**OAuth 2.0 Protocol**

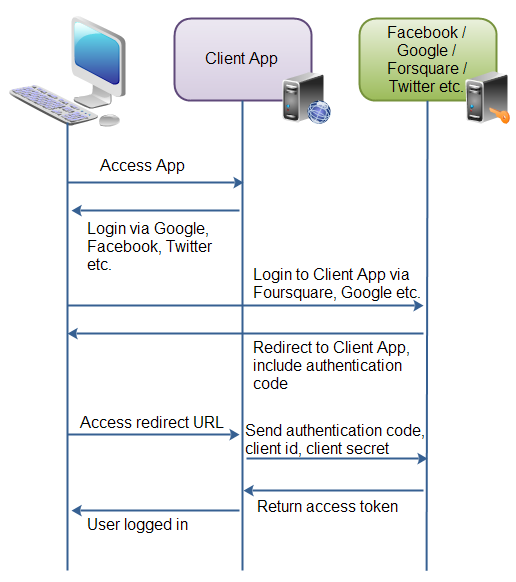
Oauth2.0 is open standard protocol which enables applications to access each other data.

Oauth2.0 is different way a client application can obtain authorization to access the resources stored on resource server. It allows users to share their private resources with a third party (client application) while keeping their own credentials secret. The resource could be photos, videos, contact list, locations etc.

**Oauth2 is a delegation protocol** : oauth2 supports “delegation authentication” i.e. granting access to another person or application to perform action 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.

This diagram shows the authentication process:



The diagram description are given below.  
First, The user accesses the client web application. In this web app, there is a button saying "Login via Facebook" (or some other system like Google or Twitter).

Second, when user clicks on login button, the user is redirected with authenticating application (i.e Facebook ). The user then logs in authenticating application. and is asked if she wants to grant access to her data in the authenticating application, to the client application. The user accepts.

Third, the authenticating application redirects the user to a redirect URI, which the client app has provided to the authenticating app. Providing this redirect URI is normally done by registering the client application with the authenticating application. During this registration the owner of the client application registers the redirect URI. It is also during this registration that the authenticating application gives the client application a client id and a client password. To the URI is appended an authentication code. This code represents the authentication.

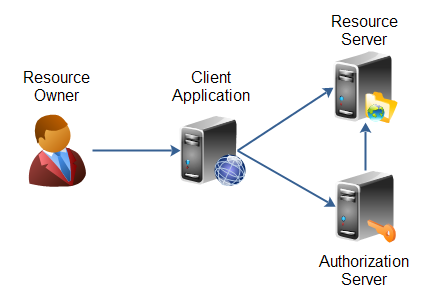
Fourth, the user accesses the page located at the redirect URI in the client application. In the background the client application contacts the authenticating application and sends client id, client password and the authentication code received in the redirect request parameters. The authenticating application sends back an access token.

Once the client application has obtained an access token, this access token can be sent to the Facebook, Google, Twitter etc. to access resources in these systems, related to the user who logged in.

**OAuth 2.0 defines the roles of users and applications**:

1. Resource owner
2. Resource Server
3. Client Application
4. Authorization Server

These roles are depicted in following diagram



1 – **Resource owner :**

The resource owner is the person or application that owns the data that is to be shared. For instance, a user on Facebook or Google could be a resource owner. The resource they own is their data. The resource owner is depicted in the diagram as a person, which is probably the most common situation. The resource owner could also be an application. The OAuth 2.0 specification mentions both possibilities.

2 – **Resource Server** :

The resource server is the server hosting the resources. For instance, Facebook or Google is a resource server (or has a resource server)

3 **– Client Application** :

The client application is the application requesting access to the resources stored on the resource server. The resources, which are owned by the resource owner. The client application could be web application or game application etc.

4 – **Authorization Server** :

The authorization server is the server authorizing the client app to access the resources of the resource owner. The authorization server and the resource server can be the same server, but it doesn't have to. The OAuth 2.0 specification does not say anything about how these two servers should communicate, if they are separate. This is an internal design decision to be made by the resource server + authorization server developers.

**OAuth 2.0 Client Types** :

The OAuth 2.0 client role is subdivided into a set of client types and profiles. The OAuth 2.0 specification defines two types of clients:

**Client Types** :

1 – **Confidential** :

A confidential client is an application that is capable of keeping a client password confidential to the world. This client password is assigned to the client app by the authorization server. This password is used to identify the client to the authorization server, to avoid fraud. An example

of a confidential client could be a web app, where no one but the administrator can get access to the server, and see the client password.

2 – **public** :

A public client is an application that is not capable of keeping a client password confidential. For instance, a mobile phone application or a desktop application that has the client password embedded inside it. Such an application could get cracked, and this could reveal the password. The same is true for a JavaScript application running in the users browser. The user could use a JavaScript debugger to look into the application, and see the client password.

**Client Profiles** :

The OAuth 2.0 specification also mentions a set of client profiles. These profiles are concrete types of applications, that can be either confidential or public. The profiles are:

**Web Application** :

A web application is an application running on a web server. In reality, a web application typically consists of both a browser part and a server part. If a web application needs access to a resource server (e.g. to Facebook user accounts), then the client password could be stored on the server. The password would thus be confidential.

**User Agent** :

A user agent application is for instance a JavaScript application running in a browser. The browser is the user agent. A user agent application may be stored on a web server, but the application is only running in the user agent once downloaded. An example could be a little JavaScript game that only runs in the browser.

**Native** :

A native application is for instance a desktop application or a mobile phone application. Native applications are typically installed on the users computer or device (phone, tablet etc.). Thus, the client password will be stored on the users computer or device too.

**OAuth 2.0 Authorization** :

When a client applications wants access to the resources of a resource owner, hosted on a resource server, the client application must first obtain an authorization grant.

**Client ID, Client Secret and Redirect URI** :

Before a client application can request access to resources on a resource server, the client application must first register with the authorization server associated with the resource server.

The registration is typically a one-time task. Once registered, the registration remains valid, unless the client app registration is revoked.

At registration the client application is assigned a client ID and a client secret (password) by the authorization server. The client ID and secret is unique to the client application on that authorization server. If a client application registers with multiple authorization servers (e.g. both Facebook, Twitter and Google), each authorization server will issue its own unique client ID to the client application.

Whenever the client application requests access to resources stored on that same resource server, the client application needs to authenticate itself by sending along the client ID and the client secret to the autorhization server.

During the registration the client also registers a redirect URI. This redirect URI is used when a resource owner grants authorization to the client application. When a resource owner has successfully authorized the client application via the authorization server, the resource owner is redirected back to the client application, to the redirect URI.

**Authorization Grant**

The authorization grant is given to a client application by the resource owner, in cooperation with the authorization server associated with the resource server.

The OAuth 2.0 specification lists four different types of authorization grants. Each type has different security characteristics. The authorization grant types are:

1-**Authorization Code**

2-**Implicit**

3-**Resource Owner Password Credentials**

4-**Client Credentials**

**Grant Type :**

* **Authorization Code** :

1) The resource owner (user) accesses the client application.

2) The client application tells the user to login to the client application via an authorization server (e.g. Facebook, Twitter, Google etc.).

3) To login via the authorizaion server, the user is redirected to the authorization server by the client application. The client application sends its client ID along to the authorization server, so the authorization server knows which application is trying to access the protected resources.

4) The user logs in via the authorization server. After successful login the user is asked if she wants to grant access to her resources to the client application. If the user accepts, the user is redirected back to the client application.

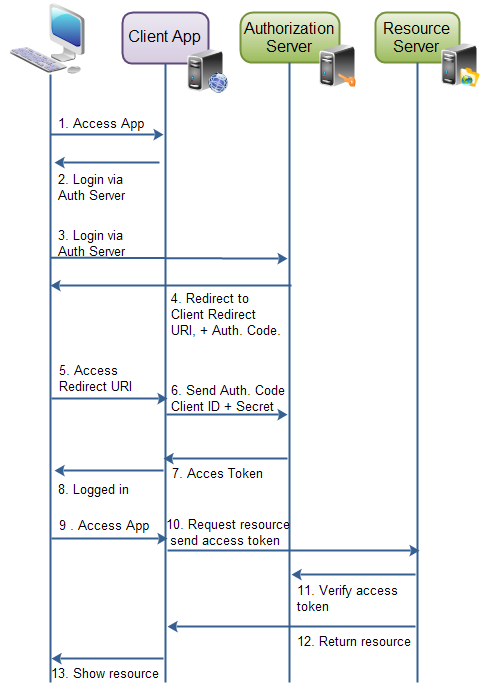
5) When redirected back to the client application, the authorization server sends the user to a specific redirect URI, which the client application has registered with the authorization server ahead of time. Along with the redirection, the authorization server sends an authorization code, representing the authorization.

6) When the redirect URI in the client application is accessed, the client application connects directly to the authorization server. The client application sends the authorization code along with its own client ID and and client secret.

7) If the authorization server can accept these values, the authorization server sends back an access token.

10) The client application can now use the access token to request resources from the resource server. The access token serves as both authentication of the client, resource owner (user) and authorization to access the resources.

We can depict the authorization code with the following diagram.



Authorization grant via authorization code

* **Implicit**

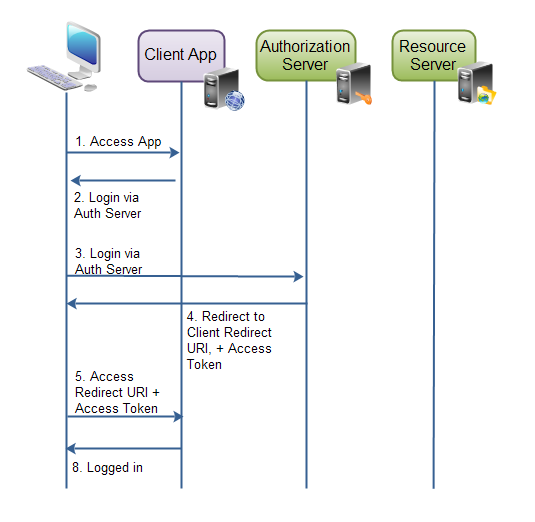
An implicit authorization grant is similar to an authorization code grant, except the access token is returned to the client application already after the user has finished the authorization. The access token is thus returned when the user agent is redirected to the redirect URI.

This of course means that the access token is accessible in the user agent, or native application participating in the implicit authorization grant. The access token is not stored securely on a web server.

Furthermore, the client application can only send its client ID to the authorization server. If the client were to send its client secret too, the client secret would have to be stored in the user agent or native application too. That would make it vulnerable to hacking.

Implicit authorization grant is mostly used in a user agent or native client application. The user agent or native application would receive the access token from the authorization server.

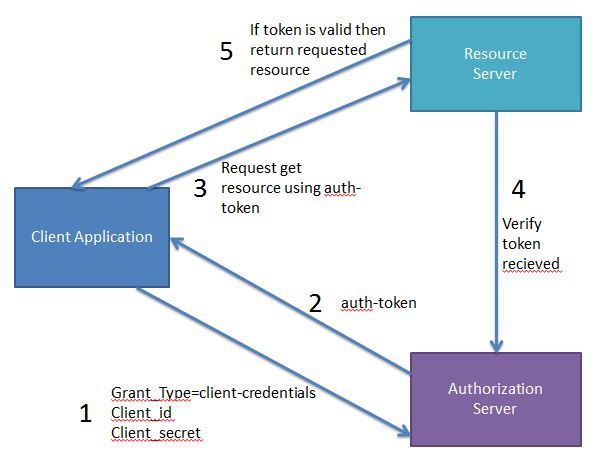
illustration of implicit authorization grant:



* **Client Credential :**

In case of Client credentials grant type the user has no role to play. As previously stated it is machine to machine communication. This is typically used by clients to access resources about themselves rather than to access a user's resources.

This type of Authentication does not involve any end-user. Unlike Authorization Grant where the end user had to authenticate himself using Authorization Server like Gmail, here the machine it self authenticates itself to access a protected resource



* **Resource Owner Password Credentials :**

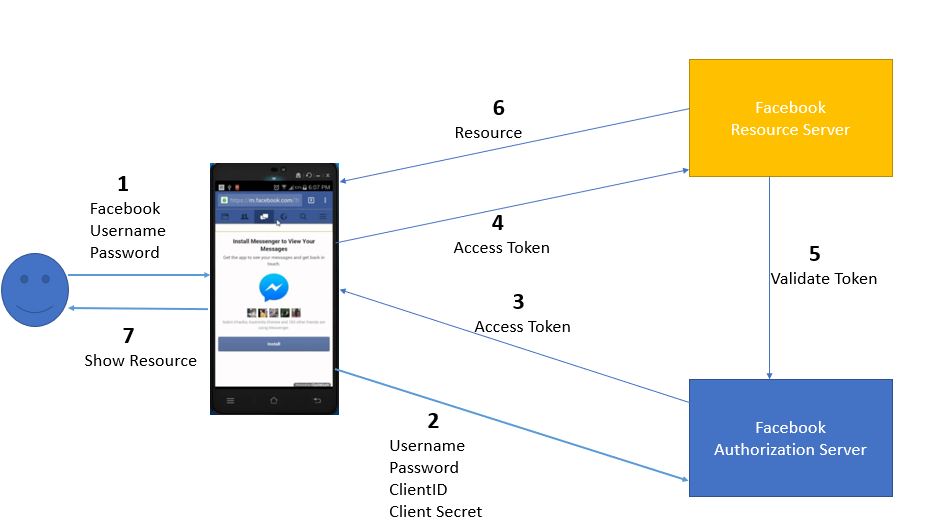
The resource owner password credentials authorization grant method works by giving the client application access to the resource owners credentials. For instance, a user could type his Twitter user name and password (credentials) into the client application. The client application could then use the user name and password to access resources in Twitter.

Using the resource owner password credentials requires a lot of trust in the client application. You do not want to type your credentials into an application you suspect might abuse it.

The resource owner password credentials would normally be used by user agent client applications, or native client applications.

In case of Password grant type the user triggers the client to get some resource. While doing so it passes the username and password to the client. The client then communicates with the authorization server using the provided username, password and also its own clientId and clientSecret to get the access token. Using this access token it then gets the required resource from the resource server.

The real life example of Password grant will be you doing a login to you facebook account using its mobile application. Here the user will have to specify the facebook credentials to the app. Also the app will be having its own client id and client secret.



**OAuth 2.0 Endpoints :**

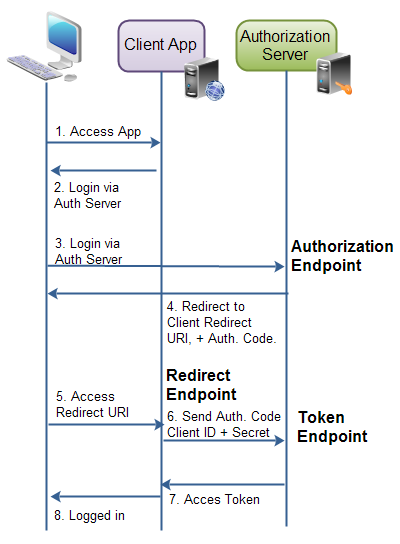
OAuth 2.0 defines a set of endpoints. An endpoint is typically a URI on a web server. For instance, the address of a Java servlet, JSP page, PHP page, ASP.NET page etc.

Authorization Endpoint

Token Endpoint

Redirection Endpoint

The authorization endpoint and token endpoint are both located on the authorization server. The redirection endpoint is located in the client application.



**Authorization Endpoint**

The authorization endpoint is the endpoint on the authorization server where the resource owner logs in, and grants authorization to the client application.

**Token Endpoint**

The token endpoint is the endpoint on the authorization server where the client application exchanges the authorization code, client ID and client secret, for an access token.

**Redirect Endpoint**

The redirect endpoint is the endpoint in the client application where the resource owner is redirected to, after having granted authorization at the authorization endpoint.

**OAuth 2.0 Requests and Responses**

When the client application requests authorization and access tokens it sends HTTP requests to the authorization server, to its authorization and token endpoints. What request and response is sent forth and back depends on the authorization grant type.

**Authorization Code Requests and Responses :**

The authorization code grant consists of 2 requests and 2 responses in total. An authorization request + response, and a token request + response.

**Authorization Request**

The authorization request is sent to the authorization endpoint to obtain an authorization code. Here are the parameters used in the request:

response\_type Required. Must be set to code

client\_id Required. The client identifier as assigned by the authorization   
 server, when the client was registered.

redirect\_uri Optional. The redirect URI registered by the client.

scope Optional. The possible scope of the request.

state Optional (recommended). Any client state that needs to be   
 passed on to the client request URI.

**Authorization Response**

The authorization response contains the authorization code needed to obtain an access token. Here are the parameters included in the response:

code Required. The authorization code.

state Required, if present in request. The same value as sent by the client in   
 the state parameter, if any.

**Token Request**

Once an authorization code is obtained, the client can use that code to obtain an access token. Here is the access token request parameters:

client\_id Required. The client application's id.

client\_secret Required. The client application's client secret .

grant\_type Required. Must be set to authorization\_code .

code Required. The authorization code received by the authorization server.

redirect\_uri Required, if the request URI was included in the authorization request.  
 Must be identical then.

**Token Response**

The response to the access token request is a JSON string containing the access token plus some more information:

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

"refresh\_token" : "...",

}

The access\_token property is the access token as assigned by the authorization server.

The token\_type property is a type of token assigned by the authorization server.

The expires\_in property is a number of seconds after which the access token expires, and is no longer valid. Expiration of access tokens is optional.

The refresh\_token property contains a refresh token in case the access token can expire. The refresh token is used to obtain a new access token once the one returned in this response is no longer valid.

**Implicit Requests and Responses**

The implicit grant consists of only 1 request and 1 response.

**Implicit Grant Request**

The implicit grant request contains the following parameters:

response\_type Required. Must be set to token .

client\_id Required. The client identifier as assigned by the authorization   
 server, when the client was registered.

redirect\_uri Optional. The redirect URI registered by the client.

scope Optional. The possible scope of the request.

state Optional (recommended). Any client state that needs to be passed   
 on to the client request URI.

**Implicit Grant Response**

The implicit grant response contains the following parameters. Note, that the implicit grant response is not JSON.

access\_token Required. The access token assigned by the authorization server.

token\_type Required. The type of the token

expires\_in Recommended. A number of seconds after which the access token  
 expires.

scope Optional. The scope of the access token.

state Required, if present in the autorization request. Must be same value   
 as state parameter in request.

**Resource Owner Password Credentials Grant - Requests and Response**

The resource owner password credentials authorization contains a single request + response.

**Resource Owner Password Credentials Grant Request**

The request contains the following parameters:

grant\_type Required. Must be set to password

username Required. The username of the resource owner, UTF-8 encoded.

password Required. The password of the resource owner, UTF-8 encoded.

scope Optional. The scope of the authorization.

**Resource Owner Password Credentials Grant Response**

The response is a JSON structure containing the access token. The JSON structure looks like this:

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

"refresh\_token" : "...",

}

The access\_token property is the access token as assigned by the authorization server.

The token\_type property is a type of token assigned by the authorization server.

The expires\_in property is a number of seconds after which the access token expires, and is no longer valid. Expiration of access tokens is optional.

The refresh\_token property contains a refresh token in case the access token can expire. The refresh token is used to obtain a new access token once the one returned in this response is no longer valid.

**Client Credentials Grant - Requests and Response :**

**Client Credentials Grant Request**

The client credentials grant request contains the following parameters:

grant\_type Required. Must be set to client\_credentials .

scope Optional. The scope of the authorization.

**Client Credentials Grant Response**

The client credentials response contains the following parameters:

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

}

The access\_token property is the access token as assigned by the authorization server.

The token\_type property is a type of token assigned by the authorization server.

The expires\_in property is a number of seconds after which the access token expires, and is no longer valid. Expiration of access tokens is optional.

A refresh token should not be included for this type of authorization request.