mozilla-django-oidc Documentation

Release 1.2.2

Mozilla

Aug 02, 2019

Contents

Contents:

CHAPTER 1

Installation

At the command line:

```
$ pip install mozilla-django-oidc
```

1.1 Quick start

After installation, you'll need to do some things to get your site using `mozilla-django-oidc`.

1.1.1 Requirements

This library supports Python 2.7 and 3.3+ on OSX and Linux.

1.1.2 Acquire a client id and client secret

Before you can configure your application, you need to set up a client with an OpenID Connect provider (OP).

You'll need to set up a *different client* for every environment you have for your site. For example, if your site has a -dev, -stage, and -prod environments, each of those has a different hostname and thus you need to set up a separate client for each one.

You need to provide your OpenID Connect provider (OP) the callback url for your site. The URL path for the callback url is `/oidc/callback/`.

Here are examples of callback urls:

- `http://127.0.0.1:8000/oidc/callback/` – for local development
- `https://myapp-dev.example.com/oidc/callback/` – -dev environment for myapp
- `https://myapp.herokuapps.com/oidc/callback/` – my app running on Heroku

The OpenID Connect provider (OP) will then give you the following:

1. a client id (`OIDC_RP_CLIENT_ID`)
2. a client secret (`OIDC_RP_CLIENT_SECRET`)

You'll need these values for settings.

1.1.3 Choose the appropriate algorithm

Depending on your OpenID Connect provider (OP) you might need to change the default signing algorithm from `HS256` to `RS256` by settings the `OIDC_RP_SIGN_ALGO` value accordingly.

For `RS256` algorithm to work, you need to set either the OP signing key or the OP JWKS Endpoint.

The corresponding settings values are:

```
OIDC_RP_IDP_SIGN_KEY = "<OP signing key in PEM or DER format>"
OIDC_OP_JWKS_ENDPOINT = "<URL of the OIDC OP jwks endpoint>"
```

If both specified, the key takes precedence.

1.1.4 Add settings to settings.py

Start by making the following changes to your `settings.py` file.

```
# Add 'mozilla_django_oidc' to INSTALLED_APPS
INSTALLED_APPS = (
    # ...
    'django.contrib.auth',
    'mozilla_django_oidc',  # Load after auth
    # ...
)

# Add 'mozilla_django_oidc' authentication backend
AUTHENTICATION_BACKENDS = (
    'mozilla_django_oidc.auth.OIDCAuthenticationBackend',
    # ...
)
```

You also need to configure some OpenID Connect related settings too.

These values come from your OpenID Connect provider (OP).

```
OIDC_RP_CLIENT_ID = os.environ['OIDC_RP_CLIENT_ID']
OIDC_RP_CLIENT_SECRET = os.environ['OIDC_RP_CLIENT_SECRET']
```

> **Warning:** The OpenID Connect provider (OP) provided client id and secret are secret values.
>
> **DON'T** check them into version control–pull them in from the environment.
>
> If you ever accidentally check them into version control, contact your OpenID Connect provider (OP) as soon as you can, disable that set of client id and secret, and generate a new set.

These values are specific to your OpenID Connect provider (OP)–consult their documentation for the appropriate values.

```
OIDC_OP_AUTHORIZATION_ENDPOINT = "<URL of the OIDC OP authorization endpoint>"
OIDC_OP_TOKEN_ENDPOINT = "<URL of the OIDC OP token endpoint>"
OIDC_OP_USER_ENDPOINT = "<URL of the OIDC OP userinfo endpoint>"
```

Warning: Don't use Django's cookie-based sessions because they might open you up to replay attacks.

You can find more info about cookie-based sessions in Django's documentation.

These values relate to your site.

```
LOGIN_REDIRECT_URL = "<URL path to redirect to after login>"
LOGOUT_REDIRECT_URL = "<URL path to redirect to after logout>"
```

1.1.5 Add routing to urls.py

Next, edit your `urls.py` and add the following:

```
urlpatterns = patterns(
    # ...
    url(r'^oidc/', include('mozilla_django_oidc.urls')),
    # ...
)
```

1.1.6 Add login link to templates

Then you need to add the login link to your templates. The view name is `oidc_authentication_init`.

Django templates example:

```
<html>
  <body>
    {% if user.is_authenticated %}
      <p>Current user: {{ user.email }}</p>
    {% else %}
      <a href="{% url 'oidc_authentication_init' %}">Login</a>
    {% endif %}
  </body>
</html>
```

Jinja2 templates example:

```
<html>
  <body>
    {% if user.is_authenticated() %}
      <p>Current user: {{ user.email }}</p>
    {% else %}
      <a href="{{ url('oidc_authentication_init') }}">Login</a>
    {% endif %}
  </body>
</html>
```

1.2 Additional optional configuration

1.2.1 Validate ID tokens by renewing them

Users log into your site by authenticating with an OIDC provider. While the user is doing things on your site, it's possible that the account that the user used to authenticate with the OIDC provider was disabled. A classic example of this is when a user quits his/her job and their LDAP account is disabled.

However, even if that account was disabled, the user's account and session on your site will continue. In this way, a user can quit his/her job, lose access to his/her corporate account, but continue to use your website.

To handle this scenario, your website needs to know if the user's id token with the OIDC provider is still valid. You need to use the `mozilla_django_oidc.middleware.SessionRefresh` middleware.

To add it to your site, put it in the settings:

```
MIDDLEWARE_CLASSES = [
    # middleware involving session and authentication must come first
    # ...
    'mozilla_django_oidc.middleware.SessionRefresh',
    # ...
]
```

The `mozilla_django_oidc.middleware.SessionRefresh` middleware will check to see if the user's id token has expired and if so, redirect to the OIDC provider's authentication endpoint for a silent re-auth. That will redirect back to the page the user was going to.

The length of time it takes for an id token to expire is set in `settings.OIDC_RENEW_ID_TOKEN_EXPIRY_SECONDS` which defaults to 15 minutes.

1.2.2 Connecting OIDC user identities to Django users

By default, mozilla-django-oidc looks up a Django user matching the email field to the email address returned in the user info data from the OIDC provider.

This means that no two users in the Django user table can have the same email address. Since the email field is not unique, it's possible that this can happen. Especially if you allow users to change their email address. If it ever happens, then the users in question won't be able to authenticate.

If you want different behavior, subclass the `mozilla_django_oidc.auth.OIDCAuthenticationBackend` class and override the *filter_users_by_claims* method.

For example, let's say we store the email address in a `Profile` table in a field that's marked unique so multiple users can't have the same email address. Then we could do this:

```
from mozilla_django_oidc.auth import OIDCAuthenticationBackend

class MyOIDCAB(OIDCAuthenticationBackend):
    def filter_users_by_claims(self, claims):
        email = claims.get('email')
        if not email:
            return self.UserModel.objects.none()

        try:
            profile = Profile.objects.get(email=email)
            return profile.user
```

(continues on next page)

(continued from previous page)

```
        except Profile.DoesNotExist:
            return self.UserModel.objects.none()
```

Then you'd use the Python dotted path to that class in the `settings.AUTHENTICATION_BACKENDS` instead of `mozilla_django_oidc.auth.OIDCAuthenticationBackend`.

1.2.3 Creating Django users

Generating usernames

If a user logs into your site and doesn't already have an account, by default, mozilla-django-oidc will create a new Django user account. It will create the `User` instance filling in the username (hash of the email address) and email fields.

If you want something different, set `settings.OIDC_USERNAME_ALGO` to a Python dotted path to the function you want to use.

The function takes in an email address as a text (Python 2 unicode or Python 3 string) and returns a text (Python 2 unicode or Python 3 string).

Here's an example function for Python 3 and Django 1.11 that doesn't convert the email address at all:

```
import unicodedata

def generate_username(email):
    # Using Python 3 and Django 1.11, usernames can contain alphanumeric
    # (ascii and unicode), _, @, +, . and - characters. So we normalize
    # it and slice at 150 characters.
    return unicodedata.normalize('NFKC', email)[:150]
```

See also:

Django 1.8 username: https://docs.djangoproject.com/en/1.8/ref/contrib/auth/#django.contrib.auth.models.User.username

Django 1.11 username: https://docs.djangoproject.com/en/1.11/ref/contrib/auth/#django.contrib.auth.models.User.username

Django 2.0 username: https://docs.djangoproject.com/en/2.0/ref/contrib/auth/#django.contrib.auth.models.User.username

Changing how Django users are created

If your website needs to do other bookkeeping things when a new `User` record is created, then you should subclass the `mozilla_django_oidc.auth.OIDCAuthenticationBackend` class and override the *create_user* method, and optionally, the *update_user* method.

For example, let's say you want to populate the `User` instance with other data from the claims:

```
from mozilla_django_oidc.auth import OIDCAuthenticationBackend
from myapp.models import Profile

class MyOIDCAB(OIDCAuthenticationBackend):
    def create_user(self, claims):
        user = super(MyOIDCAB, self).create_user(claims)
```

(continues on next page)

(continued from previous page)

```
        user.first_name = claims.get('given_name', '')
        user.last_name = claims.get('family_name', '')
        user.save()

        return user

    def update_user(self, user, claims):
        user.first_name = claims.get('given_name', '')
        user.last_name = claims.get('family_name', '')
        user.save()

        return user
```

Then you'd use the Python dotted path to that class in the `settings.AUTHENTICATION_BACKENDS` instead of `mozilla_django_oidc.auth.OIDCAuthenticationBackend`.

See also:

https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

Preventing mozilla-django-oidc from creating new Django users

If you don't want mozilla-django-oidc to create Django users, you can add this setting:

```
OIDC_CREATE_USER = False
```

You might want to do this if you want to control user creation because your system requires additional process to allow people to use it.

Advanced user verification based on their claims

In case you need to check additional values in the user's claims to decide if the authentication should happen at all (included creating new users if `OIDC_CREATE_USER` is `True`), then you should subclass the `mozilla_django_oidc.auth.OIDCAuthenticationBackend` class and override the *verify_claims* method. It should return either `True` or `False` to either continue or stop the whole authentication process.

```
class MyOIDCAB(OIDCAuthenticationBackend):
    def verify_claims(self, claims):
        verified = super(MyOIDCAB, self).verify_claims(claims)
        is_admin = 'admin' in claims.get('group', [])
        return verified and is_admin
```

See also:

https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

Log user out of the OpenID Connect provider

When a user logs out, by default, mozilla-django-oidc will end the current Django session. However, the user may still have an active session with the OpenID Connect provider, in which case, the user would likely not be prompted to log back in.

Some OpenID Connect providers support a custom (not part of OIDC spec) mechanism to end the provider's session. We can build a function for `OIDC_OP_LOGOUT_URL_METHOD` that will redirect the user to the provider after mozilla-django-oidc ends the Django session.

```
def provider_logout(request):
    # See your provider's documentation for details on if and how this is
    # supported
    redirect_url = 'https://myprovider.com/logout'
    return redirect_url
```

The `request.build_absolute_uri` can be used if the provider requires a return-to location.

1.2.4 Troubleshooting

mozilla-django-oidc logs using the `mozilla_django_oidc` logger. Enable that logger in settings to see logging messages to help you debug:

```
LOGGING = {
    ...
    'loggers': {
        'mozilla_django_oidc': {
            'handlers': ['console'],
            'level': 'DEBUG'
        },
    ...
}
```

Make sure to use the appropriate handler for your app.

CHAPTER 2

Settings

This document describes the Django settings that can be used to customize the configuration of `mozilla-django-oidc`.

OIDC_OP_AUTHORIZATION_ENDPOINT

Default No default

URL of your OpenID Connect provider authorization endpoint.

OIDC_OP_TOKEN_ENDPOINT

Default No default

URL of your OpenID Connect provider token endpoint

OIDC_OP_USER_ENDPOINT

Default No default

URL of your OpenID Connect provider userinfo endpoint

OIDC_RP_CLIENT_ID

Default No default

OpenID Connect client ID provided by your OP

OIDC_RP_CLIENT_SECRET

Default No default

OpenID Connect client secret provided by your OP

OIDC_VERIFY_JWT

Default `True`

Controls whether the OpenID Connect client verifies the signature of the JWT tokens

OIDC_USE_NONCE

Default `True`

Controls whether the OpenID Connect client uses nonce verification

OIDC_VERIFY_SSL

Default `True`

Controls whether the OpenID Connect client verifies the SSL certificate of the OP responses

OIDC_EXEMPT_URLS

Default `[]`

This is a list of absolute url paths or Django view names. This plus the mozilla-django-oidc urls are exempted from the session renewal by the `SessionRefresh` middleware.

OIDC_CREATE_USER

Default `True`

Enables or disables automatic user creation during authentication

OIDC_STATE_SIZE

Default `32`

Sets the length of the random string used for OpenID Connect state verification

OIDC_NONCE_SIZE

Default `32`

Sets the length of the random string used for OpenID Connect nonce verification

OIDC_REDIRECT_FIELD_NAME

Default `next`

Sets the GET parameter that is being used to define the redirect URL after succesful authentication

OIDC_CALLBACK_CLASS

Default `mozilla_django_oidc.views.OIDCAuthenticationCallbackView`

Allows you to substitute a custom class-based view to be used as OpenID Connect callback URL.

Note: When using a custom callback view, it is generally a good idea to subclass the default `OIDCAuthenticationCallbackView` and override the methods you want to change.

OIDC_AUTHENTICATE_CLASS

Default `mozilla_django_oidc.views.OIDCAuthenticationRequestView`

Allows you to substitute a custom class-based view to be used as OpenID Connect authenticate URL.

Note: When using a custom authenticate view, it is generally a good idea to subclass the default `OIDCAuthenticationRequestView` and override the methods you want to change.

OIDC_RP_SCOPES

Default `openid email`

The OpenID Connect scopes to request during login.

> **Warning:** When using custom scopes consider overriding the *claim verification method* since the default one only works for the default `mozilla-django-oidc` configuration.

OIDC_STORE_ACCESS_TOKEN

Default `False`

Controls whether the OpenID Connect client stores the OIDC `access_token` in the user session. The session key used to store the data is `oidc_access_token`.

By default we want to store as few credentials as possible so this feature defaults to `False` and it's use is discouraged.

> **Warning:** This feature stores authentication information in the session. If used in combination with Django's cookie-based session backend, those tokens will be visible in the browser's cookie store.

OIDC_STORE_ID_TOKEN

Default `False`

Controls whether the OpenID Connect client stores the OIDC `id_token` in the user session. The session key used to store the data is `oidc_id_token`.

OIDC_AUTH_REQUEST_EXTRA_PARAMS

Default *{}*

Additional parameters to include in the initial authorization request.

OIDC_RP_SIGN_ALGO

Default `HS256`

Sets the algorithm the IdP uses to sign ID tokens.

OIDC_RP_IDP_SIGN_KEY

Default `None`

Sets the key the IdP uses to sign ID tokens in the case of an RSA sign algorithm. Should be the signing key in PEM or DER format.

LOGIN_REDIRECT_URL

Default `/accounts/profile`

Path to redirect to on successful login. If you don't specify this, the default Django value will be used.

See also:

https://docs.djangoproject.com/en/1.11/ref/settings/#login-redirect-url

LOGIN_REDIRECT_URL_FAILURE

Default `/`

Path to redirect to on an unsuccessful login attempt.

LOGOUT_REDIRECT_URL

Default `/` (Django <= 1.9) `None` (Django 1.10+)

After the logout view has logged the user out, it redirects to this url path.

See also:

https://docs.djangoproject.com/en/1.11/ref/settings/#logout-redirect-url

OIDC_OP_LOGOUT_URL_METHOD

Default `''` (will use `LOGOUT_REDIRECT_URL`)

Function path that returns a URL to redirect the user to after `auth.logout()` is called.

Changed in version 0.7.0: The function must now take a `request` parameter.

OIDC_AUTHENTICATION_CALLBACK_URL

Default `oidc_authentication_callback`

URL pattern name for `OIDCAuthenticationCallbackView`. Will be passed to `reverse`. The pattern can also include namespace in order to resolve included urls.

See also:

https://docs.djangoproject.com/en/2.0/topics/http/urls/#url-namespaces

OIDC_ALLOW_UNSECURED_JWT

Default `False`

Controls whether the authentication backend is going to allow unsecured JWT tokens (tokens with header `{"alg":"none"}`). This needs to be set to `True` if OP is returning unsecured JWT tokens and RP wants to accept them.

See also:

https://tools.ietf.org/html/rfc7519#section-6

OIDC_TOKEN_USE_BASIC_AUTH

Default False

Use HTTP Basic Authentication instead of sending the client secret in token request POST body.

CHAPTER 3

XHR (AJAX) Usage

If you do configure the middleware that intercepts requests and potentially forces a refresh to refresh your session, this gets tricky with XHR requests. Usually XHR requests (with libraries like `fetch` or `jQuery.ajax`) follow redirects by default (which is most likely a good thing). The problem is that it can't redirect back to the OP when it's time to refresh your session. So for XHR requests, some special handling is required by you.

```
// DON'T DO THIS!

fetch('/server/api/get/stuff', {credentials: 'same-origin'})
.then(response => {
   response.json()
   .then(stuff => {
     doSomethingWith(stuff);
   })
});
```

The problem with the above code is that it's wrong to assume the XHR response is going to be `application/json` if the server's middleware insisted you need to refresh your session.

Instead watch out for a `403 Forbidden` response when, in conjunction, there is a header called `refresh_url`. Like this:

```
// This assumes the /server/api/* requests are intercepted by the
// mozilla-django-oidc refresh middleware.

fetch('/server/api/get/stuff', {credentials: 'same-origin'})
.then(response => {
   if (response.status === 403  && response.headers.get("refresh_url")) {
     // Perhaps do something fancier than alert()
     alert("You have to refresh your authentication.")
     // Redirect the user out of this application.
     document.location.href = response.headers.get("refresh_url");
   } else {
     response.json()
     .then(stuff => {
```

(continues on next page)

(continued from previous page)

```
        doSomethingWith(stuff);
      })
    }
});
```

Note: The refresh middleware only applies to `GET` requests.

You don't have to use `document.location.href` to redirect immediately inside the client-side application. Perhaps you can other things like updating the DOM to say that the user has to refresh their authentication and provide a regular link.

CHAPTER 4

DRF (Django REST Framework) integration

If you want DRF to authenticate users based on an OAuth access token provided in the `Authorization` header, you can use the DRF-specific authentication class which ships with the package.

Add this to your settings:

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'mozilla_django_oidc.contrib.drf.OIDCAuthentication',
        # other authentication classes, if needed
    ],
}
```

Note that this only takes care of authenticating against an access token, and provides no options to create or renew tokens.

If you've created a custom Django `OIDCAuthenticationBackend` and added that to your `AUTHENTICATION_BACKENDS`, the DRF class should be smart enough to figure that out. Alternatively, you can manually set the OIDC backend to use:

```
OIDC_DRF_AUTH_BACKEND = 'mozilla_django_oidc.OIDCAuthenticationBackend'
```

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/mozilla/mozilla-django-oidc/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

mozilla-django-oidc could always use more documentation, whether as part of the official mozilla-django-oidc docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mozilla/mozilla-django-oidc/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *mozilla-django-oidc* for local development.

1. Fork the *mozilla-django-oidc* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mozilla-django-oidc.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mozilla-django-oidc
$ cd mozilla-django-oidc/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mozilla_django_oidc tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Make sure you update `HISTORY.rst` with your changes in the following categories
   - Backwards-incompatible changes
   - Features
   - Bugs
7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/mozilla/mozilla-django-oidc/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

We use tox to run tests:

```
$ tox
```

To run a specific environment, use the `-e` argument:

```
$ tox -e py27-django18
```

You can also run the tests in a virtual environment without tox:

```
$ DJANGO_SETTINGS_MODULE=tests.settings django-admin.py test
```

You can specify test modules to run rather than the whole suite:

```
$ DJANGO_SETTINGS_MODULE=tests.settings django-admin.py test tests.test_views
```

CHAPTER 6

Credits

6.1 Development Lead

- Tasos Katsoulas <akatsoulas@mozilla.com>
- John Giannelos <jgiannelos@mozilla.com>

6.2 Contributors

- Will Kahn-Greene (@willkg)
- Peter Bengtsson (@peterbe)
- Jannis Leidel (@jezdez)
- Jonathan Claudius (@claudijd)
- Patrick Uiterwijk (@puiterwijk)
- Dustin J. Mitchell (@djmitche)
- Giorgos Logiotatidis (@glogiotatidis)
- Olle Jonsson (@olleolleolle)
- @GermanoGuerrini
- John Paulett (@johnpaulett)
- Andreas Lutro (@anlutro)
- @Algogator
- Rob Hudson (@robhudson)
- Garand Tyson (@SirTyson)
- Justin Azoff (@JustinAzoff)

- Antti Palola (@anttipalola)
- Bono de Visser (@kerrermanisNL)
- Chris Brantley (@chrisbrantley)

CHAPTER 7

History

7.1 1.2.2 (2019-04-18)

- Add Mozilla code of conduct
- Allow overriding OIDC settings per class

7.1.1 1.2.1 (2019-01-22)

- Make *verify_claims* compatible with custom scope configuration.

7.1.2 1.2.0 (2019-01-09)

- Improve travis automation for PyPI releases
- Allow basic auth for OIDC token endpoint requests Thanks **‘@anttipalola‘**_
- Replace phantomjs with firefox headless for e2e testing
- Add default email verification claim check Thanks **‘@kerrermanisNL‘**_
- Remove compatibility code for unsupported Django versions
- Add settings to control redirect behavior Thanks **‘@chrisbrantley‘**_

7.1.3 1.1.2 (2018-08-24)

- Fix JWKS handling when OP returns multiple keys Thanks **‘@JustinAzoff‘**_

7.1.4 1.1.1 (2018-08-09)

- Fix *is_safe_url* on Django 2.1
- Fix signature in *authenticate* method to be compatible with Django 2.1
- Remove legacy code for unsupported Django < 1.11 Thanks **'@SirTyson'_**

7.1.5 1.1.0 (2018-08-02)

- Installation doc fixes Thanks **'@mklan'_**
- Drop support for unsupported Django 1.8 and Python 3.3.
- Refactor authentication backend to make it easier to extend Required by DRF support feature.
- Add DRF support Thanks @anlutro
- Improve local docker environment setup
- Add flag to allow using unsecured tokens
- Allow using JWK with optional `alg` Thanks **'@Algogator'_**

7.1.6 1.0.0 (2018-05-09)

- Add OIDC_AUTHENTICATION_CALLBACK_URL as a new configuration parameter
- Fail earlier when JWS algorithm does not OIDC_RP_SIGN_ALGO. Thanks @anlutro
- RS256 verification through `settings.OIDC_OP_JWKS_ENDPOINT` Thanks @GermanoGuerrini
- Refactor OIDCAuthenticationBackend so that token retrieval methods can be overridden in a subclass when you need to.

Backwards-incompatible changes:

- `OIDC_OP_LOGOUT_URL_METHOD` takes a `request` parameter now.
- Changed name of `RefreshIDToken` middleware to `SessionRefresh`.

7.1.7 0.6.0 (2018-03-27)

- Add e2e tests and automation
- Add caching for exempt URLs
- Fix logout when session refresh fails

7.1.8 0.5.0 (2018-01-10)

- Add Django 2.0 support
- Fix tox configuration

Backwards-incompatible changes:

- Drop Django 1.10 support

7.1.9 0.4.2 (2017-11-29)

- Fix OIDC_USERNAME_ALGO to actually load dotted import path of callback.
- Add verify_claims method for advanced authentication checks

7.1.10 0.4.1 (2017-10-25)

- Send bytes to josepy. Fixes python3 support.

7.1.11 0.4.0 (2017-10-24)

Security issues:

- **High**: Replace python-jose with josepy and use pyca/cryptography instead of pycrypto (CVE-2013-7459).

Backwards-incompatible changes:

- `OIDC_RP_IDP_SIGN_KEY` no longer uses the JWK json as `dict` but PEM or DER keys instead.

7.1.12 0.3.2 (2017-10-03)

Features:

- Implement RS256 verification Thanks @puiterwijk

Bugs:

- Use `settings.OIDC_VERIFY_SSL` also when validating the token. Thanks @GermanoGuerrini
- Make OpenID Connect scope configurable. Thanks @puiterwijk
- Add path host injection unit-test (#171)
- Revisit OIDC_STORE_{ACCESS,ID}_TOKEN config entries
- Allow configuration of additional auth parameters

7.1.13 0.3.1 (2017-06-15)

Security issues:

- **Medium**: Sanitize next url for authentication view

7.1.14 0.3.0 (2017-06-13)

Security issues:

- **Low**: Logout using POST not GET (#126)

Backwards-incompatible changes:

- The `settings.SITE_URL` is no longer used. Instead the absolute URL is derived from the request's `get_host()`.
- Only log out by HTTP POST allowed.

Bugs:

- Test suite maintenance (#108, #109, #142)

7.1.15 0.2.0 (2017-06-07)

Backwards-incompatible changes:

- Drop support for Django 1.9 (#130)

  If you're using Django 1.9, you should update Django first.

- Move middleware to `mozilla_django_oidc.middleware` and change it to use authentication endpoint with `prompt=none` (#94)

  You'll need to update your `MIDDLEWARE_CLASSES`/`MIDDLEWARE` setting accordingly.

- Remove legacy `base64` handling of OIDC secret. Now RP secret should be plaintext.

Features:

- Add support for Django 1.11 and Python 3.6 (#85)
- Update middleware to work with Django 1.10+ (#90)
- Documentation updates
- Rework test infrastructure so it's tox-based (#100)

Bugs:

- always decode verified token before `json.load()` (#116)
- always redirect to logout_url even when logged out (#121)
- Change email matching to be case-insensitive (#102)
- Allow combining OIDCAuthenticationBackend with other backends (#87)
- fix is_authenticated usage for Django 1.10+ (#125)

7.1.16 0.1.0 (2016-10-12)

- First release on PyPI.

Index

L

O