django-ratelimit-backend Documentation

Release 2.0

Bruno Renié

Aug 27, 2018

Contents

Django-ratelimit-backend is an app that allows rate-limiting of login attempts at the authentication backend level. Login attempts are stored in the cache so you need a properly configured cache setup.

By default, it blocks any IP that has more than 30 failed login attempts in the past 5 minutes. The IP can still browse your site, only login attempts are blocked.

Note: If you use a custom authentication backend, there is an additional configuration step. Check the *custom backends* section.

CHAPTER 1

Usage

1.1 Installation

```
pip install django-ratelimit-backend
```

There's nothing to add to your `INSTALLED_APPS`, unless you want to run the tests. In which case, add `'ratelimitbackend'`.

1.2 Quickstart

- Set your `AUTHENTICATION_BACKENDS` to:

  ```
  AUTHENTICATION_BACKENDS = (
      'ratelimitbackend.backends.RateLimitModelBackend',
  )
  ```

  If you have a custom backend, see the *backends reference*.

- Everytime you use `django.contrib.auth.views.login`, use `ratelimitbackend.views.login` instead.

- Register ratelimitbackend's admin URLs in your URLConf instead of the default admin URLs.

  In your `urls.py`:

  ```
  from ratelimitbackend import admin

  urlpatterns += [
      (r'^admin/', include(admin.site.urls)),
  ]
  ```

  Ratelimitbackend's admin site overrides the default admin login view to add rate-limiting. You can keep registering your models to the default admin site and they will show up in the ratelimitbackend-enabled admin.

- Add `'ratelimitbackend.middleware.RateLimitMiddleware'` to your `MIDDLEWARE_CLASSES`, or create you own middleware to handle rate limits. See the *middleware reference*.
- If you use `django.contrib.auth.forms.AuthenticationForm` directly, replace it with `ratelimitbackend.forms.AuthenticationForm` and **always** pass it the request object. For instance:

```
if request.method == 'POST':
    form = AuthenticationForm(data=request.POST, request=request)
    # etc. etc.
```

  If you use `django.contrib.auth.authenticate`, pass it the request object as well.

1.3 Customizing rate-limiting criteria

By default, rate limits are based on the IP of the client. An IP that submits a form too many times gets rate-limited, whatever it submits. For custom rate-limiting you can subclass the backend and implement your own logic.

Let's see with an example: instead of checking the client's IP, we will use a combination of the IP *and* the tried username. This way after 30 failed attempts with one username, people can start brute-forcing a new username. Yay! More seriously, it can become useful if you have lots of users logging in at the same time from the same IP.

While we're at it, we'll also allow 50 login attempts every 10 minutes.

To do this, simply subclass `ratelimitbackend.backends.RateLimitModelBackend`:

```
from ratelimitbackend.backends import RateLimitModelBackend

class MyBackend(RateLimitModelBackend):
    minutes = 10
    requests = 50

    def key(self, request, dt):
        return '%s%s-%s-%s' % (
            self.cache_prefix,
            self.get_ip(request),
            request.POST['username'],
            dt.strftime('%Y%m%d%H%M'),
        )
```

The `key()` method is used to build the cache keys storing the login attempts. The default implementation doesn't use POST data, here we're adding another part to the cache key.

Note that we're not sanitizing anything, so we may end up with a rather long cache key. Be careful.

For all the details about the rate-limiting implementation, see the *backend reference*.

1.4 Using with other backends

The way django-ratelimit-backend is implemented requires the authentication backends to have an `authenticate()` that takes an additional `request` keyword argument.

While django-ratelimit-backend works fine with the default `ModelBackend` by providing a replacement class, it's obviously not possible to do that for every single backend.

The way to deal with this is to create a custom class using the `RateLimitMixin` class before registering the backend in your settings. For instance, for the LdapAuthBackend:

```
from django_auth_ldap.backend import LDAPBackend
from ratelimitbackend.backends import RateLimitMixin

class RateLimitedLDAPBackend(RateLimitMixin, LDAPBackend):
    pass

AUTHENTICATION_BACKENDS = (
    'path.to.settings.RateLimitedLDAPBackend',
)
```

`RateLimitMixin` lets you simply add rate-limiting capabilities to any authentication backend.

`RateLimitMixin` throws a warning when no request is passed to its `authenticate()` method. This warning also contains the username that was passed. If you use an authentication backend that doesn't take the traditional `username` and `password` arguments, set the `username_key` attribute on the backend class to the proper keyword argument name. For instance, if your backend authenticates with an `email`:

```
class CustomBackend(BaseBackend):
    def authenticate(self, email, password):
        ...

class RateLimitedLCustomBackend(RateLimitMixin, CustomBackend):
    username_key = 'email'
```

If your backend does not have the concept of a `username` at all, for example with OAuth 2 bearer token authentication, set the `no_username` attribute on the backend class to `True`.

The `RateLimitNoUsernameModelBackend` can be used for this purpose if you don't need any additional customization.

CHAPTER 2

Reference

2.1 Authentication backends

class ratelimitbackend.backends.**RateLimitMixin**
This is where the rate-limiting logic is implemented. Failed login attempts are cached for 5 minutes and when the treshold is reached, the remote IP is blocked whether its attempts are valid or not.

RateLimitMixin.**cache_prefix**
The prefix to use for cache keys. Defaults to `'ratelimitbackend-'`

RateLimitMixin.**minutes**
Number of minutes after which login attempts are not taken into account. Defaults to `5`.

RateLimitMixin.**requests**
Number of login attempts to allow during `minutes`. Defaults to `30`.

RateLimitMixin.**authenticate**(*username*, *password*, *request*)
Tries to `authenticate(username, password)` on the parent backend and use the request for rate-limiting.

RateLimitMixin.**get_counters**(*request*)
Fetches the previous failed login attempts from the cache. There is one cache key per minute slot.

RateLimitMixin.**keys_to_check**(*request*)
Returns the list of keys to try to fetch from the cache for previous login attempts. For a 5-minute limit, this returns the 5 relevant cache keys.

RateLimitMixin.**get_cache_key**(*request*)
Returns the cache key for the current time. This is the key to increment if the login attempt has failed.

RateLimitMixin.**key**(*request*, *dt*)
Derives a cache key from the request and a datetime object. The datetime object can be present (for the current request) or past (for the previous cache keys).

RateLimitMixin.**get_ip**(*request*)
Extracts the client IP address from the request. By defaults the IP is read from re-

quest.META['REMOTE_ADDR'] but you can override this if you have a proxy that uses a custom header such as `X-Forwarded-For`.

`RateLimitMixin.`**`cache_incr`**`(`*`key`*`)`
Performs an increment operation on `key`. The implementation is **not** atomic. If you have a cache backend that supports atomic increment operations, you're advised to override this method.

`RateLimitMixin.`**`expire_after`**`()`
Returns the cache timeout for keys.

`class` `ratelimitbackend.backends.`**`RateLimitModelBackend`**
A rate-limited version of `django.contrib.auth.backends.ModelBackend`.

This is a subclass of `django.contrib.auth.backends.ModelBackend` that adds rate-limiting. If you have custom backends, make sure they inherit from this instead of the default `ModelBackend`.

If your backend has nothing to do with Django's auth system, use `RateLimitMixin` to inject the rate-limiting functionality in your backend.

2.2 Exceptions

`class` `ratelimitbackend.exceptions.`**`RateLimitException`**
The exception thrown when a user reaches the limits.

`RateLimitException.`**`counts`**
A dictionnary containing the cache keys for every minute and the corresponding failed login attempts.

Example:

```
{
    'ratelimitbackend-127.0.0.1-201110181448': 12,
    'ratelimitbackend-127.0.0.1-201110181449': 18,
}
```

2.3 Admin

`class` `ratelimitbackend.admin.`**`RateLimitAdminSite`**
Rate-limited version of the default Django admin site. If you use the default admin site (`django.contrib.admin.site`), it won't be rate-limited.

If you have a custom admin site (inheriting from `AdminSite`), you need to make it inherit from `ratelimitbackend.RateLimitAdminSite`, replacing:

```
from django.contrib import admin

class AdminSite(admin.AdminSite):
    pass
site = AdminSite()
```

with:

```
from ratelimitbackend import admin

class AdminSite(admin.RateLimitAdminSite):
    pass
site = AdminSite()
```

Make sure your calls to `admin.site.register` reference the correct admin site.

`RateLimitAdminSite.`**`login`**(*request*, *extra_context=None*)
This method calls django-ratelimit-backend's version of the login view.

2.4 Middleware

`class` `ratelimitbackend.middleware.`**`RateLimitMiddleware`**
This middleware catches `RateLimitException` and returns a 403 instead, with a `'text/plain'` mime-type. Use your custom middleware if you need a different behaviour.

2.5 Views

`ratelimitbackend.views.`**`login`**(*request*[, *template_name*, *redirect_field_name*, *authentication_form*])
This function uses a custom authentication form and passes it the request object. The external API is the same as Django's login view.

2.6 Forms

`class` `ratelimitbackend.forms.`**`AuthenticationForm`**
A subclass of Django's authentication form that passes the request object to the `authenticate()` function, hence to the authentication backend.

2.7 Logging

Failed attempts are logged using a logger named `'ratelimitbackend'`. Here is an example for logging to the standard output:

```
LOGGING = {
    'formatters': {
        'simple': {
            'format': '%(asctime)s %(levelname)s: %(message)s'
        },
        # other formatters
    },
    'handlers': {
        'console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'simple',
        },
        # other handlers
    },
    'loggers': {
        'ratelimitbackend': {
            'handlers': ['console'],
            'level': 'INFO',
        },
```

(continues on next page)

(continued from previous page)

```
        # other loggers
    },
}
```

You will see two kinds of messages:

- "No request passed to the backend, unable to rate-limit. Username was…"

  This means you're not using the app correctly, the request object wasn't passed to the authentication backend. Double-check the documentation, and if you make manual calls to login-related functions you may need to pass the request object manually.

  The log level for this message is: `WARNING`.

- "Login failed: username 'foo', IP 127.0.0.1"

  This is a failed attempt that has been temporarily cached.

  The log level for this message is: `INFO`.

- "Login rate-limit reached: username 'foo', IP 127.0.0.1"

  This means someone has used all his quotas and got a `RateLimitException`, locking him temporarily until the quota decreases.

  The log level for this message is: `WARNING`.

Get involved, submit issues and pull requests on the code repository!

CHAPTER 3

Changes

- **2.0** (2018-08-27):
  - Add support for Django 2.0 and 2.1, and drop support for Django < 1.11.
- **1.2** (2017-09-13):
  - Add `no_username` attribute on authentication backend for token-based authentication (Jody McIntyre).
  - Fix Travis build for Python 3.3 (Jody McIntyre).
- **1.1.1** (2017-03-30):
  - Run tests on Python 3.6.
  - Run without warnings on supported Django versions.
- **1.1** (2017-03-16):
  - Exclude tests from being installed from the wheel file.
  - Add support for Django 1.10 and 1.11.
- **1.0** (2015-07-10):
  - Silence warnings with Django 1.8.
- **0.6.4** (2015-03-31):
  - Only set the redirect field to the value of `request.get_full_path()` if the field does not already have a value. Patch by Michael Blatherwick.
- **0.6.3** (2015-02-12):
  - Add `RatelimitMixin.get_ip`.
- **0.6.2** (2014-07-28):
  - Django 1.7 support. Patch by Mathieu Agopian.
- **0.6.1** (2014-01-21):
  - Removed calls to deprecated `check_test_cookie()`.

- **0.6** (2013-04-18):

  The `RatelimitBackend` now allows arbitrary `kwargs` for authentication, not just `username` and `password`. Patch by Trey Hunner.

- **0.5** (2013-02-14):
  - Python 3 compatibility.
  - The backend now issues a warning (`warnings.warn()`) instead of a logging call when no request is passed to the backend. This is because such cases are developer errors so a warning is more appropriate.
- **0.4** (2013-01-20):
  - Automatically re-register models which have been registered in Django's default admin site instance. There is no need to register 3rd-party models anymore.
  - Fixed a couple of deprecation warnings.
- **0.3** (2012-11-22):
  - Removed the part where the admin login form looked up a User object when an email was used to login. This brings support for Django 1.5's swappable user models.
- **0.2** (2012-07-31):
  - Added a logging call when a user reaches its rate-limit.
- **0.1**:
  - Initial version.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index