

Computer Assignment 3

Narjes Noorzad

June 22, 2021

1 INTRODUCTION

OAuth, which stands for *Open Authorization*, allows third-party services to exchange your information without you having to give away your password.

OAuth can be summerized in 6 main steps :

- **Step 1** : The User Shows Intent
- **Step 2** : The Consumer Gets Permission
- **Step 3** : The User Is Redirected to the Service Provider
- **Step 4** : The User Gives Permission
- **Step 5** : The Consumer Obtains an Access Token
- **Step 6** : The Consumer Accesses the Protected Resource

2 STEP BY STEP GUIDELINE

2.1 OAuth app

After logging into Github (our *Authorization Server*), an *OAuth app* needs to be created. Homepage URL is a localhost page on our device listening on port 8589 which will act as a login page.

Authorization callback URL is <http://localhost:8589/oauth/redirect> which is the address where the OAuth app is redirected after communicating with the authorization server on GitHub.

Register a new OAuth application

Application name *

OAuth app

Something users will recognize and trust.

Homepage URL *

http://localhost:3589

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

http://localhost:3589/oauth/redirect


Your application's callback URL. Read our [OAuth documentation](#) for more information.

Register application Cancel

Figure 2.1: OAuth app on Github

After establishing our *OAuth app*, a *Client ID* is assigned to us.

OAuth app

 narjesno owns this application. [Transfer ownership](#)

You can list your application in the [GitHub Marketplace](#) so that other users can discover it. [List this application in the Marketplace](#)

0 users [Revoke all user tokens](#)

Client ID

935b318ef8b45a2f42ab

Client secrets [Generate a new client secret](#)



You need a client secret to authenticate as the application to the API.

Figure 2.2: OAuth app

Now we need to generate a new client secret. This is actually the shared key between the authorization server and the web application for our client authentication purpose in order to avoid phishing of any kind.

Client secrets [Generate a new client secret](#)

Make sure to copy your new client secret now. You won't be able to see it again.

 **bb5c6b27d05909f215a25d2d1dc1830b2f2b0b80** 

Client secret

Added now by narjesno
Never used
You cannot delete the only client secret. Generate a new client secret first. [Delete](#)

Figure 2.3: Client secret

2.2 Code Modifications + Bonus part

Authorization server needs two main parameters in order to authorize the client (web application in our case) which are *Client ID* and *Client secret*.

A simple login page is also designed for redirection puposes which needs both *Authorization URL* and *Redirect URL*.

```
const port = 8589;
const client_ID = '935b318ef8b45a2f42ab'
const client_secret = 'bb5c6b27d05909f215a25d2d1dc1830b2f2b0b80'
const authZ_url = 'https://github.com/login/oauth/authorize';
const redirect_url = 'http://localhost:8589/oauth/redirect';
```

Figure 2.4: Added variables

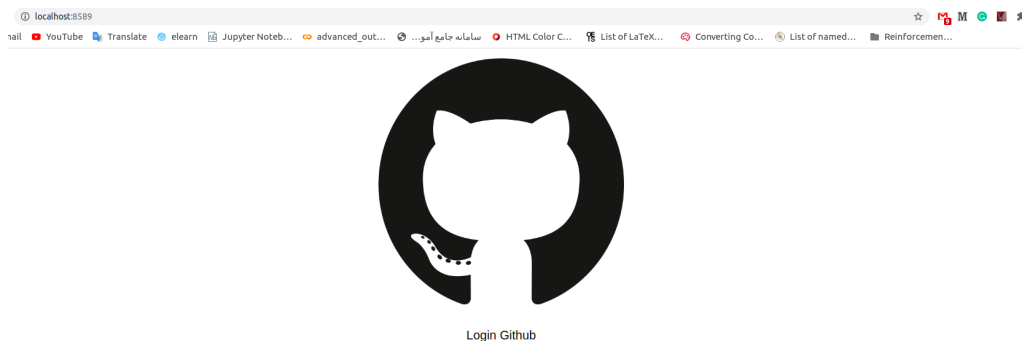


Figure 2.5: Login page

By clicking on the *Login Github* button, localhost sends an authorization request to the authorization server on GitHub via the Authorization URI. It also has a redirect URL, so it knows where to come back to. After the log-in is done and the user is authenticated, it is redirected to *Web Application*. This is where the *Code* is granted to *Web Application*. This *Code* is achieved by sending an HTTP *GET* request to the user.

```
(base) narjesnz@asus:~/Desktop/CA3/server$ node server.js
server is listening on port: 8589
Github code is: aa8afce13036a4cc995c
Request token is: aa8afce13036a4cc995c
Access token is: gho_52uVLK1VLvbkL1jXbm3ewRbBfL2mbK26uyqJ
```

Figure 2.6: Code, Request token, Access token

The Authorization server on GitHub receives 3 parameters sent by the client, *Client ID*, *Client Secret* and the *Grant code* a.k.a *Access token*.

Client Secret and the *Grant code* should be checked; The *Grant code* should be the same as the one the server handed to the user and the *Client Secret* should be the same as the secret key associated to the web application at the server's database.

We also want the sending and receiving of the request to be done automatically so other modifications are ahead.

Axios module was used as a tool for HTTP requests.

We need to use 2 methods for this purpose. First, the *POST* method was employed .

3 essential information was handed to <http://github.com/login/oauth/access-token> : *Client ID*, *Client Secret* and the *Request token (Grant code)*. The output is a *Response token* which we get our *Access token* from.

```
app.get('/oauth/redirect', async (req, res) => {
  console.log('Github code is: ${req.query.code}');

  const res_token = await axios({
    method: 'post',
    url: 'https://github.com/login/oauth/access_token?' + `client_id = ${client_ID}&` + `client_secret = ${client_secret}&` + `code = ${req.query.code}`,
    headers: {accept: 'application/json'} });

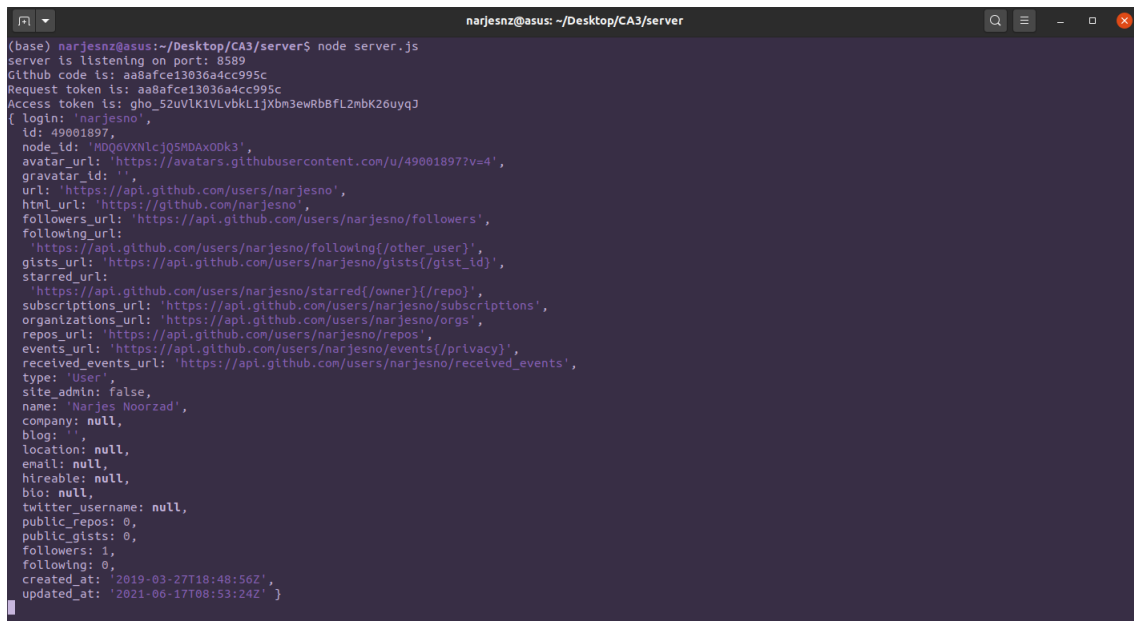
  console.log('Waiting for access token ...');
  console.log('Access token is: ${res_token.data.access_token}');

  const api_result = await axios({
    method: 'get',
    url: 'https://api.github.com/user',
    headers: { Authorization: `token ${res_token.data.access_token}` }
  });
  console.log('Waiting for API result ...');
  console.log('Just a moment ...');
  console.log(api_result.data);
  res.send(api_result.data);
});
```

Figure 2.7: Code

Then , we use the *GET* method. By employing this method, we are actually sending a GET request and by having the *Access token* gained from last part, we can access user's information form GitHub's API .

In figures below, we can see the result. We have all the information needed :



```
(base) narjesnz@asus:~/Desktop/CA3/server$ node server.js
server is listening on port: 8589
Github code is: aa8afce13836a4cc995c
Request token is: aa8afce13836a4cc995c
Access token is: gho_52uVLK1VLvbkL1jXbm3ewRbBFL2mbK26uyqJ
{ login: 'narjesno',
  id: 49001897,
  node_id: 'MDQ6VXNlcjQ5MDAxODk3',
  avatar_url: 'https://avatars.githubusercontent.com/u/49001897?v=4',
  gravatar_id: '',
  url: 'https://api.github.com/users/narjesno',
  html_url: 'https://github.com/narjesno',
  followers_url: 'https://api.github.com/users/narjesno/followers',
  following_url: 'https://api.github.com/users/narjesno/following{/other_user}',
  gists_url: 'https://api.github.com/users/narjesno/gists{/gist_id}',
  starred_url: 'https://api.github.com/users/narjesno/starred{/owner}/{repo}',
  subscriptions_url: 'https://api.github.com/users/narjesno/subscriptions',
  organizations_url: 'https://api.github.com/users/narjesno/orgs',
  repos_url: 'https://api.github.com/users/narjesno/repos',
  events_url: 'https://api.github.com/users/narjesno/events{/privacy}',
  received_events_url: 'https://api.github.com/users/narjesno/received_events',
  type: 'User',
  site_admin: false,
  name: 'Narjes Noorzad',
  company: null,
  blog: '',
  location: null,
  email: null,
  hireable: null,
  bio: null,
  twitter_username: null,
  public_repos: 0,
  public_gists: 0,
  followers: 1,
  following: 0,
  created_at: '2019-03-27T18:48:56Z',
  updated_at: '2021-06-17T08:53:24Z' }
```

Figure 2.8: User's information - terminal format

```
{ "login": "narjesno", "id": 49001897, "node_id": "MDQ6VXNlcjQ5MDAxODk3", "avatar_url": "https://avatars.githubusercontent.com/u/49001897?v=4", "gravatar_id": "", "url": "https://api.github.com/users/narjesno", "html_url": "https://github.com/narjesno", "followers_url": "https://api.github.com/users/narjesno/followers", "following_url": "https://api.github.com/users/narjesno/following/{other_user}", "gists_url": "https://api.github.com/users/narjesno/gists/{gist_id}", "starred_url": "https://api.github.com/users/narjesno/starred/{owner}/{repo}", "subscriptions_url": "https://api.github.com/users/narjesno/subscriptions", "organizations_url": "https://api.github.com/users/narjesno/orgs", "repos_url": "https://api.github.com/users/narjesno/repos", "events_url": "https://api.github.com/users/narjesno/events/{privacy}", "received_events_url": "https://api.github.com/users/narjesno/received_events", "type": "User", "site_admin": false, "name": "Narjes Noorzad", "company": null, "blog": "", "location": null, "email": null, "hireable": null, "bio": null, "twitter_username": null, "public_repos": 0, "public_gists": 0, "followers": 1, "following": 0, "created_at": "2019-03-27T18:48:56Z", "updated_at": "2021-06-17T08:53:24Z" }
```

Figure 2.9: User's information - from redirect url

3 QUESTIONS

3.1

Advantages of *Authorization Code Grant* :

- User is not involved in the request for the access token (prevention of unnecessary user interaction)
- Involves a code exchanging step which prevents an attacker from intercepting the access token (more secure than the Implicit Grant)
- gives the client website almost indefinite access to the user's profile data by featuring refresh tokens

3.2

There are a lot of reasons to avoid using *Authorization Code Grant* in mobile applications :

- User credentials are exposed to the client application
- Public client applications cannot keep a secret and therefore cannot authenticate themselves
- Client applications can request any scope without the user's knowledge
- Users can easily be phished
- Significantly increases the attack surface on user credentials (if the client is compromised, so is the user's entire account)

3.3

Unlike JWT (*JSON Web Token*), this type of token does not have a specific structure and can be viewed as just a random string.

Expires time is needed when sending this token to the client, so it should be added to the parameters as well.

Sending the expires time to the client is done so the client knows it should send a refresh token request or re-auth. Also, if the expires time is encoded in the token, you can't trust what the client sends back to you. You need to validate it against your copy, which means it should be saved somewhere.

3.4

Various vulnerabilities come to mind :

1. Some servers give special treatment to *localhost* URIs.

In some cases, any redirect URI starting with *localhost* may be accidentally permitted in the production environment. This could allow anyone to bypass the validation by registering a domain name such as *localhost.cybercriminal.com*.

2. If an attacker manages to modify a response and substitute the token coming from the AS with one stolen from elsewhere, for example a token issued for a completely different user, there's no way to detect this while using Authorization Code Grant .

3. Another vulnerability comes from *redirect URL*.

An attacker can simply provide a *poisoned* authorization request to the client, which contains an attacker-controlled domain as the redirect URL. Once the victim authenticates, they will be redirected to the attacker-controlled domain.

This redirect will include the *code*, which the attacker can then use to access the victim's account.

4. *Authorization Code grant type* was not initially designed for use in mobile or native applications, which have no way to securely store the *client-secret*. If the secret is leaked, an attacker who compromises a valid code can escalate access by retrieving an access token for that user. The cure for this problem is using *PKCE* which effectively removes the need for a *client-secret*.

Also, *redirect URL* allows cleartext protocols. If the *redirect URL* does not properly require https, the OAuth flow is opened to exploitation in the case of a MITM attack.