Flask Oidc 2

Release 1.5.0

Jul 22, 2020

Contents

Release : 1.5.0

Flask-Oidc2 is an extension to Flask that allows you to add OpenID Connect based authentication to your website in a matter of minutes. It depends on Flask and oauth2client. You can install the requirements from PyPI with *easy_install* or *pip* or download them by hand.

This library is a fork of the flask-oidc <https://github.com/puiterwijk/flask-oidc> library, and should work with any standards compliant OpenID Connect provider.

CHAPTER 1

Features

- Support for OpenID Connect 1.0
- Support for OpenID Connect Discovery 1.0
- Support for OpenID Connect Dynamic Registration 1.0
- Friendly API
- Perfect integration into Flask
- Helper functions to allow resource servers to accept OAuth2 tokens

CHAPTER 2

Installation

Install the extension with *pip*:

```
$ pip install flask-oidc2
```

CHAPTER 3

How to use

To integrate Flask-OpenID into your application you need to create an instance of the `OpenID` object first:

```
from flask_oidc import OpenIDConnect
oidc = OpenIDConnect(app)
```

Alternatively the object can be instantiated without the application in which case it can later be registered for an application with the *`init_app()`* method.

Note that you should probably provide the library with a place to store the credentials it has retrieved for the user. These need to be stored in a place where the user themselves or an attacker can not get to them. To provide this, give an object that has `__setitem__` and `__getitem__` dict APIs implemented as second argument to the `__init__()` call. Without this, the library will only work on a single thread, and only retain sessions until the server is restarted.

Using this library is very simple: you can use *`user_loggedin`* to determine whether a user is currently logged in using OpenID Connect.

If the user is logged in, you an use *`user_getfield()`* and *`user_getinfo()`* to get information about the currently logged in user.

If the user is not logged in, you can send them to any function that is decorated with *`require_login()`* to get them automatically redirected to login.

CHAPTER 4

Example

A very basic example client:

```python
@app.route('/')
def index():
    if oidc.user_loggedin:
        return 'Welcome %s' % oidc.user_getfield('email')
    else
        return 'Not logged in'

@app.route('/login')
@oidc.require_login
def login():
    return 'Welcome %s' % oidc.user_getfield('email')
```

CHAPTER 5

Custom callback

It is possible to override the default OIDC callback to keep track of a custom state dict through the OIDC authentication steps, which makes it possible to write stateless apps. To do this, add the decorator *oidc.custom_callback* to your callback function. This will get the (json-serializable) custom state that you passed in as *customstate* to *oidc.redirect_to_auth_server*. Note that to use this, you will need to set *OVERWRITE_REDIRECT_URI*.

Example:

```
@app.route('/')
def index():
    return oidc.redirect_to_auth_server(None, flask.request.values)

@app.route('/custom_callback')
@oidc.custom_callback
def callback(data):
    return 'Hello. You submitted %s' % data
```

CHAPTER 6

Resource server

Also, if you have implemented an API that can should be able to accept tokens issued by the OpenID Connect provider, just decorate those API functions with *accept_token()*:

```
@app.route('/api')
@oidc.accept_token()
def my_api():
    return json.dumps('Welcome %s' % g.oidc_token_info['sub'])
```

If you are only using this part of flask-oidc, it is suggested to set the configuration option *OIDC_RESOURCE_SERVER_ONLY* (new in 1.0.5).

CHAPTER 7

Registration

To be able to use an OpenID Provider, you will need to register your client with them. If the Provider you want to use supports Dynamic Registration, you can execute `oidc-register https://myprovider.example.com/ https://myapplication.example.com/` and the full client_secrets.json will be generated for you, and you are ready to start.

If it does not, please see the documentation of the Provider you want to use for information on how to obtain client secrets.

For example, for Google, you will need to visit Google API credentials management.

CHAPTER 8

Manual client registration

If your identity provider does not offer Dynamic Registration (and you can't push them to do so, as it would make it a lot simpler!), you might need to know the following details:

Grant type authorization_code (Authorization Code flow)

Response type Code

Token endpoint auth metod client_secret_post

Redirect URI <APPLICATION_URL>/oidc_callback

You will also need to manually craft your client_secrets.json. This is just a json document, with everything under a top-level "web" key. Underneath that top-level key, you have the following keys:

client_id Client ID issued by your IdP

client_secret Client secret belonging to the registered ID

auth_uri The Identity Provider's authorization endpoint url

token_uri The Identity Provider's token endpoint url (Optional, used for resource server)

userinfo_uri The Identity Provider's userinfo url

issuer The "issuer" value for the Identity Provider

redirect_uris A list of the registered redirect uris

CHAPTER 9

Settings reference

This is a list of all settings supported in the current release.

OIDC_SCOPES A python list with the scopes that should be requested. This impacts the information available in the UserInfo field and what the token can be used for. Please check your identity provider's documentation for valid values. Defaults to ['openid', 'email'].

OIDC_GOOGLE_APPS_DOMAIN The Google Apps domain that must be used to login to this application. Defaults to None, which means this check is skipped and the user can login with any Google account.

OIDC_ID_TOKEN_COOKIE_NAME Name of the cookie used to store the users' login state. Defaults to "oidc_id_token".

OIDC_ID_TOKEN_COOKIE_PATH Path under which the login state cookie is stored. Defaults to "/".

OIDC_ID_TOKEN_COOKIE_TTL Integer telling how long the login state of the user remains valid. Defaults to 7 days.

OIDC_COOKIE_SECURE Boolean indicating whether the cookie should be sent with the secure flag enabled. This means that a client (browser) will only send the cookie over an https connection. *Do NOT disable this in production please.* Defaults to True, indicating the cookie is marked as secure.

OIDC_VALID_ISSUERS The token issuer that is accepted. Please check your Identity Providers documentation for the correct value. Defaults to the value of "issuer" in client_secrets, or the Google issuer if not found.

OIDC_CLOCK_SKEW Number of seconds of clock skew allowed when checking the "don't use before" and "don't use after" values for tokens. Defaults to sixty seconds (one minute).

OIDC_REQUIRE_VERIFIED_EMAIL Boolean indicating whether the Identity Provider needs to mark the email as "verified". Defaults to False.

OIDC_OPENID_REALM String passed to the OpenID Connect provider to ask for the old OpenID identity for users. This helps when migrating from OpenID 2.0 to OpenID Connect because the Identity Provider will also add the OpenID 2.0 identity so you can tie them together. Defaults to None.

OIDC_USER_INFO_ENABLED Boolean whether to get user information from the UserInfo endpoint provided by the Identity Provider in addition to the token information. Defaults to True.

OIDC_CALLBACK_ROUTE URL relative to the web root to indicate where the oidc_callback url is mounted on. Defaults to /oidc_callback.

OVERWRITE_REDIRECT_URI URL to use as return url when passing to the Identity Provider. To be used when Flask could not detect the correct hostname, scheme or path to your application. Alternatively used when custom handler is to be used. Defaults to False (disabled).

OIDC_RESOURCE_SERVER_ONLY Boolean whether to disable the OpenID Client parts. You can enable this in applications where you only use the resource server parts (accept_token) and will skip checking for any cookies.

OIDC_RESOURCE_CHECK_AUD Boolean to indicate whether the current application needs to be the "audience" of tokens passed. Defaults to False.

OIDC_INTROSPECTION_AUTH_METHOD String that sets the authentication method used when communicating with the token_introspection_uri. Valid values are 'client_secret_post', 'client_secret_basic', or 'bearer'. Defaults to 'client_secret_post'.

CHAPTER 10

API References

The full API reference:

class flask_oidc.**OpenIDConnect**(*app=None*, *credentials_store=None*, *http=None*, *time=None*, *urandom=None*)

The core OpenID Connect client object.

accept_token(*require_token=False*, *scopes_required=None*, *render_errors=True*)

Use this to decorate view functions that should accept OAuth2 tokens, this will most likely apply to API functions.

Tokens are accepted as part of the query URL (access_token value) or a POST form value (access_token).

Note that this only works if a token introspection url is configured, as that URL will be queried for the validity and scopes of a token.

Parameters

- **require_token** (*bool*) – Whether a token is required for the current function. If this is True, we will abort the request if there was no token provided.
- **scopes_required** (*list*) – List of scopes that are required to be granted by the token before being allowed to call the protected function.
- **render_errors** (*callback(obj) or None*) – Whether or not to eagerly render error objects as JSON API responses. Set to False to pass the error object back unmodified for later rendering.

New in version 1.0.

authenticate_or_redirect()

Helper function suitable for @app.before_request and @check. Sets g.oidc_id_token to the ID token if the user has successfully authenticated, else returns a redirect object so they can go try to authenticate.

Returns A redirect object, or None if the user is logged in.

Return type Redirect

Deprecated since version 1.0: Use *require_login()* instead.

check(*view_func*)
Deprecated since version 1.0: Use `require_login()` instead.

custom_callback(*view_func*)
Wrapper function to use a custom callback. The custom OIDC callback will get the custom state field passed in with redirect_to_auth_server.

flow_for_request()
Deprecated since version 1.0: Use `require_login()` instead.

get_access_token()
Method to return the current requests' access_token.

Returns Access token or None

Return type str

New in version 1.2.

get_cookie_id_token()
Deprecated since version 1.0: Use `user_getinfo()` instead.

get_refresh_token()
Method to return the current requests' refresh_token.

Returns Access token or None

Return type str

New in version 1.2.

init_app(*app*)
Do setup that requires a Flask app.

Parameters **app** (*Flask*) – The application to initialize.

logout()
Request the browser to please forget the cookie we set, to clear the current session.

Note that as described in [1], this will not log out in the case of a browser that doesn't clear cookies when requested to, and the user could be automatically logged in when they hit any authenticated endpoint.

[1]: https://github.com/puiterwijk/flask-oidc/issues/5#issuecomment-86187023

New in version 1.0.

redirect_to_auth_server(*destination=None*, *customstate=None*)
Set a CSRF token in the session, and redirect to the IdP.

Parameters

- **destination** (*Url to return the client to if a custom handler is not used. Not available with custom callback.*) – The page that the user was going to, before we noticed they weren't logged in.
- **customstate** (*Anything that can be serialized*) – The custom data passed via the ODIC state. Note that this only works with a custom_callback, and this will ignore destination.

Returns A redirect response to start the login process.

Return type Flask Response

Deprecated since version 1.0: Use `require_login()` instead.

`require_keycloak_role`(*client*, *role*)
Function to check for a KeyCloak client role in JWT access token.

This is intended to be replaced with a more generic 'require this value in token or claims' system, at which point backwards compatibility will be added.

New in version 1.5.0.

`require_login`(*view_func*)
Use this to decorate view functions that require a user to be logged in. If the user is not already logged in, they will be sent to the Provider to log in, after which they will be returned.

New in version 1.0: This was `check()` before.

`set_cookie_id_token`(*id_token*)
Deprecated since version 1.0.

`user_getfield`(*field*, *access_token=None*)
Request a single field of information about the user.

Parameters **`field`** (`str`) – The name of the field requested.

Returns The value of the field. Depending on the type, this may be a string, list, dict, or something else.

Return type object

New in version 1.0.

`user_getinfo`(*fields*, *access_token=None*)
Request multiple fields of information about the user.

Parameters **`fields`** (`list`) – The names of the fields requested.

Returns The values of the current user for the fields requested. The keys are the field names, values are the values of the fields as indicated by the OpenID Provider. Note that fields that were not provided by the Provider are absent.

Return type dict

Raises **`Exception`** – If the user was not authenticated. Check this with user_loggedin.

New in version 1.0.

`user_loggedin`
Represents whether the user is currently logged in.

Returns: bool: Whether the user is logged in with Flask-OIDC.

New in version 1.0.

`validate_token`(*token*, *scopes_required=None*)
This function can be used to validate tokens.

Note that this only works if a token introspection url is configured, as that URL will be queried for the validity and scopes of a token.

Parameters **`scopes_required`** (`list`) – List of scopes that are required to be granted by the token before returning True.

Returns True if the token was valid and contained the required scopes. An ErrStr (subclass of string for which bool() is False) if an error occured.

Return type Boolean or String

New in version 1.1.

class flask_oidc.**MemoryCredentials**

Non-persistent local credentials store. Use this if you only have one app server, and don't mind making everyone log in again after a restart.

CHAPTER 11

Discovery

`flask_oidc.discovery.`**`discover_OP_information`**(*OP_uri*)

Discovers information about the provided OpenID Provider.

Parameters **`OP_uri`** (*`str`*) – The base URI of the Provider information is requested for.

Returns The contents of the Provider metadata document.

Return type dict

New in version 1.0.

CHAPTER 12

Registration

exception `flask_oidc.registration.`**`RegistrationError`**(*response*)

This class is used to pass errors reported by the OpenID Provider during dynamic registration.

New in version 1.0.

`flask_oidc.registration.`**`check_redirect_uris`**(*uris*, *client_type=None*)

This function checks all return uris provided and tries to deduce as what type of client we should register.

Parameters

- **`uris`** (*`list`*) – The redirect URIs to check.
- **`client_type`** (*`str`*) – An indicator of which client type you are expecting to be used. If this does not match the deduced type, an error will be raised.

Returns The deduced client type.

Return type str

Raises **`ValueError`** – An error occured while checking the redirect uris.

New in version 1.0.

`flask_oidc.registration.`**`register_client`**(*provider_info*, *redirect_uris*)

This function registers a new client with the specified OpenID Provider, and then returns the regitered client ID and other information.

Parameters

- **`provider_info`** (*`dict`*) – The contents of the discovery endpoint as specified by the OpenID Connect Discovery 1.0 specifications.
- **`redirect_uris`** (*`list`*) – The redirect URIs the application wants to register.

Returns An object containing the information needed to configure the actual client code to communicate with the OpenID Provider.

Return type dict

Raises

- **`ValueError`** – The same error as used by check_redirect_uris.
- ***`RegistrationError`*** – Indicates an error was returned by the OpenID Provider during registration.

New in version 1.0.

Python Module Index

Index