Django-Scatter-Auth Documentation

Release 0.2.0

Denis Bobrov

Mar 13, 2020

Contents

1 django-scatter-auth **3**
- 1.1 Documentation . . . 3
- 1.2 Example project . . . 3
- 1.3 Features . . . 3
- 1.4 Quickstart . . . 4
- 1.5 Important details and FAQ . . . 5
- 1.6 Running Tests . . . 5
- 1.7 Credits . . . 5

2 Overview **7**
- 2.1 Sign up . . . 7
- 2.2 Login . . . 8

3 Settings **9**

4 Contributing **11**
- 4.1 Types of Contributions . . . 11
- 4.2 Get Started! . . . 12
- 4.3 Pull Request Guidelines . . . 13
- 4.4 Tips . . . 13

5 Credits **15**
- 5.1 Development Lead . . . 15
- 5.2 Special Thanks . . . 15

6 History **17**
- 6.1 0.2.0 (2020-03-14) . . . 17
- 6.2 0.1.1 (2018-09-10) . . . 17
- 6.3 0.1.0 (2018-08-13) . . . 17

Contents:

CHAPTER 1

django-scatter-auth

django-scatter-auth is a pluggable Django app that enables login/signup via Scatter (EOS extension wallet). The user authenticates themselves by digitally signing the hostname with their wallet's private key.

1.1 Documentation

The full documentation is at https://django-scatter-auth.readthedocs.io.

1.2 Example project

https://github.com/Bearle/django-scatter-auth/tree/master/example

You can check out our example project by cloning the repo and heading into example/ directory. There is a README file for you to check, also.

1.3 Features

- Scatter API login, signup
- Scatter Django forms for signup, login
- Checks signature (validation)
- Uses hostname signing as proof of private key posession
- Easy to set up and use (just one click)
- Custom auth backend
- VERY customizable - uses Django settings, allows for custom User model

- Vanilla Javascript helpers included

1.4 Quickstart

Install django-scatter-auth with pip:

```
pip install django-scatter-auth
```

Add it to your *INSTALLED_APPS*:

```
INSTALLED_APPS = (
    ...
    'scatterauth.apps.scatterauthConfig',
    ...
)
```

Set *'scatterauth.backend.ScatterAuthBackend'* as your authentication backend:

```
AUTHENTICATION_BACKENDS = [
'django.contrib.auth.backends.ModelBackend',
'scatterauth.backend.ScatterAuthBackend'
]
```

Set your User model's field to use as public key storage:

```
SCATTERAUTH_USER_PUBKEY_FIELD = 'username'
```

And if you have some other fields you want to be in the SignupForm, add them too:

```
SCATTERAUTH_USER_SIGNUP_FIELDS = ['email',]
```

Add django-scatter-auth's URL patterns:

```
from scatterauth import urls as scatterauth_urls


urlpatterns = [
    ...
    url(r'^', include(scatterauth_urls)),
    ...
]
```

Add some javascript to handle login:

```
<script src="{% static 'scatterauth/js/scatterauth.js' %}"></script>
```

```
var login_url = '{% url 'scatterauth_login_api' %}';
document.addEventListener('scatterLoaded', scatterExtension => {
  console.log('scatter loaded');
  if (scatter.identity) {
    console.log("Identity found");
    loginWithAuthenticate(login_url,console.log,console.log,console.log,console.log,␣
↪function (resp) {
      window.location.replace(resp.redirect_url);
    });
```

(continues on next page)

(continued from previous page)

```
  } else {
    console.log('identity not found, have to signup');
  }
});
```

You can access signup using {% url 'scatterauth_signup' %} and API signup using {% url 'scatterauth_signup_api' %}.

If you have any questions left, head to the example app https://github.com/Bearle/django-scatter-auth/tree/master/example

1.5 Important details and FAQ

1. *If you set a custom public key field (SCATTERAUTH_USER_PUBKEY_FIELD), it MUST be unique (unique=True).*

This is needed because if it's not, the user can register a new account with the same public key as the other one, meaning that the user can now login as any of those accounts (sometimes being the wrong one).

2. *How do i deal with user passwords or Password is not set*

There should be some code in your project that generates a password using `User.objects.make_random_password` and sends it to a user email. Or, even better, sends them a 'restore password' link. Also, it's possible to copy signup_view to your project, assign it a url, and add the corresponding lines to set some password for a user.

3. *Why don't i have to sign a message? It's needed in django-web3-auth, how this app is secure?*

This app uses scatter's `authenticate` function to handle message signing - hostname being the signed message. This means that the user & the client share knowledge of the original message and the server can verify client's possession of the private key corresponding to the public key.

1.6 Running Tests

Does the code actually work?

```
source <YOURVIRTUALENV>/bin/activate
(myenv) $ pip install tox
(myenv) $ tox
```

1.7 Credits

Tools used in rendering this package:

- Cookiecutter
- cookiecutter-djangopackage

CHAPTER 2

Overview

Django-scatter-auth features 1 view for login (with JSON responses) and 2 views for Signup (one with JSON responses, and the other - using Django Forms and rendered templates).

It also has 2 forms, SignupForm (rendered) and LoginForm (uses hidden inputs, used to validate data only).

Possible configuration includes customizable address field (`SCATTERAUTH_USER_PUBKEY_FIELD`), additional fields for User model (`SCATTERAUTH_USER_SIGNUP_FIELDS`),on/off switch for registration (`SCATTERAUTH_SIGNUP_ENABLED`) and domain which will be used for signed message validation (`SCATTERAUTH_DOMAIN`). You can read more on that in the Settings section. You should also definitely check example app, it features most of the features needed.

2.1 Sign up

The signup process is as follows (signup_view example, signup_api is similar):

1. User heads to the signup URL (`{% url 'scatterauth_signup' %}`)
2. The signup view is rendered with a `SignupForm` which includes `SCATTERAUTH_USER_SIGNUP_FIELDS` and `SCATTERAUTH_USER_PUBKEY_FIELD`
3. The user enters required data and clicks the submit button and the POST request fires to the same URL with `signup_view`
4. **Signup view does the following:** 4.1. Creates an instance of a `SignupForm`. 4.2. Checks if the registration is enabled. 4.3. If the registration is closed or form has errors, returns form with errors 4.4 If the form is valid, saves the user without saving to DB 4.5. Sets the user public key from the form, saves it to DB 4.6. Logins the user using `scatterauth.backend.ScatterAuthBackend` 4.7. Redirects the user to `LOGIN_REDIRECT_URL` or 'next' in get or post params
5. The user is signed up and logged in

2.2 Login

The login process is as follows (login_api example):

1. On some page of the website, there is Javascript which gets the user signature for the website's hostname.
2. The signature is sent to the login_api url (`{% url 'scatterauth_login_api' %}`) alongside the public key.
3. The view validates given parameters agains `LoginForm`
4. The view validates signature with the given public key, and then tries to `authenticate` the user
5. If the user is found, the user is signed in and the view responds with a `redirect_url` for Javascript to handle
6. If the user is not found, the corresponding error is returned

The Javascript is included in the app, also you can check out example app if you are struggling with logging in the user.

CHAPTER 3

Settings

You should specify settings in your settings.py like this:

```
SCATTERAUTH_USER_PUBKEY_FIELD = 'pubkey'
SCATTERAUTH_USER_SIGNUP_FIELDS = ['email', 'username']
```

In the above example the following User model is used:

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils.translation import ugettext_lazy as _

class User(AbstractUser):
    pubkey = models.CharField(max_length=53, verbose_name=_("Public key"),␣
↪unique=True, null=True, blank=True)

    def __str__(self):
        return self.username
```

Here's a list of available settings:

Setting	Default	Description
SCATTERAUTH_SIGNUP_ENABLED	True	If False, new users won't be able to sign up (used in `signup_view`)
SCATTERAUTH_USER_SIGNUP_FIELDS	['email']	Specifies field to be used in signup form for a new User model
SCATTERAUTH_USER_PUBKEY_FIELD	'username'	Field on the User model, which has public key to check against.
SCATTERAUTH_DOMAIN	''	Determines what domain to use for signature verification. If '' - request.get_host() is used

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/Bearle/django-scatter-auth/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Bearle/django-scatter-auth/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/django-scatter-auth.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 django-scatter-auth
$ cd django-scatter-auth/
$ 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 scatterauth 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.

4.3 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/Bearle/django-scatter-auth/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_scatterauth
```

CHAPTER 5

Credits

5.1 Development Lead

- Denis Bobrov <tech@bearle.ru>

5.2 Special Thanks

- Nathan James <scatter.eos@gmail.com>

CHAPTER 6

History

6.1 0.2.0 (2020-03-14)

- Added support for Scatter-Desktop

6.2 0.1.1 (2018-09-10)

- Fixed signup bug in js - added 'pubkey_field_name' param

6.3 0.1.0 (2018-08-13)

- First release on PyPi