python-oauth2 Documentation

Release 0.5.0

Markus Meyer

September 28, 2014

Contents

python-oauth2 is a framework that aims at making it easy to provide authentication via OAuth 2.0 within an application stack.

CHAPTER 1

Usage

Example:

```
from wsgiref.simple_server import make_server
import oauth2
import oauth2.grant
import oauth2.error
import oauth2.store.memory
import oauth2.tokengenerator
import oauth2.web

# Create a SiteAdapter to interact with the user.
# This can be used to display confirmation dialogs and the like.
class ExampleSiteAdapter(oauth2.web.SiteAdapter):
    def authenticate(self, request, environ, scopes):
        if request.post_param("confirm") == "1":
            return {}

        raise oauth2.error.UserNotAuthenticated

    def render_auth_page(self, request, response, environ, scopes):
        response.body = '''
<html>
    <body>
        <form method="POST" name="confirmation_form">
            <input name="confirm" type="hidden" value="1" />
            <input type="submit" value="confirm" />
        </form>
    </body>
</html>'''
        return response

# Create an in-memory storage to store your client apps.
client_store = oauth2.store.memory.ClientStore()
# Add a client
client_store.add_client(client_id="abc", client_secret="xyz",
                        redirect_uris=["http://localhost/callback"])

# Create an in-memory storage to store issued tokens.
# LocalTokenStore can store access and auth tokens
token_store = oauth2.store.memory.TokenStore()

# Create the controller.
auth_controller = oauth2.Provider(
    access_token_store=token_store,
    auth_code_store=token_store,
    client_store=client_store,
    site_adapter=ExampleSiteAdapter(),
    token_generator=oauth2.tokengenerator.Uuid4()
)

# Add Grants you want to support
auth_controller.add_grant(oauth2.grant.AuthorizationCodeGrant())
auth_controller.add_grant(oauth2.grant.ImplicitGrant())

# Add refresh token capability and set expiration time of access tokens
# to 30 days
auth_controller.add_grant(oauth2.grant.RefreshToken(expires_in=2592000))

# Wrap the controller with the Wsgi adapter
app = oauth2.web.Wsgi(server=auth_controller)

if __name__ == "__main__":
    httpd = make_server('', 8080, app)
    httpd.serve_forever()
```

CHAPTER 2

Installation

python-oauth2 is available on PyPI:

```
pip install python-oauth2
```

Contents:

`oauth2.grant` — Grant classes and helpers

Grants are the heart of OAuth 2.0. Each Grant defines one way for a client to retrieve an authorization. They are defined in Section 4 of the OAuth 2.0 spec.

OAuth 2.0 comes in two flavours of how an access token is issued: two-legged and three-legged auth. To avoid confusion they are explained in short here.

3.1 Three-legged OAuth

The "three" symbolizes the parties that are involved:

- The client that wants to access a resource on behalf of the user.
- The user who grants access to her resources.
- The server that issues the access token if the user allows it.

3.2 Two-legged OAuth

The two-legged OAuth process differs from the three-legged process by one missing paricipant. The user cannot allow or deny access.

So there are two remaining parties:

- The client that wants to access a resource.
- The server that issues the access.

3.3 Helpers and base classes

class `oauth2.grant.`**`GrantHandlerFactory`**

Base class every handler factory can extend.

This class defines the basic interface of each Grant.

class `oauth2.grant.`**`ScopeGrant`**(*default_scope=None*, *scopes=None*, *scope_class=<class 'oauth2.grant.Scope'>*)

Handling of scopes in the OAuth 2.0 flow.

Inherited by all grants that need to support scopes.

Parameters

- **default_scope** – The scope identifier that is returned by default. (optional)
- **scopes** – A list of strings identifying the scopes that the grant supports.
- **scope_class** – The class that does the actual handling in a request. Default: `oauth2.grant.Scope`.

class `oauth2.grant.`**`Scope`**(*available=None*, *default=None*)

Handling of the “scope” parameter in a request.

If `available` and `default` are both `None`, the “scope” parameter is ignored (the default).

Parameters

- **available** – A list of strings each defining one supported scope.
- **default** – Value to fall back to in case no scope is present in a request.

`Scope.`**`parse`**(*request*, *source*)

Parses scope value in given request.

Expects the value of the “scope” parameter in request to be a string where each requested scope is separated by a white space:

```
# One scope requested
"profile_read"

# Multiple scopes
"profile_read profile_write"
```

Parameters

- **request** – An instance of `oauth2.web.Request`.
- **source** – Where to read the scope from. Pass “body” in case of a application/x-www-form-urlencoded body and “query” in case the scope is supplied as a query parameter in the URL of a request.

3.4 Grant classes

class `oauth2.grant.`**`AuthorizationCodeGrant`**(*default_scope=None*, *scopes=None*, *scope_class=<class 'oauth2.grant.Scope'>*)

Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`

Implementation of the Authorization Code Grant auth flow.

This is a three-legged OAuth process.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()

auth_controller.add_grant_type(AuthorizationCodeGrant())
```

class `oauth2.grant.`**`ImplicitGrant`**(*default_scope=None*, *scopes=None*, *scope_class=<class 'oauth2.grant.Scope'>*)

Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`

Implementation of the Implicit Grant auth flow.

This is a three-legged OAuth process.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()

auth_controller.add_grant_type(ImplicitGrant())
```

class `oauth2.grant.`**`ResourceOwnerGrant`**(*default_scope=None*, *scopes=None*, *scope_class=<class 'oauth2.grant.Scope'>*)

Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`

Implementation of the Resource Owner Password Credentials Grant auth flow.

In this Grant a user provides a user name and a password. An access token is issued if the auth server was able to verify the user by her credentials.

Register an instance of this class with `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()

auth_controller.add_grant_type(ResourceOwnerGrant())
```

class `oauth2.grant.`**`RefreshToken`**(*expires_in*, *default_scope=None*, *scopes=None*, *scope_class=<class 'oauth2.grant.Scope'>*)

Bases: `oauth2.grant.GrantHandlerFactory`, `oauth2.grant.ScopeGrant`

Handles requests for refresk tokens as defined in http://tools.ietf.org/html/rfc6749#section-6.

Adding a Refresh Token to the `oauth2.AuthorizationController` like this:

```
auth_controller = AuthorizationController()

auth_controller.add_grant_type(RefreshToken(expires_in=600))
```

will cause `oauth2.grant.AuthorizationCodeGrant` and `oauth2.grant.ResourceOwnerGrant` to include a refresh token and expiration in the response.

CHAPTER 4

`oauth2.store` — Storing and retrieving data

Store adapters to persist and retrieve data during the OAuth 2.0 process or for later use. This module provides base classes that can be extended to implement your own solution specific to your needs. It also includes implementations for popular storage systems like memcache.

4.1 Data types

class `oauth2.datatype.`**`AccessToken`**(*client_id*, *grant_type*, *token*, *data={}*, *expires_at=None*, *refresh_token=None*, *scopes=[ ]*)

An access token and associated data.

class `oauth2.datatype.`**`AuthorizationCode`**(*client_id*, *code*, *expires_at*, *redirect_uri*, *scopes*, *data=None*)

Holds an authorization code and additional information.

class `oauth2.datatype.`**`Client`**(*identifier*, *secret*, *redirect_uris=[ ]*)

Representation of a client application.

4.2 Base classes

class `oauth2.store.`**`AccessTokenStore`**

Base class for persisting an access token after it has been generated.

Used in two-legged and three-legged authentication flows.

`fetch_by_refresh_token`(*refresh_token*)

Fetches an access token from the store using its refresh token to identify it.

Parameters **refresh_token** – A string containing the refresh token.

`save_token`(*access_token*)

Stores an access token and additional data.

Parameters **access_token** – An instance of `oauth2.datatype.AccessToken`.

class `oauth2.store.`**`AuthCodeStore`**

Base class for writing and retrieving an auth token during the Authorization Code Grant flow.

`fetch_by_code`(*code*)

Returns an AuthorizationCode fetched from a storage.

Parameters **code** – The authorization code.

Returns An instance of `oauth2.datatype.AuthorizationCode`.

Raises `AuthCodeNotFound` if no data could be retrieved for given code.

save_code (*authorization_code*)
Stores the data belonging to an authorization code token.

Parameters **authorization_code** – An instance of `oauth2.AuthorizationCode`.

class `oauth2.store.`**ClientStore**
Base class for handling OAuth2 clients.

fetch_by_client_id (*client_id*)
Retrieve a client by its identifier.

Parameters **client_id** – Identifier of a client app.

Returns An instance of `oauth2.Client`.

Raises ClientNotFoundError

4.3 Implementations

4.3.1 `oauth2.store.memcache` — Memcache store adapters

class `oauth2.store.memcache.`**TokenStore** (*mc=None*, *prefix='oauth2'*, **args*, ***kwargs*)
Uses memcache to store access tokens and auth tokens.

This Store supports `python-memcached`. Arguments are passed to the underlying client implementation.

Initialization by passing an object:

```
# This example uses python-memcached
import memcache

# Somewhere in your application
mc = memcache.Client(servers=['127.0.0.1:11211'], debug=0)
# ...
token_store = TokenStore(mc=mc)
```

Initialization using `python-memcached`:

```
token_store = TokenStore(servers=['127.0.0.1:11211'], debug=0)
```

4.3.2 `oauth2.store.memory` — In-memory store adapters

Read or write data from or to local memory.

Though not very valuable in a production setup, these store adapters are great for testing purposes.

class `oauth2.store.memory.`**ClientStore**
Stores clients in memory.

add_client (*client_id*, *client_secret*, *redirect_uris*)
Add a client app.

Parameters

- **client_id** – Identifier of the client app.

- **client_secret** – Secret the client app uses for authentication against the OAuth 2.0 provider.
- **redirect_uris** – A `list` of URIs to redirect to.

`fetch_by_client_id`(*client_id*)
Retrieve a client by its identifier.

Parameters **client_id** – Identifier of a client app.

Returns An instance of `oauth2.Client`.

Raises ClientNotFoundError

class `oauth2.store.memory.`**`TokenStore`**
Stores tokens in memory.

Useful for testing purposes or APIs with a very limited set of clients. Use memcache or redis as storage to be able to scale.

`fetch_by_code`(*code*)
Returns an AuthorizationCode.

Parameters **code** – The authorization code.

Returns An instance of `oauth2.datatype.AuthorizationCode`.

Raises `AuthCodeNotFound` if no data could be retrieved for given code.

`fetch_by_refresh_token`(*refresh_token*)
Find an access token by its refresh token.

Parameters **refresh_token** – The refresh token that was assigned to an `AccessToken`.

Returns The `oauth2.datatype.AccessToken`.

Raises `oauth2.error.AccessTokenNotFound`

`fetch_by_token`(*token*)
Returns data associated with an access token or `None` if no data was found.

Useful for cases like validation where the access token needs to be read again.

Parameters **token** – A access token code.

Returns An instance of `oauth2.datatype.AccessToken`.

`save_code`(*authorization_code*)
Stores the data belonging to an authorization code token.

Parameters **authorization_code** – An instance of `oauth2.datatype.AuthorizationCode`.

`save_token`(*access_token*)
Stores an access token and additional data in memory.

Parameters **access_token** – An instance of `oauth2.datatype.AccessToken`.

4.3.3 `oauth2.store.mongodb` — Mongodb store adapters

Store adapters to read/write data to from/to mongodb using pymongo.

class `oauth2.store.mongodb.`**`MongodbStore`**(*collection*)
Base class extended by all concrete store adapters.

class `oauth2.store.mongodb.`**`AccessTokenStore`**(*collection*)
Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = AccessTokenStore(collection=db["access_tokens"])
```

class `oauth2.store.mongodb.`**`AuthCodeStore`**(*collection*)

Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = AuthCodeStore(collection=db["auth_codes"])
```

class `oauth2.store.mongodb.`**`ClientStore`**(*collection*)

Create a new instance like this:

```
from pymongo import MongoClient

client = MongoClient('localhost', 27017)

db = client.test_database

access_token_store = ClientStore(collection=db["clients"])
```

CHAPTER 5

oauth2 — Provider class

class oauth2.**Provider**(*access_token_store*, *auth_code_store*, *client_store*, *site_adapter*, *token_generator*, *response_class=<class 'oauth2.web.Response'>*)

Provider.**add_grant**(*grant*)
Adds a Grant that the provider should support.

Provider.**dispatch**(*request*, *environ*)
Checks which Grant supports the current request and dispatches to it.

Parameters

- **request** – An instance of `oauth2.web.Request`.
- **environ** – Hash containing variables of the environment.

Returns An instance of `oauth2.web.Response`.

CHAPTER 6

`oauth2.web` — Interaction over HTTP

Classes for handling a HTTP request/response flow.

class `oauth2.web.`**`SiteAdapter`**

Interact with a user.

Display HTML or redirect the user agent to another page of your website where she can do something before being returned to the OAuth 2.0 server.

`SiteAdapter.`**`authenticate`**`(`*request*, *environ*, *scopes*`)`

Authenticates a user and checks if she has authorized access.

Parameters

- **request** – An instance of `oauth2.web.Request`.
- **environ** – Environment variables of the request.
- **scopes** – A list of strings with each string being one requested scope.

Returns A `dict` containing arbitrary data that will be passed to the current storage adapter and saved with auth code and access token.

Raises `oauth2.error.UserNotAuthenticated` if the user could not be authenticated.

`SiteAdapter.`**`render_auth_page`**`(`*request*, *response*, *environ*`)`

Defines how to display a confirmation page to the user.

Parameters

- **request** – An instance of `oauth2.web.Request`.
- **response** – An instance of `oauth2.web.Response`.
- **environ** – Environment variables of the request.

Returns The response passed in as a parameter. It can contain HTML or issue a redirect.

`SiteAdapter.`**`user_has_denied_access`**`(`*request*`)`

Checks if the user has denied access. This will lead to python-oauth2 returning a "acess_denied" response to the requesting client app.

Parameters **request** – An instance of `oauth2.web.Request`.

Returns Return `True` if the user has denied access.

class `oauth2.web.`**`Request`**`(`*env*`)`

Contains data of the current HTTP request.

`Request.`**`get_param`**`(`*name*, *default=None*`)`
Returns a param of a GET request identified by its name.

`Request.`**`post_param`**`(`*name*, *default=None*`)`
Returns a param of a POST request identified by its name.

class `oauth2.web.`**`Response`**
Contains data returned to the requesting user agent.

`oauth2.error` — Error classes

Errors raised during the OAuth 2.0 flow.

class `oauth2.error.`**`AuthCodeNotFound`**

Error indicating that an authorization code could not be read from the storage backend by an instance of `oauth2.store.AuthCodeStore`.

class `oauth2.error.`**`ClientNotFoundError`**

Error raised by an implementation of `oauth2.store.ClientStore` if a client does not exists.

class `oauth2.error.`**`OAuthBaseError`**(*error*, *error_uri=None*, *explanation=None*)

Base class used by all OAuth 2.0 errors.

Parameters

- **error** – Identifier of the error.
- **error_uri** – Set this to delivery an URL to your documentation that describes the error. (optional)
- **explanation** – Short message that describes the error. (optional)

class `oauth2.error.`**`OAuthClientError`**(*error*, *error_uri=None*, *explanation=None*)

Indicates an error during recognition of a client.

class `oauth2.error.`**`OAuthUserError`**(*error*, *error_uri=None*, *explanation=None*)

Indicates that the user denied authorization.

class `oauth2.error.`**`OAuthInvalidError`**(*error*, *error_uri=None*, *explanation=None*)

Indicates an error during validation of a request.

class `oauth2.error.`**`UserNotAuthenticated`**

Raised by a `oauth2.web.SiteAdapter` if a user could not be authenticated.

CHAPTER 8

Using python-oauth2 with other frameworks

8.1 Flask

Wrapper classes to integrate an OAuth 2.0 Authorization Server into a Flask application:

```python
from flask import request, Flask
from oauth2 import AuthorizationController
from oauth2.store import LocalClientStore, LocalTokenStore
from oauth2.tokengenerator import Uuid4
from oauth2.web import SiteAdapter
from oauth2.grant import AuthorizationCodeGrant

class Request(object):
    """
    Simple wrapper around the Flask request object
    """
    @property
    def path(self):
        return request.path

    def get_param(self, name, default=None):
        return request.args.get(key=name, default=default)

    def post_param(self, name, default=None):
        return request.form.get(key=name, default=default)

class OAuth2(object):
    """
    Extend your Flask application to serve OAuth 2.0.
    """
    def __init__(self, access_token_store,
                 auth_token_store,
                 client_store,
                 site_adapter,
                 token_generator,
                 app=None,
                 authorize_path="/authorize",
                 token_path="/token"):
        self.access_token_store = access_token_store
        self.auth_token_store   = auth_token_store
        self.client_store       = client_store
        self.site_adapter       = site_adapter
        self.token_generator    = token_generator
        self.authorize_path      = authorize_path
        self.token_path          = token_path

        if app is not None:
            self.init_app(app)
        else:
            self.app = None

    def add_grant(self, grant):
        """
        Add a grant that your auth server shall support.
        """
        self.controller.add_grant(grant)

    def init_app(self, app):
        """
        Initializes view functions.
        """
        self.app = app

        self.controller = AuthorizationController(
            access_token_store=self.access_token_store,
            auth_token_store=self.auth_token_store,
            client_store=self.client_store,
            site_adapter=self.site_adapter,
            token_generator=self.token_generator)

        self.controller.authorize_path = self.authorize_path
        self.controller.token_path = self.token_path

        self.app.add_url_rule(self.authorize_path, "authorize", self._dispatch,
                              methods=["GET", "POST"])
        self.app.add_url_rule(self.token_path, "token", self._dispatch,
                              methods=["GET", "POST"])

    def _dispatch(self):
        assert self.controller is not None

        response = self.controller.dispatch(Request(), environ={})

        return response.body, response.status_code, response.headers

class MySiteAdapter(SiteAdapter):
    def authenticate(self, request, environ, scopes):
        # Authenticate every request
        return {}

def main():
    app = Flask(__name__)

    # Initialize storage
    client_store = LocalClientStore()
    client_store.add_client(client_id="abc", client_secret="xyz",
                            redirect_uris=["http://localhost:8081/callback"])

    token_store = LocalTokenStore()

    oauth_app = OAuth2(app=app, access_token_store=token_store,
                             auth_token_store=token_store, client_store=client_store,
                             site_adapter=MySiteAdapter(), token_generator=Uuid4())

    oauth_app.add_grant(AuthorizationCodeGrant())

    app.run(port=5000, debug=True)

    if __name__ == "__main__":
        main()
```

8.2 Tornado

Use Tornado to serve token requests:

```
import tornado.web
import tornado.ioloop
from oauth2.store import LocalClientStore, LocalTokenStore
from oauth2 import AuthorizationController
from oauth2.tokengenerator import Uuid4
from oauth2.web import SiteAdapter
from oauth2.grant import ImplicitGrant, AuthorizationCodeGrant

class Request(object):
    """
    Wraps ``tornado.web.RequestHandler``.
    """
    def __init__(self, request_handler):
        self.request_handler = request_handler
        self.path = request_handler.request.path

    def get_param(self, name, default=None):
        return self._read_argument(name, default, source="GET")

    def post_param(self, name, default=None):
        return self._read_argument(name, default, source="POST")

    def _read_argument(self, name, default, source):
        if self.request_handler.request.method != source:
            return None
        try:
            return self.request_handler.get_argument(name)
        except tornado.web.MissingArgumentError:
            return default

class OAuth2Handler(tornado.web.RequestHandler):
    """
    Dispatches requests to an authorization controller
    """
    def initialize(self, controller):
        self.controller = controller

    def get(self):
        response = self._dispatch_request()

        self._map_response(response)
    def post(self):
        response = self._dispatch_request()

        self._map_response(response)

    def _dispatch_request(self):
        request = Request(request_handler=self)

        return self.controller.dispatch(request, environ={})

    def _map_response(self, response):
        for name, value in list(response.headers.items()):
            self.set_header(name, value)

        self.set_status(response.status_code)
        self.write(response.body)

class MySiteAdapter(SiteAdapter):
    def authenticate(self, request, environ, scopes):
        # Authenticate every request
        return {}

def main():
    # Initialize AuthorizationController as usual
    client_store = LocalClientStore()
    client_store.add_client(client_id="abc", client_secret="xyz",
                            redirect_uris=["http://localhost:8081/callback"])

    token_store = LocalTokenStore()

    auth_controller = AuthorizationController(
        access_token_store=token_store,
        auth_token_store=token_store,
        client_store=client_store,
        site_adapter=MySiteAdapter(),
        token_generator=Uuid4()
    )

    auth_controller.add_grant(AuthorizationCodeGrant())
    auth_controller.add_grant(ImplicitGrant())

    # Create your Tornado application and add the handler
    app = tornado.web.Application([
        (r'/authorize', OAuth2Handler, dict(controller=auth_controller))
    ])

    # Start the server
    app.listen(8888)
    tornado.ioloop.IOLoop.instance().start()

if __name__ == "__main__":
    main()
```

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index