# pwned-passwords-django Documentation

*Release 1.3.1*

**James Bennett**

**Sep 18, 2018**

# Contents

pwned-passwords-django provides helpers for working with the Pwned Passwords database of Have I Been Pwned in Django powered sites. Pwned Passwords is an extremely large database of passwords known to have been compromised through data breaches, and is useful as a tool for rejecting common or weak passwords.

There are three main components to this:

- *A password validator* which checks the Pwned Passwords database.
- *A middleware* which automatically checks certain request payloads against the Pwned Passwords database.
- *Code providing direct access* to the Pwned Passwords database.

All three use a secure, anonymized API which never transmits the password or its hash to any third party. To learn more, see *the FAQ*.

CHAPTER 1

# Documentation contents

## 1.1 Installation

pwned-passwords-django 1.3.1 supports Django 1.11, 2.0 and 2.1, on the following Python versions:

- Django 1.11 supports Python 2.7, 3.4, 3.5 and 3.6.
- Django 2.0 supports Python 3.4, 3.5, 3.6 and 3.7.
- Django 2.1 supports Python 3.5, 3.6 and 3.7.

To install pwned-passwords-django, run:

```
pip install pwned-passwords-django
```

This will use `pip`, the standard Python package-installation tool. If you are using a supported version of Python, your installation of Python should have come with `pip` bundled. To make sure you have the latest version of `pip`, run:

```
python -m ensurepip --upgrade
```

If this fails, instructions are available for how to obtain and manually install pip.

If you don't already have a supported version of Django installed, using `pip` to install pwned-passwords-django will also install the latest supported version of Django.

## 1.2 Configuration and use

You may need to modify certain Django settings, depending on how you'd like to use `pwned-passwords-django`. See the following documentation for notes on additional configuration:

- Using *the password validator*
- Using *the automatic password-checking middleware*
- Using *the Pwned Passwords API directly*

## 1.3 Using the password validator

**class** pwned_passwords_django.validators.**PwnedPasswordsValidator**

Django's auth system (located in `django.contrib.auth`) includes a configurable password-validation framework with several built-in validators; pwned-passwords-django provides an additional validator which checks the Pwned Passwords database. To enable it, set your `AUTH_PASSWORD_VALIDATORS` setting to include `pwned_passwords_django.validators.PwnedPasswordsValidator`, like so:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'pwned_passwords_django.validators.PwnedPasswordsValidator',
    },
]
```

This will cause most high-level password-setting operations to check the Pwned Passwords database, and reject any password found there. Specifically, password validators are applied:

- Whenever a user changes or resets their password with Django's built-in auth views.
- Whenever a new user is created via Django's built-in `UserCreationForm`.
- Whenever the `createsuperuser` or `changepassword` management commands are used.
- Whenever an instance of the built-in `User` model is saved after the instance's `set_password()` method has been called.

Keep in mind that validation is **not** run when code sets or changes a user's password in other ways. If you manipulate user passwords through means other than the high-level APIs listed above, you'll need to manually check passwords.

> **Warning: API failures**
>
> pwned-passwords-django needs to communicate with the Pwned Passwords API in order to check passwords. If Pwned Passwords is down or timing out (the default connection timeout is 1 second), this validator will fall back to using Django's CommonPasswordValidator, which uses a smaller, locally-stored list of common passwords. Whenever this happens, a message of level `logging.WARNING` will appear in your logs, indicating what type of failure was encountered in talking to the Pwned Passwords API.

## 1.4 Customizing the validator's messages

To change the error or help messages shown to the user, you can pass `OPTIONS` when adding the validator to your settings:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'pwned_passwords_django.validators.PwnedPasswordsValidator',
        'OPTIONS': {
            'error_message': 'That password was pwned',
            'help_message': 'Your password can\'t be a commonly used password.',
        }
    },
]
```

The number of times the password has appeared in a breach can also be included in the error message, including a plural form:

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'pwned_passwords_django.validators.PwnedPasswordsValidator',
        'OPTIONS': {
            'error_message': (
               'Pwned %(amount)d time',
               'Pwned %(amount)d times',
            )
        }
    },
]
```

## 1.5 Using the middleware

**class** pwned_passwords_django.middleware.**PwnedPasswordsMiddleware**

To help catch situations where a potentially-compromised password is used in ways Django's password validators won't catch, pwned-passwords-django also provides a middleware which monitors every incoming HTTP request for payloads which appear to contain passwords, and checks them against Pwned Passwords.

To enable the middleware, add `pwned_passwords_django.middleware.PwnedPasswordsMiddleware` to your `MIDDLEWARE` setting. This will add a new attribute – `pwned_passwords` – to each `HttpRequest` object. The `request.pwned_passwords` attribute will be a dictionary.

> **Warning: Middleware order**
>
> The order of middleware classes in the Django `MIDDLEWARE` setting can be sensitive. In particular, any middlewares which affect file upload handlers *must* be listed above middlewares which inspect `request.POST`. Since this middleware has to inspect `request.POST` for likely passwords, it must be listed after any middlewares which might change upload handlers. If you're unsure what this means, just put this middleware at the bottom of your `MIDDLEWARE` list.

The `request.pwned_passwords` dictionary will be *empty* if any of the following is true:

- The request method is not `POST`.
- The request method is `POST`, but the payload does not appear to contain a password.
- The request method is `POST`, and the payload appears to contain a password, but the password is not listed as compromised in Pwned Passwords.

If the request method is `POST`, and the payload appears to contain a password, and the password is listed in Pwned Passwords, then `request.pwned_passwords` will contain a key corresponding to the key in `request.POST` which appeared to contain a password, and the value associated with that key will be the number of times that password appears in the Pwned Passwords database.

For example, if `request.POST` contains a key named `password`, and the value associated with it appears 42 times in the Pwned Passwords database, `request.pwned_passwords` will be `{'password': 42}`.

> **Warning: API failures**
>
> pwned-passwords-django needs to communicate with the Pwned Passwords API in order to check passwords. If Pwned Passwords is down or timing out (the default connection timeout is 1 second), this

middleware will not re-try the check or fall back to an alternate mechanism; it will leave `request.pwned_passwords` empty. Whenever this happens, a message of level `logging.WARNING` will appear in your logs, indicating what type of failure was encountered in talking to the Pwned Passwords API.

Here's an example of how you might use Django's message framework to indicate to a user that they've just submitted a password that appears to be compromised:

```
from django.contrib import messages


def some_view(request):
    if request.method == 'POST' and request.pwned_passwords:
        messages.warning(
            request,
            'You just entered a password which appears to be compromised!'
        )
```

pwned-passwords-django uses a regular expression to guess which items in `request.POST` are likely to be passwords. By default, it matches on any key in `request.POST` containing `'PASS'` (case-insensitive), which catches input names like `'password'`, `'passphrase'`, and so on. If you use something significantly different than this for a password input name, specify it – as a raw string, *not* as a compiled regex object! – in the setting *PWNED_PASSWORDS_REGEX* to tell the middleware what to look for.

## 1.6 Using the Pwned Passwords API directly

If the validator and middleware do not cover your needs, you can also directly check a password against Pwned Passwords.

`pwned_passwords_django.api.`**`pwned_password`**(*password*)

Given a password, checks it against the Pwned Passwords database and returns a count of the number of times that password occurs in the database.

The password to check **must** be a Unicode string (the type `str` on Python 3, `unicode` on Python 2). Passing a bytes object (`bytes` on Python 3, `str` on Python 2) will raise `TypeError`.

**Warning: API failures**

pwned-passwords-django needs to communicate with the Pwned Passwords API in order to check passwords. If Pwned Passwords is down or timing out (the default connection timeout is 1 second), this function will not re-try the check or fall back to an alternate mechanism; it will return `None`. Whenever this happens, a message of level `logging.WARNING` will appear in your logs, indicating what type of failure was encountered in talking to the Pwned Passwords API.

**Parameters** **`password`** (*Unicode string*) – The password to check.

**Return type** `int` or `None`

## 1.7 Custom settings

Two optional custom Django settings can be used to customize the behavior of pwned-passwords-django.

`django.conf.settings.`**`PWNED_PASSWORDS_API_TIMEOUT`**
: A `float` indicating, in seconds, how long to wait for a response from the Pwned Passwords API before giving up.

  Defaults to `1.0` (1 second) if not set.

`django.conf.settings.`**`PWNED_PASSWORDS_REGEX`**
: A `str` containing a regular expression to use when *`PwnedPasswordsMiddleware`* is scanning HTTP `POST` payloads for possible passwords. Will be checked case-insensitively.

  Defaults to `r'PASS'` (thus matching `'password'`, `'passphrase'`, etc.) if not set.

## 1.8 Frequently asked questions

The following notes answer some common questions, and may be useful to you when using pwned-passwords-django.

### 1.8.1 What versions of Django and Python are supported?

As of pwned-passwords-django 1.3.1, Django 1.11, 2.0 and 2.1 are supported, on Python 2.7, (Django 1.11 only), 3.4 (Django 1.11 and 2.0 only), 3.5, 3.6 and 3.7 (Django 2.0 and 2.1 only).

### 1.8.2 Should I use the validator, the middleware, or the API directly?

It's probably best to enable both the validator and the middleware. *The validator* by itself can catch many attempts to set a user's password to a known-compromised value, but cannot catch cases where a user already has a compromised password and is continuing to use it. *The middleware* can catch that case, provided you're checking the `request.pwned_passwords` attribute in your view code.

Using *the direct API* should only be necessary in rare cases where neither the validator nor the middleware is sufficient.

### 1.8.3 I'm getting timeouts from the Pwned Passwords API. What can I do?

By default, pwned-passwords-django makes requests to the Pwned Passwords API with a timeout of one second. You can change this by specifying the Django setting *`PWNED_PASSWORDS_API_TIMEOUT`* and setting it to a float indicating your preferred timeout; for example, to have a timeout of one and a half seconds, you'd set:

```
PWNED_PASSWORDS_API_TIMEOUT = 1.5
```

### 1.8.4 How can this be secure? It's sending passwords to some random site!

It's *not* actually sending passwords to any other site, and that's the magic.

You can read about this in the post announcing the launch of version 2 of Pwned Passwords, but the summary of how it works is:

1. pwned-passwords-django hashes the password, and sends only the first five digits of the hexadecimal digest of the hash to Pwned Passwords.
2. Pwned Passwords responds with a list of hash suffixes (all the digits of the hash *except* the first five) for every entry in its database matching the submitted five-digit prefix.
3. pwned-passwords-django checks that list to see if the remainder of the password hash is present, and if so treats the password as compromised.

This means that neither the password, nor the full hash of the password, is ever sent to any third-party site or service by pwned-passwords-django.

**Warning: You can still accidentally disclose passwords!**

pwned-passwords-django uses an API that never discloses the password or its hash, but that doesn't mean the rest of your code or third-party libraries won't.

You should take care to use Django's tools for filtering sensitive information from tracebacks and error reports to ensure that your logging and monitoring systems don't accidentally log passwords. You should also be extremely conservative about allowing third-party JavaScript to run on your site, and periodically audit all JavaScript you use; remember that JavaScript can access anything your users enter on your site, and potentially do malicious things with that information.

### 1.8.5 How do I run the tests?

pwned-passwords-django's tests are run using tox, but typical installation of pwned-passwords-django (via `pip install pwned-passwords-django`) will not install the tests.

To run the tests, download the source (`.tar.gz`) distribution of pwned-passwords-django 1.3.1 from its page on the Python Package Index, unpack it (`tar zxvf pwned-passwords-django-|version|.tar.gz` on most Unix-like operating systems), and in the unpacked directory run `tox`.

Note that you will need to have `tox` installed already (`pip install tox`), and to run the full test matrix you will need to have each supported version of Python available. To run only the tests for a specific Python version and Django version, you can invoke `tox` with the `-e` flag. For example, to run tests for Python 3.6 and Django 2.0: `tox -e py36-django20`.

### 1.8.6 How am I allowed to use this code?

The pwned-passwords-django module is distributed under a three-clause BSD license. This is an open-source license which grants you broad freedom to use, redistribute, modify and distribute modified versions of pwned-passwords-django. For details, see the file `LICENSE` in the source distribution of pwned-passwords-django.

### 1.8.7 I found a bug or want to make an improvement!

The canonical development repository for pwned-passwords-django is online at <https://github.com/ubernostrum/pwned-passwords-django>. Issues and pull requests can both be filed there.

## 1.9 Changelog

This document lists changes between released versions of pwned-passwords-django.

### 1.9.1 1.3.1 – released 2018-09-18

Released to include documentation updates which were inadvertently left out of the 1.3 package.

### 1.9.2 1.3 – released 2018-09-18

No new features. No bug fixes. Released only to add explicit markers of Python 3.7 and Django 2.1 compatibility.

### 1.9.3 1.2.1 – released 2018-06-18

Released to correct the date of the 1.2 release listed in this changelog document. No other changes.

### 1.9.4 1.2 – released 2018-06-18

#### New features:

- Password-validator error messages are now *customizable*.
- The request-timeout value for contacting the Pwned Passwords API defaults to one second, and is customizable via the setting *PWNED_PASSWORDS_API_TIMEOUT*.
- When a request to the Pwned Passwords API times out, or encounters an error, it logs the problem with a message of level `logging.WARNING`. The *PwnedPasswordsValidator* will fall back to Django's CommonPasswordValidator, which has a smaller list of common passwords. The *PwnedPasswordsMiddleware* does not have a fallback behavior; *pwned_password()* will return `None` to indicate the error case.

#### Bugs fixed:

N/A

#### Other changes:

- *pwned_password()* will now raise `TypeError` if its argument is not a Unicode string (the type `unicode` on Python 2, `str` on Python 3). This is debatably backwards-incompatible; `pwned_password()` encodes its argument to UTF-8 bytes, which will raise `AttributeError` if attempted on a `bytes` object in Python 3. As a result, all supported environments other than Python 2.7/Django 1.11 would already raise `AttributeError` (due to `bytes` objects lacking the `encode()` method) in both 1.0 and 1.1. Enforcing the `TypeError` on all supported environments ensures users of pwned-passwords-django do not write code that accidentally works in one and only one environment, and supplies a more accurate and comprehensible exception than the `AttributeError` which would have been raised in previous versions.
- The default error and help messages of *PwnedPasswordsValidator* now match the messages of Django's `CommonPasswordValidator`. Since `PwnedPasswordsValidator` falls back to `CommonPasswordValidator` when the Pwned Passwords API is unresponsive, this provides consistency of messages, and also ensures the messages are translated (Django provides translations for its built-in messages).

### 1.9.5 1.1 – released 2018-03-06

#### New features:

N/A

#### Bugs fixed:

- Case sensitivity issue. The Pwned Passwords API always uses uppercase hexadecimal digits for password hashes; pwned-passwords-django was using lowercase. Fixed by switching pwned-passwords-django to use uppercase.

#### Other changes

N/A

### 1.9.6 1.0 – released 2018-03-06

Initial public release.

**See also:**

- About Have I Been Pwned
- The Pwned Passwords range-search API

# Python Module Index

## d



## p

# Index

## D



## P