
# OAuth 2.0

## EXPLICATIONS ET UTILISATIONS

Léo LAROU-CHALOT
SAFEBEAR | PARIS

# OAuth 2.0, une histoire de confiance

OAuth 2.0, successeur du protocole Oauth 1.0a, est un framework d'autorisation permettant à une application tierce d'accéder à un service web.

Largement utilisé dans le domaine du web avec notamment Facebook ou encore Google, OAuth est devenu un incontournable.

En tant que développeurs nous sommes régulièrement amenés à utiliser un serveur fournissant un accès via OAuth 2.0 ou à implémenter un serveur d'autorisation pour sécuriser une API en utilisant ce framework.

L'objectif de cette documentation est de comprendre comment fonctionne OAuth 2.0, en évitant d'y rentrer trop en profondeur. Nous éviterons donc d'aborder l'implémentation de celui-ci.

# Table des matières

# 1 LA NAISSANCE DE OAUTH

L'apparition de OAuth est né d'un besoin simple.

Comment une application peut-elle accéder à une ressource protégée au nom de son propriétaire, sans connaitre ses identifiants ?

Avant la naissance de OAuth, l'utilisateur devait faire confiance à un site web, en lui fournissant directement ses identifiants.

Cependant, pour éviter de redemander les identifiants à l'utilisateur, les applications tierces avaient tendance à conserver le mot de passe de celui-ci en clair. En plus des limites de sécurité liées à cette conservation du mot de passe en clair, l'application tierce était en mesure d'accéder à toutes les informations protégées sans distinction.

Pour répondre à ces problématiques, OAuth 2.0 a été créé afin de permettre des interactions entre applications dans un contexte sécuritaire optimal. OAuth 2.0 a été conçu pour être utilisé avec le protocole HTTPS.

OAuth doit toujours s'utiliser avec une connexion sécurisée (HTTPS) pour remplir correctement ses objectifs.

# 2 LES ROLES

## 2.1 CONCEPT TRADITIONNEL

Afin de bien comprendre les mécanismes en jeu lors d'une demande d'autorisation avec OAuth 2.0, nous devons d'abord définir quelques notions utilisées dans les spécifications de ce framework.

Le schéma classique d'un processus d'authentification et d'autorisation fait intervenir deux parties :

- un serveur en mesure d'authentifier et d'autoriser l'accès à une ressource ;
- un utilisateur qui fournit ses identifiants pour accéder à la ressource.

Dans ce contexte, une ressource désigne toute information appartenant à l'utilisateur, et qui est exposée par le serveur (information de profil, photos, etc…)

## 2.2 PROCESSUS D'AUTORISATION AVEC OAUTH 2.0

Pour OAuth 2.0, le processus d'autorisation est un peu plus complexe et peut faire intervenir **jusqu'à quatre acteurs**

- Le propriétaire de la ressource / Ressource Owner : Utilisateur en mesure de donnér l'accès à une ressource protégée.
- Le client : L'application tierce qui demande l'accès à la ressource au nom de son propriétaire. Le terme client peut aussi bien désigner une application mobile qu'un serveur web. Il permet d'identifier l'entité qui souhaite accéder à une ressource.
- Le serveur de ressoure / Ressource Server : Le serveur qui héberge les ressources protégées.
- Le serveur d'autorisation / Authorization Server : Le serveur qui délivre le droit d'accès à la ressource protégée au client après avoir authentifié le propriétaire de celle-ci.

Dans les faits, les serveurs d'autorisation et de ressource sout souvent confondus. Ainsi, un même sevreur pourra jouer le rôle de serveur d'autorisation, et héberger les ressources protégées.

Par exemple, lorsqu'un site web nous demande l'autorisation de publier du contenu sur notre mur Facebook, nous sommes le propriétaire, le site web est le client, et les serveurs de Facebook jouent les double rôle de serveurs d'autorisation et de ressource.

# 3 LA GESTION DES CLIENTS

Une demande d'autorisation avec OAuth 2.0 est toujours initiée par un client. Pour tous les clients, il faudra donc les enregistrer auprès du serveur d'autorisation.

L'enregistrement nécessite au moins trois informations :

- l'identifiant du client ;
- le mot de passe ou la paire de clés (publique/privée) pour les clients confidentiels ;
- et une ou plusieurs URL de redirection.

# 4 LES TOKENS

La demande d'accès à une ressource protégée via OAuth se traduit par la délivrance d'un token au client. Le token représente juste une chaîne de caractère unique permettant d'identifier le client et les différentes informations utiles durant le processus d'autorisation.

Le serveur d'autorisation est en mesure d'en fournir deux types.

## 4.1 TOKEN D'ACCES : ACCESS TOKEN

Le token d'accès permet au client d'accéder à la ressource protégée. Ce token a une **durée de validité** limitée et peut avoir une portée limitée (**scope**).

Cette notion de portée permet d'accorder un accès limité au client. Ainsi, un utilisateur autoriser un client à accéder à ses ressources qu'en lecture seule.

## 4.2 TOKEN DE RAFAICHISSEMENT : REFRESH TOKEN

Le **token de rafraîchissement** permet au client d'obtenir un nouveau **token** d'accès une fois que celui-ci a expiré. Sa durée de validité est aussi limité mais est beaucoup plus élevée que celle du **token** d'accès.

## 4.3 RESUME

En résumé, OAuth 2.0 formalise un ensemble de mécanismes permettant à une application tierce (client) d'accéder à une ressource protégée au nom de son propriétaire (resource owner) ou en son propre nom. Cette autorisation se traduit par la délivrance d'un token d'accès (et éventuellement d'un token de rafraîchissement) qui permet au client de dialoguer avec le serveur hébergeant les ressources protégées (serveur de ressource).

# 5 SCENARIOS D'AUTORISATION

Contrairement à son prédécesseur qui ne proposait qu'un seul scénario d'autorisation, le framework OAuth 2.0 a été spécifié avec quatre méthodes différentes pour obtenir une autorisation. En plus, le framework est extensible et permet donc de rajouter autant de méthodes que nous le souhaitons.

Selon la nature du client et du niveau d'accès souhaité, le scénario à utiliser n'est pas forcément le même. Le serveur d'autorisation peut ainsi autoriser l'utilisation d'un ou de plusieurs scénarios selon les besoins.

## 5.1 AUTORISATION AVEC CODE : AUTHORIZATION GRANT FLOW

Ce processus d'autorisation principalement utilisé par les clients confidentiels permet d'obtenir un token d'accès (Access token) et un token de rafraîchissement (Refresh token). Il nécessite l'intervention du client, du propriétaire de la ressource protégée et du serveur d'autorisation. C'est d'ailleurs le mode d'utilisation à l'origine du protocole OAuth 1.0.

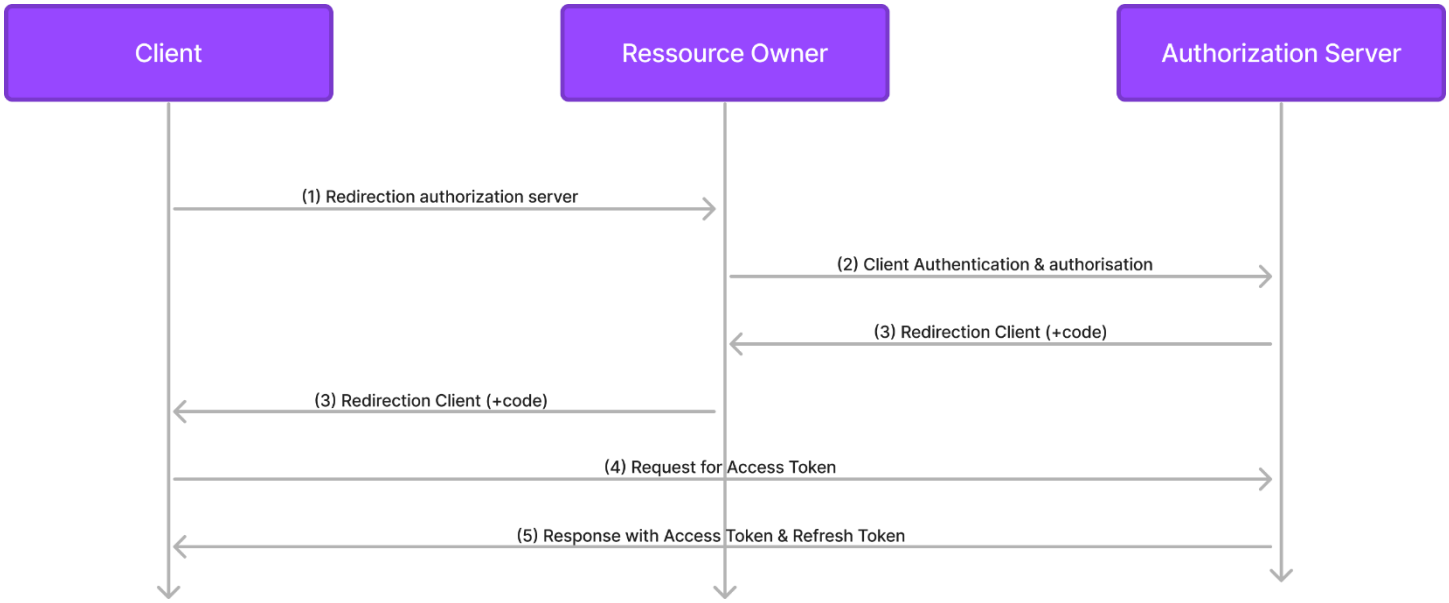
Les clients dits confidentiels sont les clients en mesure de conserver en toute confidentialité les identifiants qui leurs sont affectés.

### 5.1.1 Cas d'usage

Le cas d'usage typique pour ce type d'autorisation est lorsque le client est un code côté serveur.

Exemple : Nous avons développé une application web, et nous souhaitons accéder à certains services Google d'un utilisateur depuis le serveur, le procédé d'autorisation avec un code serait le plus approprié. L'accès au code serveur étant sécurisé, le client peut être considéré comme confidentiel.

### 5.1.2 Description



(1) Le client redirige le propriétaire de la ressource vers le serveur d'autorisation. Le client doit inclure son identifiant dans la requête de redirection et le niveau d'accès qu'il souhaite obtenir.
(2) Le propriétaire de la ressource s'authentifie auprès du serveur d'autorisation et approuve ou non la requête du client.
(3) Si la requête est autorisée, le serveur d'autorisation redirige à nouveau le propriétaire de la ressource en utilisant l'URL de redirection défini par le client. Cette URL est renseignée à l'enregistrement du client et peut aussi être choisie dans la requête à l'étape 1. La requête de redirection contient un code d'autorisation dans l'URL.
(4) Avec le code d'autorisation ainsi obtenu, le client demande un token d'accès en prenant le soin de s'authentifier à son tour auprès du serveur d'autorisation.
(5) Une fois le client authentifié, le serveur d'autorisation valide le code d'autorisation et s'assure que l'URL de redirection est identique à celle utilisée dans la troisième étape. Si toutes ces contraintes sont respectées, le serveur d'autorisation renvoie au client un token d'accès et éventuellement un token de rafraîchissement.

Pour assurer ces différentes interactions, le serveur d'autorisation doit mettre à disposition des clients deux URL. Une URL d'autorisation (Authorization endpoint) qui sera utilisée dans l'étape 1 et permettra d'obtenir un code d'autorisation et une URL de génération de tokens (Tokens endpoint) permettant d'obtenir les différents tokens.

### 5.1.3 Exemple d'utilisation

Nous développons un site web appelé exemple.com permettant aux utilisateurs d'enregistrer leurs mails importants sur Gmail.

Google dispose de deux endpoints pour gérer ce type d'autorisation :

- **https://accounts.google.com/o/oauth2/v2/auth** pour initialiser le processus d'autorisation (par exemple, la génération de code d'autorisation) ;
- **https://www.googleapis.com/oauth2/v4/token** pour la génération des tokens d'accès.

**Le processus commence par une redirection de l'utilisateur** (propriétaire de la ressource) vers l'URL https://accounts.google.com/o/oauth2/v2/auth?scope=email&redirect_uri=https%3A%2F%2Fexample.com%2Foauth2-redirect&response_type=code&client_id=id_1 (1).

Si l'utilisateur s'authentifie (ou est déjà authentifié) et autorise notre application (Étape 2), Google le redirige vers notre site avec le code d'autorisation.

L'**URL de redirection** est **utilisée** ainsi pour **transmettre les informations à notre site** : https://example.com/auth2-redirect?code=code_autorisation&state=utile_pour_eviter_les_failles_csrf (Étape 3).

```
POST /oauth2/v4/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=code_authorization&
client_id=<id client>&
client_secret=<secret client>&
redirect_uri=<redirection client>&
grant_type=authorization_code
```

*5-2Requête du client au serveur d'autorisation*

```
{
    "access_token": "access_token_1",
    "expire_in": 15000,
    "token_type": "Bearer"
}
```

*5-1Réponse du serveur d'autorisation*

## 5.2 AUTORISATION IMPLICITE : IMPLICIT GRANT

Ce processus est idéal pour les applications clientes dites publiques. Comme pour l'autorisation avec un code, il fait intervenir le client, le propriétaire de la ressource protégée et le serveur d'autorisation.

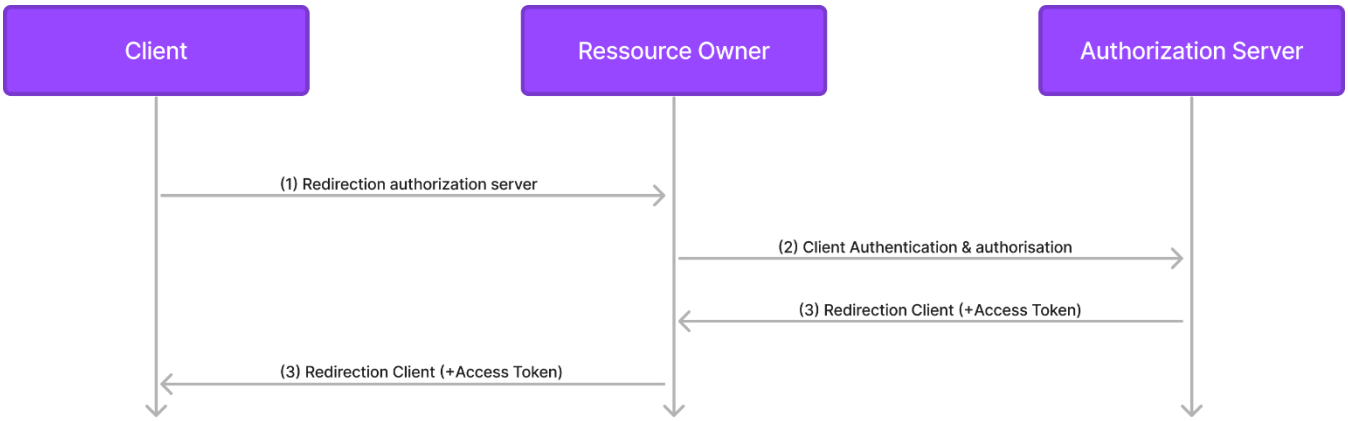
Les clients publics, à l'opposé des clients confidentiels, ne peuvent pas assurer la confidentialité de leurs identifiants.

### 5.2.1 Cas d'usage

Les applications mobiles ou encore les applications JavaScript exécutées sur le navigateur de l'utilisateur sont les principales utilisatrices de ce moyen d'autorisation. Le client étant publique, il ne peut pas s'authentifier de manière fiable.

Ainsi, contrairement à l'autorisation avec un code décrite plus haut, le client ne peut fournir de mot de passe. Son identifiant et l'URL de redirection préenregistrée permettront de l'authentifier d'où le nom d'autorisation implicite. Du moment où le serveur d'autorisation redirige le propriétaire de la ressource vers une URL associée au client, il considère implicitement que le client a fait la demande d'autorisation et va traiter la réponse.

### 5.2.2 Description




(1) Le client commence par rediriger le propriétaire de la ressource vers le serveur d'autorisation. Le client doit inclure son identifiant dans la requête de redirection et le niveau d'accès qu'il souhaite obtenir.
(2) Le propriétaire de la ressource s'authentifie auprès du serveur d'autorisation et approuve ou non la requête du client.
(3) Si la requête est autorisée, le serveur d'autorisation redirige à nouveau le propriétaire de la ressource en utilisant l'URL de redirection défini par le client. La requête de redirection contient dans son URL le token d'accès qui permettra d'accéder aux ressources protégées.

## 5.3 AUORISATION AVEC LES IDENTIFIANTS DU CLIENT : CLIENT CREDENTIALS GRANT

Ce processus est le seul qui ne fait intervenir que deux acteurs :

- le client ;
- le serveur d'autorisation.

Il est souvent appelé two-legged OAuth (deux jambes) ou 2LO.

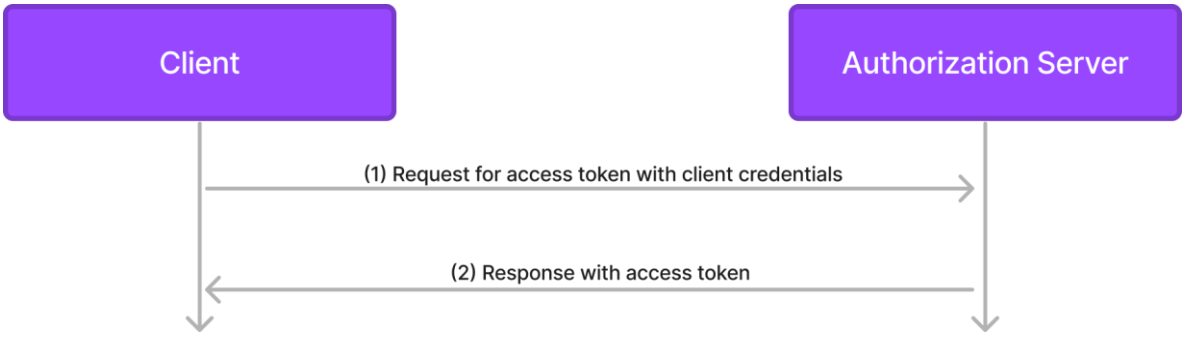
Le propriétaire de la ressource n'est pas sollicité car cette méthode ne permet pas en général d'accéder à des ressources protégées (à part ceux du client).

### 5.3.1 Cas d'usage

L'utilisation de ce mode d'autorisation peut s'avérer très utile si un serveur fournit une API publique mais veut ajouter une limitation à l'usage qui est faite par les clients.

Ainsi, le serveur pourra mettre en place par exemple un système de limitation du nombre de requêtes par client ou encore permettre à un client de configurer ses propres paramètres.

### 5.3.2 Description




(1) Le client envoie ses identifiants au serveur d'autorisation pour obtenir un token d'accès.
(2) Si les identifiants du client sont valides, le serveur d'autorisation génère un token unique.

### 5.3.3 Exemple d'utilisation

Utilisons l'exemple de Facebook avec sa notion de **token d'accès d'app.**

Ce type de token est nécessaire pour modifier et lire les paramètres d'une application Facebook.

La requête pourrait ressembler à :

```
GET /oauth/access_token
    ?client_id=<id client>
    &client_secret=<secret client>
    &grant_type=client_credentials
```

*5-4Reqête HTTP du client au server*

```
{
    "access_token": "<access_token>",
    "token_type": "<type>",
    "expire_in": <second untill expiration>
}
```

*5-3Reponse du serveur d'autorisation*