OAuth 2.0 Beginner's Guide

This article provides an overview of OAuth 2.0 protocol. It discusses the different actors and steps involved in the process of OAuth 2.0 implementation.

Introduction:

OAuth stands for Open Authorization. It's a free and open protocol, built on IETF standards and licenses from the Open Web Foundation. It allows users to share their private resources with a third party while keeping their own credentials secret. These resources could be photos, videos, contact lists, location and billing capabilities, and so on, and are usually stored with another service provider. OAuth does this by granting the requesting (client) applications a token, once access is approved by the user. Each token grants limited access to a specific resource for a specific period.

1. Oauth2 Is a Delegation Protocol:

OAuth2 supports "delegated authentication," i.e. granting access to another person or application to perform actions on your behalf. Consider this scenario: you drive your car to a classy hotel, they may offer valet parking. You then authorize the valet attendant to drive your car by handing him the key in order to let him perform actions on your behalf.

OAuth2 works similarly—a user grants access to an application to perform limited actions on the user's behalf and access can be revoked when it become suspicious.

2. Actors Involved in OAuth2:

i) Resource Server: The server hosting user-owned resources that are protected by OAuth2. The resource server validates the access-token and serves the protected resources.

ii) Resource Owner: Typically, the user of the application is the resource owner. The resource owner has the ability to grant or deny access to their own data hosted on the resource server.

iii) Authorization Server: The authorization server gets consent from the resource owner and issues access tokens to clients for accessing protected resources hosted by a resource server.

iv) Client: An application making API requests to perform actions on protected resources on behalf of the resource owner. Before it may do so, it must be authorized by the resource owner, and the authorization must be validated by resource server/authorization server. OAuth2 defines two client types, based on their ability to authenticate securely with the authorization server (i.e., ability to maintain the confidentiality of their client credentials):

a)Confidential: Clients capable of maintaining the confidentiality of their credentials. Confidential ...mented on a secure server with restricted access to the client credentials (e.g., a web ...ing on a web server).

b) Public: Clients incapable of maintaining the confidentiality of their credentials (e.g. an installed native application or a web browser-based application), and incapable of secure client authentication via any other means.

3. You Are an Application Developer, Here Is a Use Case:

Consider a scenario. You are developing a fun Facebook app, and you call it "FunApp." FunApp needs access to the user's public profile, photos, posts, friends, etc. Now the problem is, how can FunApp get the user's permission to access his/her data from Facebook, while simultaneously informing Facebook that the user has granted this permission to FunApp so that Facebook will share the user's data with this app?

The old way: the user shares his/her facebook credentials (username, password) with FunApp. This approach has some challenges: trust, unrestricted access, changes to the Facebook password made by the user, etc.

The OAuth2 way: Funapp would redirect users to an authorization page on Facebook if the app needed access to their user data. Users would log in to their accounts and grant access, and then FunApp would get an access token from Facebook to access the user's data. While Oauth2 has solved these challenges, it also created costs for developers.

Let's see this scenario from developer's eyes and find out the actors involved here:

- Since Facebook has all the resources (user's public profile, photos, posts, friends, etc.), it becomes the **resource server**.

- The user is the **resource owner**.

- When FunApp requests the user's protected resources, it becomes the **client**.

- When Facebook gets the user's consent and issues the access token to FunApp, it becomes the **authorization server.**

4. Register Client (FunApp) and Get Client Credentials:

OAuth requires that the client registers with the authorization server. The authorization server asks for some basic information about the client such as name, redirect_uri (the URL where the authorization server will redirect to when the resource owner grants permission) and returns client credentials (client-id, client-redirect to when the resource owner grants permission) and returns client credentials (client-id, client secret) to the client. These credentials are critical in protecting the authenticity of requests when performing ope[illegible]changing authorization codes for access tokens and refreshing access tokens.

For [illegible]k requires you to register your client on the Facebook Developers portal. Go to Facebook developers portal and register FunApp and get client credentials.

5. Get an Access Token Step-by-Step:

FunApp needs to get an access token from Facebook to access the user's data. In order to get an access token, FunApp redirects the user to Facebook's login page. Upon successful login, Facebook redirects to the redirect_uri (as registered in step 4) along with the short-lived authorization code. FunApp exchanges the authorization code to get the long-lived access token. The access token is used to access the user's data. This is the most popular flow in OAuth2, called authorization code grant. Here is the sequence diagram to get an access token in the authorization code grant:

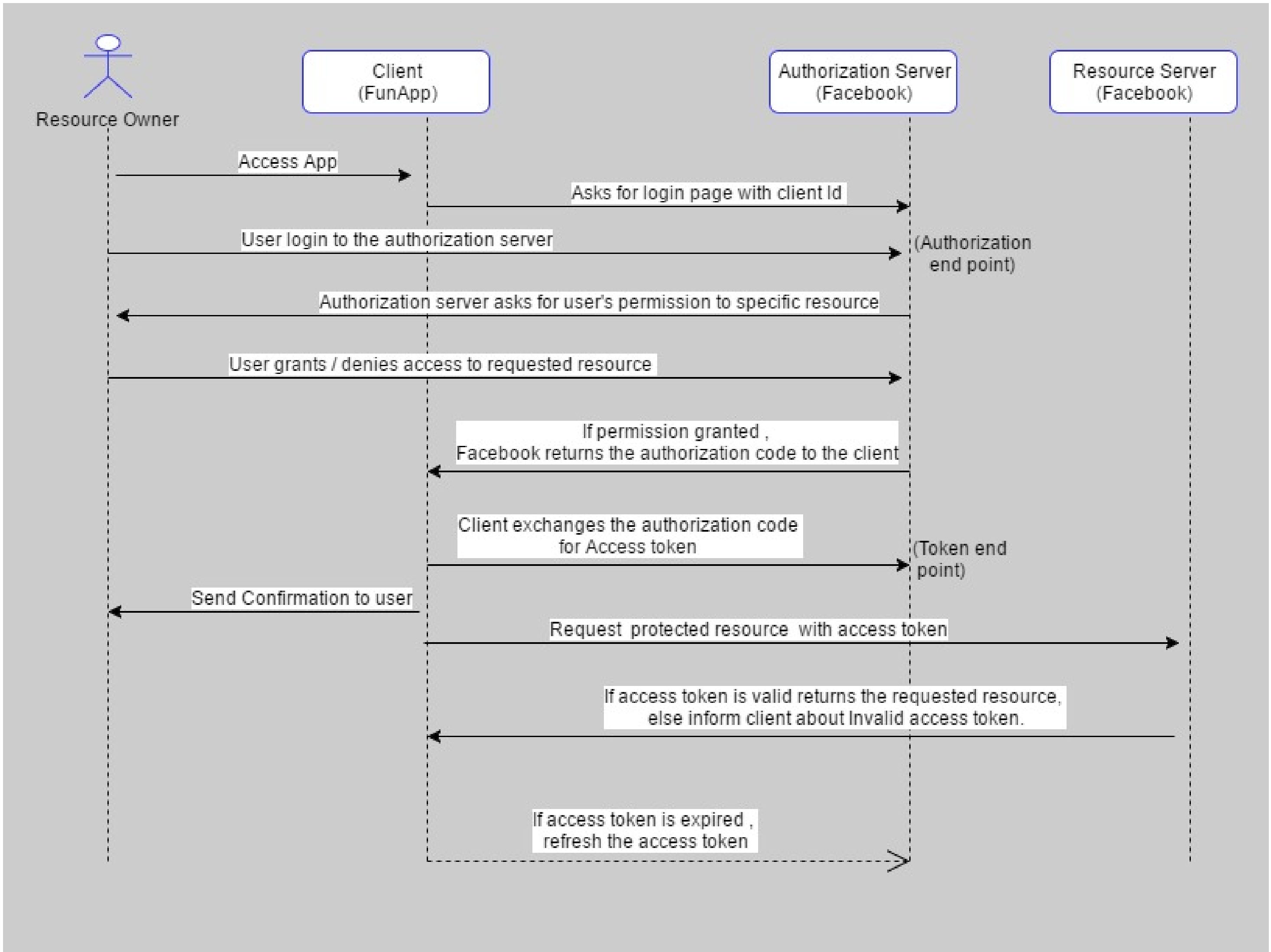
Resource Owner
Client (FunApp)
Authorization Server (Facebook)
Resource Server (Facebook)
Access App
Asks for login page with client Id
User login to the authorization server
(Authorization end point)
Authorization server asks for user's permission to specific resource
User grants / denies access to requested resource
If permission granted , Facebook returns the authorization code to the client
Client exchanges the authorization code for Access token
(Token end point)
Send Confirmation to user
Request protected resource with access token
If access token is valid returns the requested resource, else inform client about Invalid access token.
If access token is expired , refresh the access token

Fig: Authorization code sequence diagram

6. Understanding Authorization Grant Types:

To get the access token, the client obtains authorization from the resource owner. The authorization is

To get the access token, the client obtains authorization from the resource owner. The authorization is expr[illegible] of an authorization grant, which the client uses to request the access token. OAuth2 defi[illegible] grant types: authorization code, implicit, resource owner password credentials, and clie[illegible] so provides an extension mechanism for defining additional grant types.

i) Authorization Code Grant: this grant type is optimized for confidential clients (web application server). The authorization code flow does not expose the access token to the resource owner’s browser. Instead, authorization is accomplished using an intermediary “authorization code” that is passed through the browser. This code must be exchanged for an access token before calls can be made to protected APIs.

ii) Implicit Grant: this grant type is suitable for public clients. The implicit grant flow does not accommodate refresh tokens. If the authorization server expires access tokens regularly, your application will need to run through the authorization flow whenever it needs access. In this flow, an access token is immediately returned to the client after a user grants the requested authorization. An intermediate authorization code is not required as it is in the authorization code grant.

iii) Resource Owner Password Credentials: the resource owner password credentials grant type is suitable in cases where the resource owner has a trust relationship with the client, and the resource owner agrees to share hise/her credentials (username, password) with the client. Then the client can use resources from the owner's credentials to get the access token from the authorization server.

iv) Client Credentials: this grant type is suitable when the client itself owns the data and does not need delegated access from a resource owner, or delegated access has already been granted to the application outside of a typical OAuth flow. In this flow, user consent is not involved. The client exchanges his client credentials to get an access token.

7. Token Expired, Get a New Access Token:

Making API calls using the OAuth 2.0 access token may encounter errors if the access token is no longer valid because the token expired or was revoked. In this case, the resource server will return a 4xx error code. The client can get the new access token using the refresh token (which was obtained when the authorization code was exchanged for an access token).

8. Conclusion:

This is an attempt to provide an overview of the OAuth 2.0 process, as well as provide a way to obtain an access token. I hope it has been helpful.

Have fun integrating apps!