A single token belonging to a *`StaticDevice`*.

`device`
ForeignKey: A foreign key to *`StaticDevice`*.

`token`
CharField: A random string up to 16 characters.

class `django_otp.plugins.otp_totp.models.`**`TOTPDevice`**(**args*, ***kwargs*)
A generic TOTP `Device`. The model fields mostly correspond to the arguments to `django_otp.oath.totp()`. They all have sensible defaults, including the key, which is randomly generated.

`key`
CharField: A hex-encoded secret key of up to 40 bytes. (Default: 20 random bytes)

`step`
PositiveSmallIntegerField: The time step in seconds. (Default: 30)

`t0`
BigIntegerField: The Unix time at which to begin counting steps. (Default: 0)

`digits`
PositiveSmallIntegerField: The number of digits to expect in a token (6 or 8). (Default: 6)

`tolerance`
PositiveSmallIntegerField: The number of time steps in the past or future to allow. For example, if this is 1, we'll accept any of three tokens: the current one, the previous one, and the next one. (Default: 1)

`drift`
SmallIntegerField: The number of time steps the prover is known to deviate from our clock. If `OTP_TOTP_SYNC` is `True`, we'll update this any time we match a token that is not the current one. (Default: 0)

`last_t`
BigIntegerField: The time step of the last verified token. To avoid verifying the same token twice, this will be updated on each successful verification. Only tokens at a higher time step will be verified subsequently. (Default: -1)

6.4 Middleware

class `django_otp.middleware.`**`OTPMiddleware`**(*get_response=None*)
This must be installed after `AuthenticationMiddleware` and performs an analogous function. Just

as AuthenticationMiddleware populates `request.user` based on session data, OTPMiddleware populates `request.user.otp_device` to the `Device` object that has verified the user, or `None` if the user has not been verified. As a convenience, this also installs `user.is_verified()`, which returns `True` if `user.otp_device` is not `None`.

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

`two_factor.templatetags.two_factor.`**`device_action`**`(`*device*`)`

Generates an actionable text for a `PhoneDevice`.

Examples:

- Send text message to *+31 * ******58*
- Call number *+31 * ******58*

`two_factor.templatetags.two_factor.`**`format_phone_number`**`(`*number*`)`

Formats a phone number in international notation. :param number: str or phonenumber object :return: str

`two_factor.templatetags.two_factor.`**`mask_phone_number`**`(`*number*`)`

Masks a phone number, only first 3 and last 2 digits visible.

Examples:

- *+31 * ******58*

Parameters **`number`** – str or phonenumber object

Returns str

6.7 Views

`class` `two_factor.views.`**`LoginView`**`(`***kwargs*`)`

View for handling the login process, including OTP verification.

The login process is composed like a wizard. The first step asks for the user's credentials. If the credentials are correct, the wizard proceeds to the OTP verification step. If the user has a default OTP device configured, that device is asked to generate a token (send sms / call phone) and the user is asked to provide the generated token. The backup devices are also listed, allowing the user to select a backup device for verification.

class `two_factor.views.`**`SetupView`**(***kwargs*)
View for handling OTP setup using a wizard.

The first step of the wizard shows an introduction text, explaining how OTP works and why it should be enabled. The user has to select the verification method (generator / call / sms) in the second step. Depending on the method selected, the third step configures the device. For the generator method, a QR code is shown which can be scanned using a mobile phone app and the user is asked to provide a generated token. For call and sms methods, the user provides the phone number which is then validated in the final step.

class `two_factor.views.`**`SetupCompleteView`**(***kwargs*)
View congratulation the user when OTP setup has completed.

class `two_factor.views.`**`BackupTokensView`**(***kwargs*)
View for listing and generating backup tokens.

A user can generate a number of static backup tokens. When the user loses its phone, these backup tokens can be used for verification. These backup tokens should be stored in a safe location; either in a safe or underneath a pillow ;-).

class `two_factor.views.`**`PhoneSetupView`**(***kwargs*)
View for configuring a phone number for receiving tokens.

A user can have multiple backup *`PhoneDevice`* for receiving OTP tokens. If the primary phone number is not available, as the battery might have drained or the phone is lost, these backup phone numbers can be used for verification.

class `two_factor.views.`**`PhoneDeleteView`**(***kwargs*)
View for removing a phone number used for verification.

class `two_factor.views.`**`ProfileView`**(***kwargs*)
View used by users for managing two-factor configuration.

This view shows whether two-factor has been configured for the user's account. If two-factor is enabled, it also lists the primary verification method and backup verification methods.

class `two_factor.views.`**`DisableView`**(***kwargs*)
View for disabling two-factor for a user's account.

6.8 View Mixins

class `two_factor.views.mixins.`**`OTPRequiredMixin`**
View mixin which verifies that the user logged in using OTP.

Note: This mixin should be the left-most base class.

`get_login_url`()
Returns login url to redirect to.

`get_verification_url`()
Returns verification url to redirect to.

`login_url = None`
If *`raise_anonymous`* is set to *False*, this defines where the user will be redirected to. Defaults to `two_factor:login`.

`raise_anonymous = False`
Whether to raise PermissionDenied if the user isn't logged in.

`raise_unverified = False`
Whether to raise PermissionDenied if the user isn't verified.

`redirect_field_name = 'next'`
URL query name to use for providing the destination URL.

`verification_url = None`
If *`raise_unverified`* is set to *False*, this defines where the user will be redirected to. If set to `None`, an explanation will be shown to the user on why access was denied.

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, Spanish, French, Swedish, Portuguese (Brazil), Polish, Italian, Hungarian, Finnish and Danish. You can contribute your own language using Transifex.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index