steemconnect-python-client Documentation

Release 0.0.4

emrebeyler

Apr 14, 2020

Contents

hivesigner-python-client is a simple **yet powerful** library to interact with the hivesigner. hivesigner is a central-single sign on solution for HIVE based applications.

hivesigner implements Oauth2 for the authorization logic.

CHAPTER 1

What can you do with this client?

- Implementing Authorization/Authentication flow through OAuth
- Broadcasting supported operations to the HIVE blockchain with the user of your app.

CHAPTER 2

Installation

hivesigner-python-client requires python3.6 and above. Even though it's easy to make it compatible with lower versions, it's doesn't have support by design to keep the library simple.

You can install the library by typing to your console:

```
$ (sudo) pip install hivesigner
```

After that, you can continue with *Getting Started*.

CHAPTER 3

Documentation Pages

3.1 Getting Started

Hivesigner supports two different oauth flows for authorization.

3.1.1 Implicit grant flow

- You create an authorization link with your app's client id and permission scopes.
- User visits the page (Hivesigner) and authorizes the application with the given permission scopes.
- Hivesigner returns user to the your app with a access token in the query string.

From that point, you can use this access token to broadcast operations on user's behalf.

3.1.2 Code authorization flow

access tokens has a short TTL on Oauth standards. Every time a user has their token expired, you have two choices:

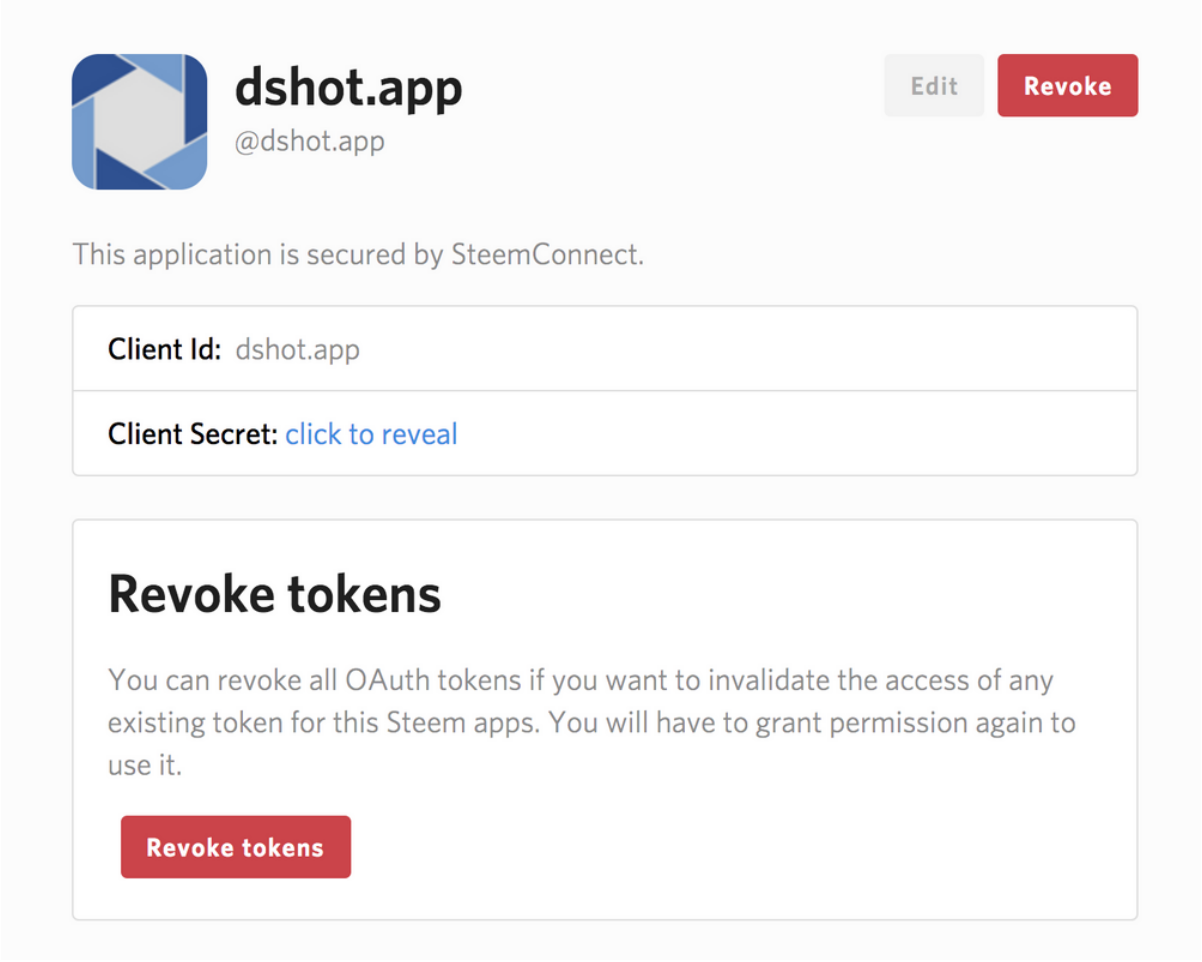
- Re-log the user and get a fresh token
- Add "offline" scope to the required scopes and get a refresh token to refresh the access tokens.

The second approach is required on some cases for the dApps and you will need to use Code authorization flow this. When you add the "offline" scope to the required scopes, you will get a code instead of access token.

With this code, you can get new access tokens till forever. *(As long as the user don't revoke access of your app.)*

3.1.3 Creating your app on Hivesigner

You need to register your app into Hivesigner before working with them. This will provide client_id and client_secret information which you will need to interact with the API.


dshot.app
@dshot.app
Edit
Revoke
This application is secured by SteemConnect.
Client Id: dshot.app
Client Secret: click to reveal
Revoke tokens
You can revoke all OAuth tokens if you want to invalidate the access of any existing token for this Steem apps. You will have to grant permission again to use it.
Revoke tokens

3.1.4 Redirecting user

```
from hivesigner.client import Client

c = Client(
    client_id="app_name",
    client_secret="client_secret",
)
```

At this point, we need to redirect the user to Hivesigner for they to log in. That requires creating a URL.

```
auth_url = c.get_login_url(
    "http://callback.to.your.app",
    "login,vote",
)
```

- The first parameter is the callback URL when the user authorizes your app on Hivesigner.
- Second parameter defines the scopes you need.

Important: If you need to use the Code authorization flow, you need to pass **get_refresh_token=True** to this function. Also, "offline" scope is mandatory.

Once the user authorizes your app, Hivesigner will redirect the user to your app with an access token or code depending the flow you choose. If you get a **code** in the query string, you can use this code to create access tokens for the specified user.

```
c.get_access_token(
    code,
)
```

Example output

```
{
    'access_token': 'access_token_string',
    'expires_in': 604800,
    'username': 'emrebeyler',
    'refresh_token': 'refresh_token_string'
}
```

If you use the Implicit grant flow, then you may skip this step.

Continue with *Using access tokens* to learn what can you do with the access tokens.

3.2 Using access tokens

Once you get the access token, you can create a new Client instance with just access_token.

```
c = Client(
    access_token="<access_token>",
)
```

3.2.1 Getting authorized user's information

This api call gives information about the authorized user.

```
print(c.me())
```

3.2.2 Updating user profile (metadata)

```
metadata = {
    "profile": {
        "name": "Emre",
        "location": "Istanbul, Turkey",
        "about": "Developer, HIVE witness.",
        "profile_image": "http://foo.bar/image.png"
    }
}

resp = c.update_user_metadata(metadata)
```

3.2.3 Broadcasting operations

It's possible to

- vote a post
- create a post/comment
- follow/unfollow/ignore/resteem
- claim reward balance
- delete comment
- create custom jsons

via steemconnect's broadcast apis.

Note: All operations live inside the **hivesigner.operations** module. You need to import the corresponding classes before using them.

3.2.4 Voting for a post

```
vote = Vote("account", "author", "permlink", percent)
c.broadcast([vote.to_operation_structure()])
```

3.2.5 Creating a comment/post

```
comment = Comment(
    "author",
    "permlink",
    "body",
```

(continues on next page)

(continued from previous page)

```
    title="test title",
    json_metadata={"app":"foo/0.0.1"},
)
c.broadcast([comment.to_operation_structure()])
```

3.2.6 Creating a comment/post with CommentOptions

```
comment = Comment(
    "author",
    "permlink",
    "body",
    title="test title",
    json_metadata={"app":"foo/0.0.1"},
)

comment_options = CommentOptions(
      parent_comment=comment,
    allow_curation_rewards=False,
)

c.broadcast([
    comment.to_operation_structure(),
    comment_options.to_operation_structure()
])
```

3.2.7 Follow an account

```
follow = Follow("follower", "following")
c.broadcast([follow.to_operation_structure()])
```

3.2.8 Unfollow an account

```
unfollow = Unfollow("follower", "following")
c.broadcast([unfollow.to_operation_structure()])
```

3.2.9 Mute an account

```
ignore = Mute("follower", "following")
c.broadcast([ignore.to_operation_structure()])
```

3.2.10 Reblog a post

```
reblog = Reblog("account", "author", "permlink")
c.broadcast([reblog.to_operation_structure()])
```

3.2.11 Claim reward balance

```
claim_reward_balance = ClaimRewardBalance('account', '0.000 HIVE', '1.500 HBD', '1132.
↪996000 VESTS')
c.broadcast([claim_reward_balance.to_operation_structure()])
```

3.2.12 Delete comment

```
delete_comment = DeleteComment(
    "author", "permlink"
)
c.broadcast([delete_comment.to_operation_structure()])
```

3.2.13 Create custom jsons

```
custom_json = CustomJson(
    required_auth,
    required_posting_auths,
    id
    json_structure,
)
c.broadcast([custom_json.to_operation_structure()])
```

3.3 Hot signing

client's hot_sign() method creates a SteemConnect specific URL which you can redirect users and expect them to broadcast operations are not supported in the api. (transfer, create_delegation, etc.)

hot_sign(self, operation, params, redirect_uri=None):

Parameters

- **operation** – String. Operation name. Ex: transfer.
- **params** – Dict. Operation data.
- **redirect_uri** – String. Optional. If you pass that, SteemConnect will redirect

the user to that URL after the operation succeeds.

3.3.1 Example: A transfer to emrebeyler with 1 HBD

```
url = self.c.hot_sign(
    "transfer",
    {
        "to": "emrebeyler",
        "amount": "1 HBD",
        "memo": "Donation",
    },
    redirect_uri="http://localhost"
)
```

3.4 Example flask application

This simple flask application redirects the user to steemconnect for the authorization. Once the user authorizes your app, it calls /me endpoint and gives a warm welcome message with the "name" property of the user.

```
from flask import Flask, request
from steemconnect.client import Client

app = Flask(__name__)


client_id = "your.app"
client_secret = "your_secret"

c = Client(client_id=client_id, client_secret=client_secret)


@app.route('/')
def index():
    login_url = c.get_login_url(
        "http://localhost:5000/welcome",
        "login",
    )
    return "<a href='%s'>Login with SteemConnect</a>" % login_url


@app.route('/welcome')
def welcome():
    c.access_token = request.args.get("access_token")
    return "Welcome <strong>%s</strong>!" % c.me()["name"]
```