Sathya Bandara Follow
May 1, 2017 · 4 min read ★ · Listen

An Introduction to OAuth 2.0

Prior to introducing what OAuth2 is first let me give you a brief idea on the concepts of authentication and authorization.

Authentication is the process of verifying the identity of a user by obtaining some sort of credentials for example his username password combination, and using those credentials to verify the user's identity.

Authorization is the process of allowing an authenticated user to access his resources by checking whether the user has access rights to the system. You can control access rights by granting or denying specific permissions to an authenticated user. So If the authentication was successful, the authorization process starts. Authentication process always proceeds to Authorization process.

So **OAuth** as it name suggests is simply a standard for authorization. With OAuth you can log into third party websites with your Google, Facebook, Twitter or Microsoft accounts without having the necessity to provide your passwords. This way you can avoid creating accounts and remembering passwords on each and every web application that you use on the Internet.

OAuth is based on an access token concept. When you authenticate yourself using lets say your Google account, to a third party web application. Google authorization server issues an access token to that web application with the approval of the owner. Thus the protected resource data. It's also a safer and more secure way for people to give you access to their resource data.

What is OAuth 2.0?

This is the version 2 of the OAuth protocol. It can be referred to as a authorization framework as well. Version 2 simplifies the previous version of the protocol and facilitates interoperability between different applications. Even Google and popular social websites like Facebook and Twitter also uses OAuth2 protocol for authentications and authorizations.

Roles

OAuth2 defines 4 roles :

- **Resource Owner**: the owner of the resources that needs to be authorized in order to gain access.
- **Resource Server**: server hosting protected data/resources. (for example Google hosting your personal information such as contacts).
- **Client**: application requesting access to a resource server. This can be a third party application that needs to access your protected data.
- **Authorization Server**: server issuing access token to the client. This token will be used for the client(application) to request the resource server. This server can be the same as the authorization server for example Google authorization server.

Tokens

When the client application is authorized by the resource owner, the authorization server issues an access token. The client application can use that token to access resource server APIs. For an example a third party application can request an access token from Google server to use Google contacts API. These tokens are random string generated by the authorization server.
There are 2 types of token:

- **Access Token**: This allows a third-party application to access user data on a application. The client application should provide the scopes it requires with the request to the authorization server. If the scope is minimum there is a greater chance of being authorized by the user.

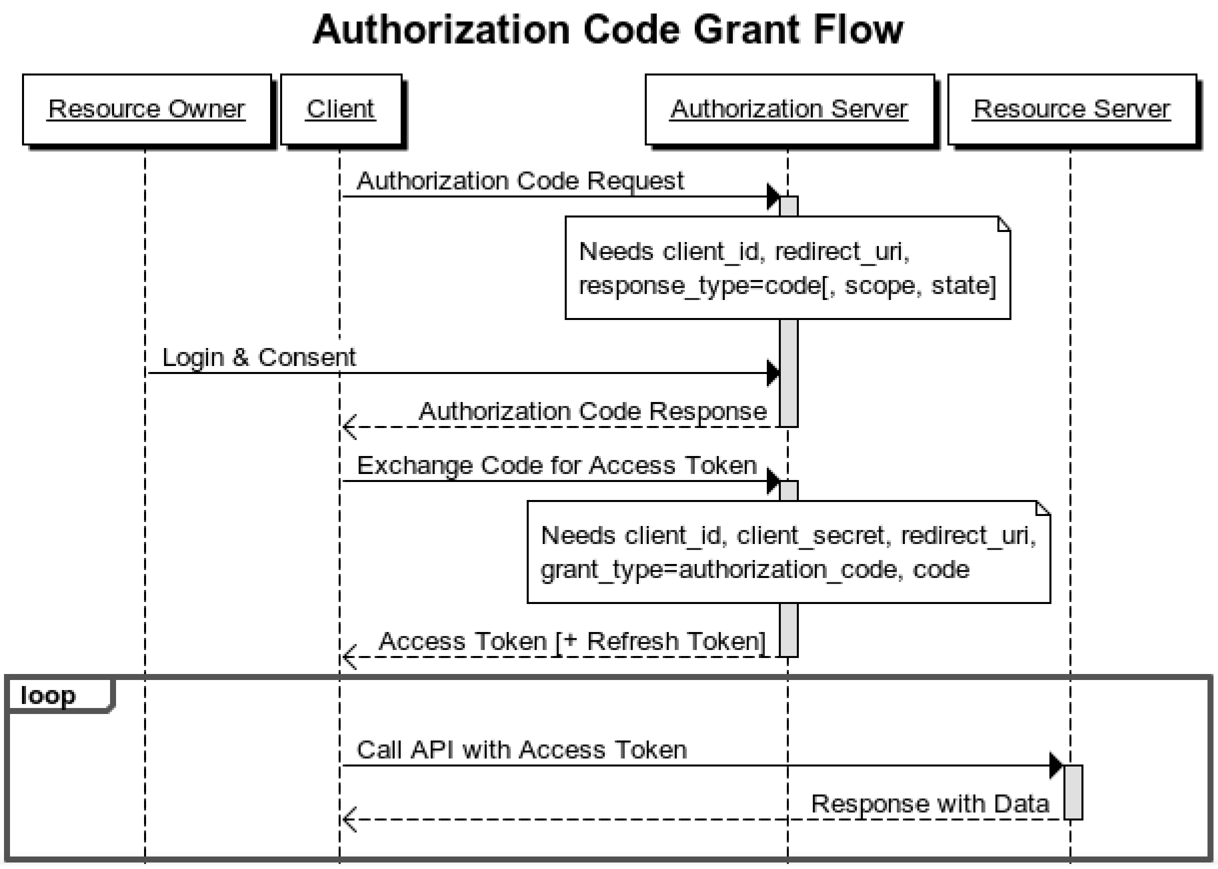
Example of sending an access token to the resource server using HTTP GET:

https://example.com/profile**?access_token=MzJmNDc3M2VjMmQzN**

- **Refresh Token**: this token is issued with the access token but unlike the latter, it is not sent in each request from the client to the resource server. The client application can simply use a ref 241 ew access token when it expires.

OAuth2 uses HTTPS for communication between the client and the authorization server because of confidential data for example client credentials. passing between the two applications.

The below diagram depicts the basic sequence flow for an Oauth2 authorization process.


Authorization Code Grant Flow
Resource Owner
Client
Authorization Server
Resource Server
Authorization Code Request
Needs client_id, redirect_uri, response_type=code[, scope, state]
Login & Consent
Authorization Code Response
Exchange Code for Access Token
Needs client_id, client_secret, redirect_uri, grant_type=authorization_code, code
Access Token [+ Refresh Token]
loop
Call API with Access Token
Response with Data

First the client application requests authorization to access service resources from the user. If the user has not already authenticated with the authorization server, the user will be required to login and consent at the server. After successful authentication, user has to authorize the client at the authorization server. The server responds to the request with an authorization code. The client can then exchange the authorization code with an access token by using client secret. In return the authorization server will issue an access token and a refresh token in addition to that.
With the acquired access token, the client can send requests to the resource server and access resource server API and user data presenting the access token as the means of authentication.