mozilla-django-oidc Documentation

Release 0.3.0

Mozilla

Jun 15, 2017

Contents

Contents:

CHAPTER 1

Installation

At the command line:

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

Quick start

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

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.

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 = (
    # ...
    'django.contrib.auth.backends.ModelBackend',
    '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 = "<ULR path to redirect to after login>"
LOGOUT_REDIRECT_URL = "<URL path to redirect to after logout>"
```

Add routing to urls.py

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

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

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>
```

Additional optional configuration

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.RefreshIDToken` middleware.

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

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

The `RefreshIDToken` 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.

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, claim):
        email = claims.get('email')
        if not email:
            return self.UserModel.objects.none()

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

        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`.

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 1.10 username: https://docs.djangoproject.com/en/1.10/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

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.

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(OIDCAuthenticationBackend, self).create_user(claims)

        user.first_name = claim.get('given_name', '')
        user.last_name = claim.get('family_name', '')

        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.

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 url paths or Django view names. This plus the mozilla-django-oidc urls are exempted from the id token renewal by the `RenewIDToken` 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.

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

CHAPTER 3

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:

Types of Contributions

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.

Fix Bugs

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

Implement Features

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

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.

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 :)

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. 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
   ```

7. Submit a pull request through the GitHub website.

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.

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 4

Credits

Development Lead

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

Contributors

- Will Kahn-Greene
- Peter Bengtsson <@peterbe>

CHAPTER 5

History

0.3.0 (2017-06-13)

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.

Features:

- None

Bugs:

- Logout using POST not GET (#126)
- Test suite maintenance (#108, #109, #142)

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)

0.1.0 (2016-10-12)

- First release on PyPI.

Index