django-pwned-passwords Documentation

Release 1.1

James Bennett

Mar 06, 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 1.1 supports Django 1.11 and Django 2.0, on Python versions supported by those versions of Django:

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

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 came with `pip` bundled, but if it is missing, instructions are available for how to obtain and install it.

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 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.

1.3 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.

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.

1.4 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, or `None` if it is not found.

Parameters **`password`** (`str`) – The password to check.

Return type `int` or `None`

1.5 Frequently asked questions

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

1.5.1 What versions of Django and Python are supported?

Django 1.11 and 2.0 are supported, on any version of Python supported by those Django versions. This includes Python 2.7 (only on Django 1.11), Python 3.4, Python 3.5, and Python 3.6.

1.5.2 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.

1.5.3 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.

1.5.4 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.

1.5.5 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.

See also:

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

Python Module Index

Index