
Django Axes Documentation

Release 2.2.0

jazzband

Jan 19, 2018

Contents

1	Contents	1
1.1	Installation	1
1.2	Configuration	1
1.3	Usage	3
1.4	Requirements	5
1.5	Development	6
1.6	Using a captcha	6
2	Indices and tables	9

1.1 Installation

You can install the latest stable package running this command:

```
$ pip install django-axes
```

1.2 Configuration

Just add *axes* to your `INSTALLED_APPS`:

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    ...  
    'axes',  
    ...  
)
```

Remember to run `python manage.py migrate` to sync the database.

1.2.1 Known configuration problems

If you are running Axes on a deployment with in-memory Django cache, the `axes_reset` functionality might not work predictably.

Axes caches access attempts application-wide, and the in-memory cache only caches access attempts per Django process, so for example resets made in one web server process or the command line with `axes_reset` might not

remove lock-outs that are in the separate process' in-memory cache such as the web server process serving your login or admin page.

To circumvent this problem please use something else than `django.core.cache.backends.locmem.LocMemCache` as your cache backend in Django cache `BACKEND` setting.

There are no known problems in other cache backends such as `DummyCache`, `FileBasedCache`, or `MemcachedCache` backends.

1.2.2 Customizing Axes

You have a couple options available to you to customize `django-axes` a bit. These should be defined in your `settings.py` file.

- `AXES_FAILURE_LIMIT`: The number of login attempts allowed before a record is created for the failed logins. Default: 3
- `AXES_LOCK_OUT_AT_FAILURE`: After the number of allowed login attempts are exceeded, should we lock out this IP (and optional user agent)? Default: `True`
- `AXES_USE_USER_AGENT`: If `True`, lock out / log based on an IP address AND a user agent. This means requests from different user agents but from the same IP are treated differently. Default: `False`
- `AXES_COOLOFF_TIME`: If set, defines a period of inactivity after which old failed login attempts will be forgotten. Can be set to a python `timedelta` object or an integer. If an integer, will be interpreted as a number of hours. Default: `None`
- `AXES_LOGGER`: If set, specifies a logging mechanism for axes to use. Default: `'axes.watch_login'`
- `AXES_LOCKOUT_TEMPLATE`: If set, specifies a template to render when a user is locked out. Template receives `cooloff_time` and `failure_limit` as context variables. Default: `None`
- `AXES_LOCKOUT_URL`: If set, specifies a URL to redirect to on lockout. If both `AXES_LOCKOUT_TEMPLATE` and `AXES_LOCKOUT_URL` are set, the template will be used. Default: `None`
- `AXES_VERBOSE`: If `True`, you'll see slightly more logging for Axes. Default: `True`
- `AXES_USERNAME_FORM_FIELD`: the name of the form field that contains your users usernames. Default: `username`
- `AXES_PASSWORD_FORM_FIELD`: the name of the form field that contains your users password. Default: `password`
- `AXES_LOCK_OUT_BY_COMBINATION_USER_AND_IP`: If `True` prevents the login from IP under a particular user if the attempt limit has been exceeded, otherwise lock out based on IP. Default: `False`
- `AXES_ONLY_USER_FAILURES` : If `True` only locks based on user id and never locks by IP if attempts limit exceed, otherwise utilize the existing IP and user locking logic Default: `False`
- `AXES_NEVER_LOCKOUT_WHITELIST`: If `True`, users can always login from whitelisted IP addresses. Default: `False`
- `AXES_IP_WHITELIST`: A list of IP's to be whitelisted. For example: `AXES_IP_WHITELIST=['0.0.0.0']`. Default: `[]` Default: `False`
- `AXES_DISABLE_ACCESS_LOG`: If `True`, disable all access logging, so the admin interface will be empty. Default: `False`
- `AXES_DISABLE_SUCCESS_ACCESS_LOG`: If `True`, successful logins will not be logged, so the access log shown in the admin interface will only list unsuccessful login attempts. Default: `False`

1.3 Usage

`django-axes` listens to signals from `django.contrib.auth.signals` to log access attempts:

- `user_logged_in`
- `user_logged_out`
- `user_login_failed`

You can also use `django-axes` with your own auth module, but you'll need to ensure that it sends the correct signals in order for `django-axes` to log the access attempts.

1.3.1 Quickstart

Once `axes` is in your `INSTALLED_APPS` in your project settings file, you can login and logout of your application via the `django.contrib.auth` views. The access attempts will be logged and visible in the "Access Attempts" section of the admin app.

By default, `django-axes` will lock out repeated attempts from the same IP address. You can allow this IP to attempt again by deleting the relevant `AccessAttempt` records in the admin.

You can also use the `axes_reset` management command using Django's `manage.py`.

- `manage.py axes_reset` will reset all lockouts and access records.
- `manage.py axes_reset ip` will clear lockout/records for `ip`

In your code, you can use `from axes.utils import reset`.

- `reset()` will reset all lockouts and access records.
- `reset(ip=ip)` will clear lockout/records for `ip`
- `reset(username=username)` will clear lockout/records for a `username`

1.3.2 Example usage

Here is a more detailed example of sending the necessary signals using `django-axes` and a custom auth backend at an endpoint that expects JSON requests. The custom authentication can be swapped out with `authenticate` and `login` from `django.contrib.auth`, but beware that those methods take care of sending the necessary signals for you, and there is no need to duplicate them as per the example.

forms.py:

```
from django import forms

class LoginForm(forms.Form):
    username = forms.CharField(max_length=128, required=True)
    password = forms.CharField(max_length=128, required=True)
```

views.py:

```
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from django.http import JsonResponse, HttpResponse
from django.contrib.auth.signals import user_logged_in, \
    user_logged_out, \
    user_login_failed
```

```
import json
from myapp.forms import LoginForm
from myapp.auth import custom_authenticate, custom_login

@method_decorator(csrf_exempt, name='dispatch')
class Login(View):
    ''' Custom login view that takes JSON credentials '''

    http_method_names = ['post',]

    def post(self, request):
        # decode post json to dict & validate
        post_data = json.loads(request.body.decode('utf-8'))
        form = LoginForm(post_data)

        if not form.is_valid():
            # inform axes of failed login
            user_login_failed.send(
                sender = User,
                request = request,
                credentials = {
                    'username': form.cleaned_data.get('username')
                }
            )
            return HttpResponse(status=400)
        user = custom_authenticate(
            request = request,
            username = form.cleaned_data.get('username'),
            password = form.cleaned_data.get('password'),
        )

        if user is not None:
            custom_login(request, user)
            user_logged_in.send(
                sender = User,
                request = request,
                user = user,
            )
            return JsonResponse({'message': 'success!'}, status=200)
        else:
            user_login_failed.send(
                sender = User,
                request = request,
                credentials = {
                    'username': form.cleaned_data.get('username')
                },
            )
            return HttpResponse(status=403)
```

urls.py:

```
from django.urls import path
from myapp.views import Login

urlpatterns = [
    path('login/', Login.as_view(), name='login'),
]
```


1.3.3 Integration with django-allauth

axes relies on having login information stored under `AXES_USERNAME_FORM_FIELD` key both in `request.POST` and in `credentials` dict passed to `user_login_failed` signal. This is not the case with `allauth`. `allauth` always uses `login` key in post `POST` data but it becomes `username` key in `credentials` dict in signal handler.

To overcome this you need to use custom login form that duplicates the value of `username` key under a `login` key in that dict (and set `AXES_USERNAME_FORM_FIELD = 'login'`).

You also need to decorate `dispatch()` and `form_invalid()` methods of the `allauth` login view. By default axes is patching only the `LoginView` from `django.contrib.auth` app and with `allauth` you have to do the patching of views yourself.

settings.py:

```
AXES_USERNAME_FORM_FIELD = 'login'
```

forms.py:

```
from allauth.account.forms import LoginForm

class AllauthCompatLoginForm(LoginForm):
    def user_credentials(self):
        credentials = super(AllauthCompatLoginForm, self).user_credentials()
        credentials['login'] = credentials.get('email') or credentials.get('username')
        return credentials
```

urls.py:

```
from allauth.account.views import LoginView
from axes.decorators import axes_dispatch
from axes.decorators import axes_form_invalid
from django.utils.decorators import method_decorator

from my_app.forms import AllauthCompatLoginForm

LoginView.dispatch = method_decorator(axes_dispatch)(LoginView.dispatch)
LoginView.form_invalid = method_decorator(axes_form_invalid)(LoginView.form_invalid)

urlpatterns = [
    # ...
    url(r'^accounts/login/$', # Override allauth's default view with a patched view
        LoginView.as_view(form_class=AllauthCompatLoginForm),
        name="account_login"),
    url(r'^accounts/', include('allauth.urls')),
    # ...
]
```

1.4 Requirements

`django-axes` requires a supported Django version. The application is intended to work around the Django admin and the regular `django.contrib.auth` login-powered pages. Look here <https://github.com/jazzband/django-axes/blob/master/.travis.yml> to check if your `django` / `python` version are supported.

1.5 Development

You can contribute to this project forking it from github and sending pull requests.

This is a [Jazzband](#) project. By contributing you agree to abide by the [Contributor Code of Conduct](#) and follow the [guidelines](#).

1.5.1 Running tests

Clone the repository and install the django version you want. Then run:

```
$ ./runtests.py
```

1.6 Using a captcha

Using <https://github.com/mbi/django-simple-captcha> you do the following:

1. Change axes lockout url in `settings.py`:

```
AXES_LOCKOUT_URL = '/locked'
```

2. Add the url in `urls.py`:

```
url(r'^locked/$', locked_out, name='locked_out'),
```

3. Create a captcha form:

```
class AxesCaptchaForm(forms.Form):  
    captcha = CaptchaField()
```

4. Create a captcha view for the above url that resets on captcha success and redirects:

```
def locked_out(request):  
    if request.POST:  
        form = AxesCaptchaForm(request.POST)  
        if form.is_valid():  
            ip = get_ip_address_from_request(request)  
            reset(ip=ip)  
            return HttpResponseRedirect(reverse_lazy('signin'))  
    else:  
        form = AxesCaptchaForm()  
  
    return render_to_response('locked_out.html', dict(form=form), context_  
↪instance=RequestContext(request))
```

5. Add a captcha template:

```
<form action="" method="post">  
    {% csrf_token %}  
  
    {{ form.captcha.errors }}  
    {{ form.captcha }}  
  
    <div class="form-actions">
```

```
<input type="submit" value="Submit" />
</div>
</form>
```


CHAPTER 2

Indices and tables

- search