OAuth & OpenID Connect

How does the app connect with another service previously?

Previously,

Imagine that an app, such as Yelp, needs to use a service like Gmail. To do this, the app needed to log in using our email and password. By doing so, the app can then access our contacts and other information from Gmail.\

Although the application claims that it will not misuse our Gmail account credentials, it's not a good idea to give access to them. We should not trust any unfamiliar application with our Gmail credentials, as it may lead to the exposure of sensitive information that resides within our Gmail account.

Authorization with O Auth 2.0

In apps like Yelp, there is a "Connect with Google" button that, when clicked, will take the user to the Google website "accounts.google.com".

Here, the user will be prompted to log in, which is safer as we are giving our login information to Google instead of a third-party app.

Then, you may be asked to allow access to Yelp to your contacts and public profile. You can choose to allow or deny access.

If allowed, the browser will redirect to Yelp (as provided in the callback or redirect URI), and Yelp can then connect to the Google Contacts API to access contacts with a little bit of magic 😉

Yelp.com/callback

Yes

No

www.accounts.google.com

Allow Yelp to access your public profile & contacts?

Password

Login

www.accounts.google.com

Yelp.com

Connect with Google

OAuth 2.0 Terminology:

Resource Owner

User, who owns the data.

Client

An application that needs to access data. Yelp in this case.

Authorization Server

The service that authorizes the client to access data. In this case, accounts.google.com presents a login screen.

Resource Server

The server which holds the data, that the client needs. Here, Google contacts API.

Authorization Grant

That means user has consented to client to access the information.

Redirect URI

The URL where the authorization server redirects to.

Access Token

The client eventually needs the access token to access the resource server.

Let’s fit these terminologies in our workflow:

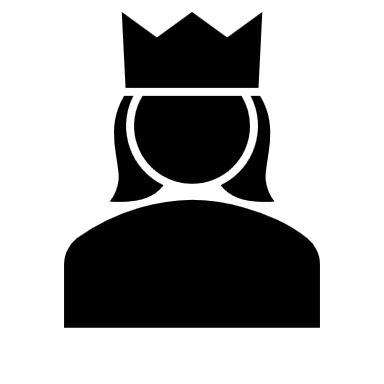
www.accounts.google.com

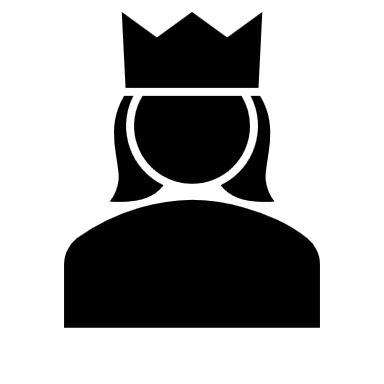
Go to the authorization server

Login

Password

Connect with Google

 Redirect URI: Yelp.com/callback

Resource owner

Response type: code

Client

Authorization Server

Exchange authorization code for access token

Talk to resource server with token

Back to redirect URI

Allow Yelp to access your public profile & contacts?

With authorization code

Yelp.com/callback

www.accounts.google.com

No

Yes

* Resource Owner sitting on the client when clicks on Connect with Google gets redirected to the authorization server.
* At the beginning of the flow client needs to pass configuration like redirect uri & authorization grant as code here (discussed later) so that the authorization server gives code and not token.
* Then the authorization server asks the consent and if allowed then it redirects to redirect uri with the authorization code.
* Then the client exchanges the code with the access token
* Client can now go to the Google Contacts api with the access token

How the Client specifies what accesses it needs

Scope: here scope comes into the picture

In OAuth, the Authorization server provides a list of scopes that it understands,

Like Contacts: read, Contacts: delete. So that clients can provide whatever scopes/permissions they need. The consent screen is based on what the client needs the scopes.

So that the permissions are not black & white i.e. full permission or no permission. It should be in a granular level which can be turned on or off.

So, now the diagram does not change much except in the beginning:

The client needs to specify all scopes needed for the user separated by spaces.

The access token is now only scoped to what the client asked in the beginning

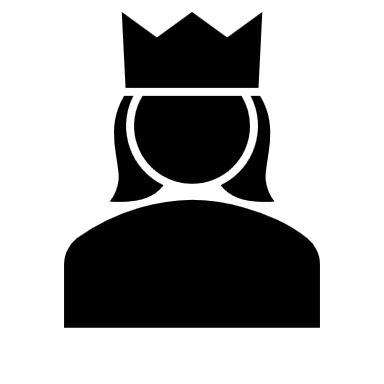
www.accounts.google.com

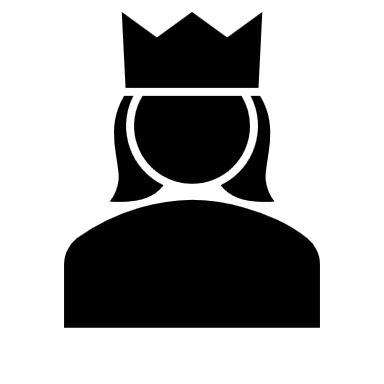
Go to the authorization server

Login

Password

Connect with Google

 Redirect URI: Yelp.com/callback

Resource owner

Response type: code

Scope: profile contacts

Why TWO Steps??

-Why do we get the code first and then the token? Why don't we get a token from the authorization server?

Let's first go through the:

Back Channel (highly secure) – My Backend Server-to-API server communication.

Front Channel (less secure)- My browser to API communication.

Browsers are less secure than backends as anybody interested can find out by checking the browser address bar or network calls.

We can completely trust the calls happening on our server.

So in the above diagram of complete flow,

Some of the steps till redirecting to the client happen in the front channel (solid lines), taking advantage of the best things of the browser.

The redirect URI and the scopes go from the browser as query parameters of the request.

That means the authorization code is also transmitted through the browser. (in the query parameter of the address bar)

The next step happens through the back channel.

What if someone grabs your code before you and exchanges it for an access token?

However, it is not possible as the exchange does not happen through the browser.

The request to the resource server goes with code and secret key also.

So even if someone has stolen your authorization code, they can't do an exchange for an access token because they don’t have a secret key.

So, the exchange request happens in the back channel as we don’t want to keep the secret key in the browser so nobody can steal it.

Talking to resource server with an access token also happens in the back channel as access token is sensitive information.

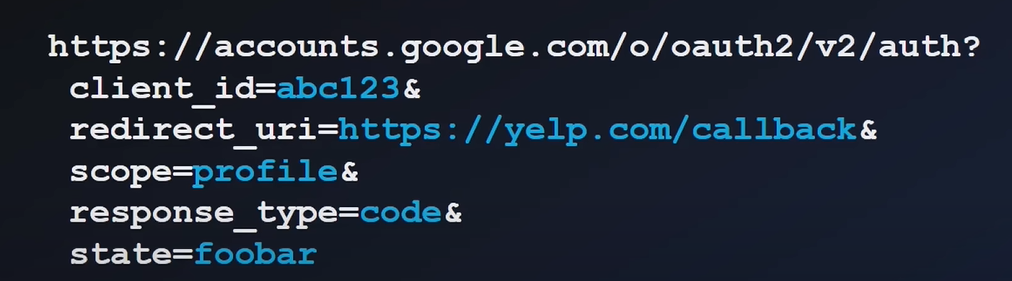
So, the flow is designed to take advantage of best things of both channels. Front channel requests are used to interact with user, present them the login screen, the consent screen. But the exchange of tokens and secret key happens in back channel as it is safe.

Does that mean the client (YELP) needs to setup something with authorization server (Google) to get the secret key??

Yes,

When we click on connect with google, this is the link browser takes us to:

Address of the authorization server



So, for apps here (YELP for ex) to be authorized by the Google authorization server they have to register or set up the client. By creating a client on authorization server (google in this case) we get client ID and secret key.

This secret key is very sensitive info and used by back channel to do token exchange. Client ID is passed in the front channel request to authorization server for the client identification because it is not sensitive.

So exchange with token looks like:

A computer screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

Client can talk to resource owner now:

A computer screen with white text

Description automatically generated

OAuth Flow Types:

Authorization Code Flow (Front Channel + Back Channel) – discussed above

Implicit Flow (Front Channel Flow)

* Implicit Flow is what in which we don’t have back channel in the application (pure React APP maybe).
* Here, instead of sending response type as code to authorization server, we send response type as token.
* We get token immediately not the code that we need to exchange.
* Its less secure as token is directly exposed to the browser

A screenshot of a computer

Description automatically generated

So, OAuth is a protocol that is used for exchanging permissions across different systems.

OAuth is for Authorization. Care about scopes and permissions. It does not have any info about the user’s info.

So, not suitable for authentication.

OpenID Connect is layer on top of OAuth 2.0 protocol. It is not separate from OAuth.

It adds ID token.

So whenever we are talking to authorization server we just don’t ask for access token but we can ask for id token as well.

ID token has information about the user.

In OIDC request, we need to pass a scope openid which means that this is openid request.

Then the normal stuffs happen asking for consent and getting the authorization code then when we exchange the code from authorization server we get access token and ID token as well.

ID token is responsible for getting info of user who just logged in.

The access token can be used to call apis.

ID token is also called JWT. Json web token.

