



BubbleCode

By Johann Reinke – PHP Developer – Reims, France



Understanding OAuth2

Friday January 22nd, 2016

If OAuth2 is still a vague concept for you or you simply want to be sure you understand its behaviours, this article should interest you.

What is OAuth2?



OAuth2 is, you guessed it, the version 2 of the OAuth protocol (also called framework).

This protocol allows third-party applications to grant limited access to an HTTP service, either on behalf of a resource owner or by allowing the third-party application to obtain access on its own behalf. Access is requested by a client, it can be a website or a mobile application for example.

Version 2 is expected to simplify the previous version of the protocol and to facilitate interoperability between different applications.

Specifications are still being drafted and the protocol is constantly evolving but that does not prevent it from being implemented and acclaimed by several internet giants such as Google or Facebook.

Basic knowledge

Roles

OAuth2 defines 4 roles :

- **Resource Owner:** generally yourself.
- **Resource Server:** server hosting protected data (for example Google hosting your profile and personal information).
- **Client:** application requesting access to a resource server (it can be your PHP website, a Javascript application or a mobile application).
- **Authorization Server:** server issuing access token to the client. This token will be used for the client to request the resource server. This server can be the same as the authorization server (same physical server and same application), and it is often the case.

Tokens

Tokens are random strings generated by the authorization server and are issued when the client requests them.

There are 2 types of token:

- **Access Token:** this is the most important because it allows the user data from being accessed by a third-party application. This token is sent by the client as a parameter or as a header in the request to the resource server. It has a limited lifetime, which is defined by the authorization server. **It must be kept confidential as soon as possible** but we will see that this is not always possible, especially when the client is a web browser that sends requests to the resource server via Javascript.
- **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. It merely serves to be sent to the authorization server for renewing the access token when it has expired. For security reasons, it is not always possible to obtain this token. We will see later in what circumstances.

Access token scope

The scope is a parameter used to limit the rights of the access token. This is the authorization server that defines the list of the available scopes. The client must then send the scopes he wants to use for his application during the request to the authorization server. More the scope is reduced, the greater the chance that the resource owner authorizes access.

More information: <http://tools.ietf.org/html/rfc6749#section-3.3>.

- What is OAuth2?
- Basic knowledge
 - Roles
 - Tokens
 - Access token scope
 - HTTPS
- Register as a client
 - Client registration
 - Authorization server response
- Authorization grant types
 - Authorization Code Grant
 - Implicit Grant
 - Resource Owner Password Credentials Grant
 - Client Credentials Grant
- Access token usage
 - Request parameter (GET or POST)
 - Authorization header
- Security
 - Vulnerability in Authorization Code Grant
 - Vulnerability in Implicit Grant
 - Clickjacking
- Conclusion

HTTPS

OAuth2 requires the use of HTTPS for communication between the client and the authorization server because of sensitive data passing between the two (tokens and possibly resource owner credentials). In fact you are not forced to do so if you implement your own authorization server but you must know that you are opening a big security hole by doing this.

Register as a client

Since you want to retrieve data from a resource server using OAuth2, you have to register as a client of the authorization server.

Each provider is free to allow this by the method of his choice. The protocol only defines the parameters that must be specified by the client and those to be returned by the authorization server.

Here are the parameters (they may differ depending of the providers):

Client registration

- Application Name: the application name
- Redirect URLs: URLs of the client for receiving authorization code and access token
- Grant Type(s): authorization types that will be used by the client
- Javascript Origin (optional): the hostname that will be allowed to request the resource server via XMLHttpRequest

Authorization server response

- Client Id: unique random string
- Client Secret: secret key that must be kept confidential

More information: [RFC 6749 — Client Registration](#).

Authorization grant types

OAuth2 defines 4 grant types depending on the location and the nature of the client involved in obtaining an access token.

Authorization Code Grant

When it should be used?

It should be used as soon as the client is a web server. It allows you to obtain a long-lived access token since it can be renewed with a refresh token (if the authorization server enables it).

Example:

- Resource Owner: you
- Resource Server: a Google server
- Client: any website
- Authorization Server: a Google server

Scenario:

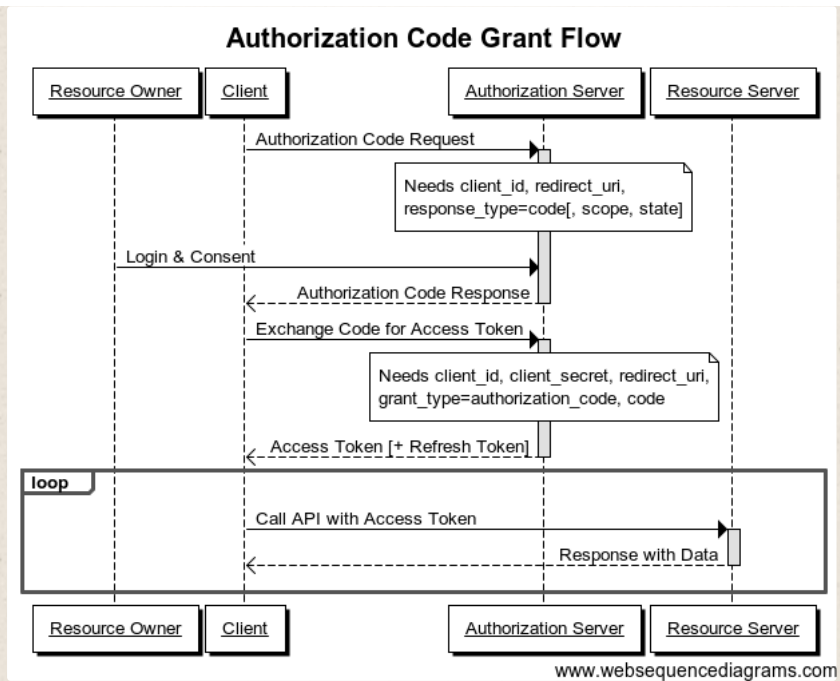
1. A website wants to obtain information about your Google profile.
2. You are redirected by the client (the website) to the authorization server (Google).
3. If you authorize access, the authorization server sends an authorization code to the client (the website) in the callback response.
4. Then, this code is exchanged against an access token between the client and the authorization server.
5. The website is now able to use this access token to query the resource server (Google again) and retrieve your profile data.

You never see the access token, it will be stored by the website (in session for example). Google also sends other information with the access token, such as the token lifetime and eventually a refresh token.

This is the ideal scenario and the safer one because the access token is not passed on the client side (web browser in our example).

More information: [RFC 6749 — Authorization Code Grant](#).

Sequence diagram:



Implicit Grant

When it should be used?

It is typically used when the client is running in a browser using a scripting language such as Javascript. This grant type does not allow the issuance of a refresh token.

Example:

- Resource Owner: you
- Resource Server: a Facebook server
- Client: a website using **AngularJS** for example
- Authorization Server: a Facebook server

Scenario:

1. The client (AngularJS) wants to obtain information about your Facebook profile.
2. You are redirected by the browser to the authorization server (Facebook).
3. If you authorize access, the authorization server redirects you to the website with the access token in the URI fragment (not sent to the web server). Example of callback: `http://example.com/oauthcallback#access_token=MzJmNDc3M2VjMmQzN.`
4. This access token can now be retrieved and used by the client (AngularJS) to query the resource server (Facebook). Example of query: `https://graph.facebook.com/me?access_token=MzJmNDc3M2VjMmQzN.`

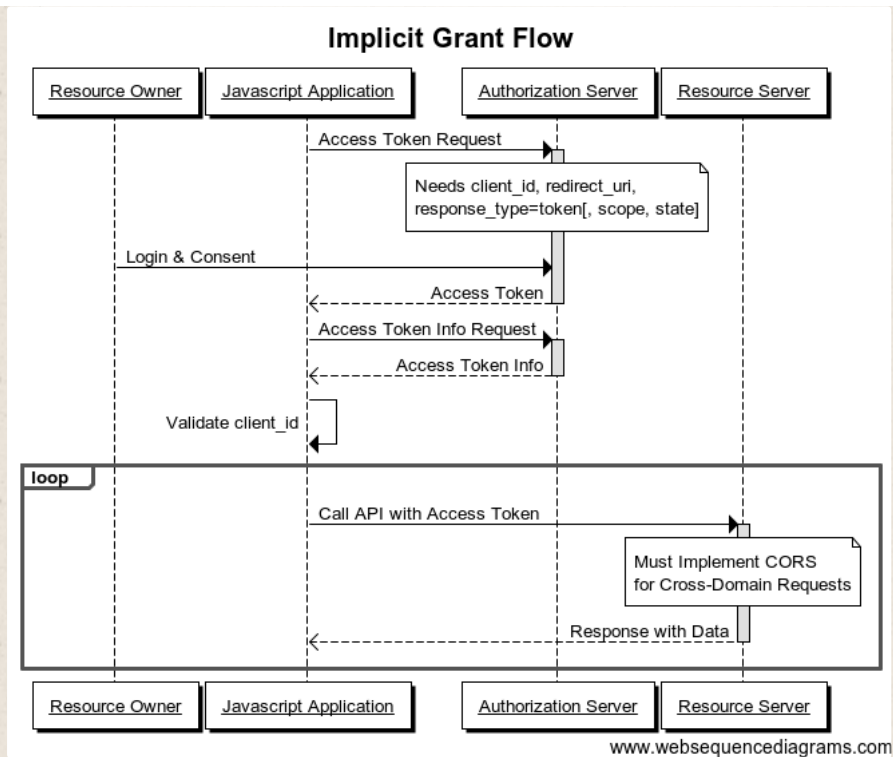
Maybe you wonder how the client can make a call to the Facebook API with Javascript without being blocked because of the **Same Origin Policy**? Well, this cross-domain request is possible because Facebook authorizes it thanks to a header called `Access-Control-Allow-Origin` present in the response.

More information about Cross-Origin Resource Sharing (CORS): https://developer.mozilla.org/en-US/docs/HTTP/Access_control_CORS#The_HTTP_response_headers.

Attention! This type of authorization should only be used if no other type of authorization is available. Indeed, it is the least secure because the access token is exposed (and therefore vulnerable) on the client side.

More information: [RFC 6749 — Implicit Grant](#).

Sequence diagram:



Resource Owner Password Credentials Grant

When it should be used?

With this type of authorization, the credentials (and thus the password) are sent to the client and then to the authorization server. It is therefore imperative that there is absolute trust between these two entities. It is mainly used when the client has been developed by the same authority as the authorization server. For example, we could imagine a website named example.com seeking access to protected resources of its own subdomain api.example.com. The user would not be surprised to type his login/password on the site example.com since his account was created on it.

Example:

- Resource Owner: you having an account on acme.com website of the Acme company
- Resource Server: Acme company exposing its API at api.acme.com
- Client: acme.com website from Acme company
- Authorization Server: an Acme server

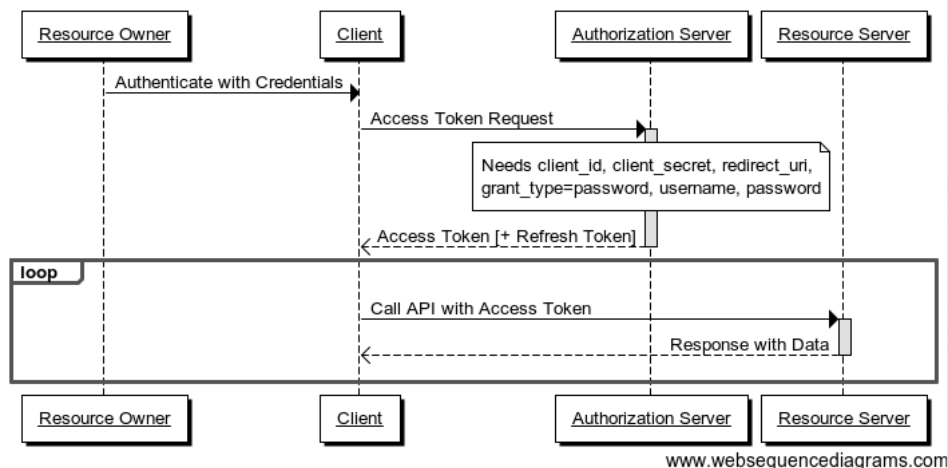
Scenario:

1. Acme company, doing things well, thought to make available a RESTful API to third-party applications.
2. This company thinks it would be convenient to use its own API to avoid reinventing the wheel.
3. Company needs an access token to call the methods of its own API.
4. For this, company asks you to enter your login credentials via a standard HTML form as you normally would.
5. The server-side application (website acme.com) will exchange your credentials against an access token from the authorization server (if your credentials are valid, of course).
6. This application can now use the access token to query its own resource server (api.acme.com).

More information: [RFC 6749 — Resource Owner Password Credentials Grant](#).

Sequence diagram:

Resource Owner Password Credentials Grant Flow



Client Credentials Grant

When it should be used?

This type of authorization is used when the client is himself the resource owner. There is no authorization to obtain from the end-user.

Example:

- Resource Owner: any website
- Resource Server: Google Cloud Storage
- Client: the resource owner
- Authorization Server: a Google server

Scenario:

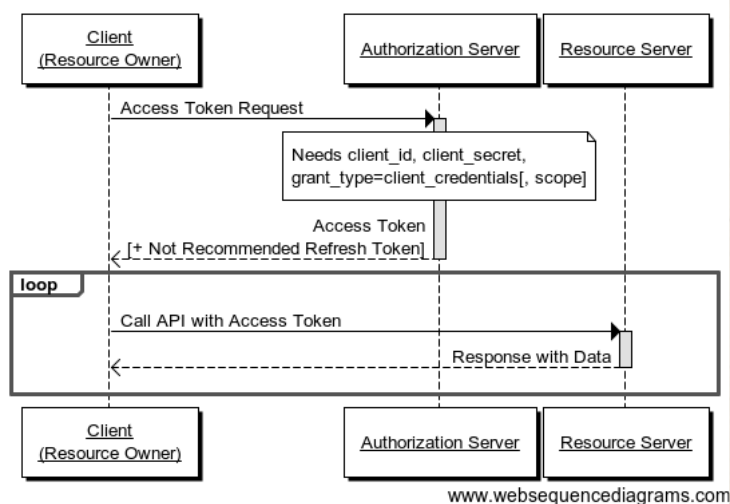
1. A website stores its files of any kind on Google Cloud Storage.
2. The website must go through the Google API to retrieve or modify files and must authenticate with the authorization server.
3. Once authenticated, the website obtains an access token that can now be used for querying the resource server (Google Cloud Storage).

Here, the end-user does not have to give its authorization for accessing the resource server.

More information: [RFC 6749 — Client Credentials Grant](#).

Sequence diagram:

Client Credentials Grant Flow



Access token usage

The access token can be sent in several ways to the resource server.

Request parameter (GET or POST)

Example using GET: https://api.example.com/profile?access_token=MzJmNDc3M2VjMmQzN

This is not ideal because the token can be found in the access logs of the web server.

Authorization header

GET /profile HTTP/1.1

Host: api.example.com

Authorization: Bearer MzJmNDc3M2VjMmQzN

It is elegant but all resource servers do not allow this.

Security

OAuth2 is sometimes criticized for its permeability, but it is often due to bad implementations of the protocol. There are big mistakes to avoid when using it, here are some examples.

Vulnerability in Authorization Code Grant

There is a vulnerability in this flow that allows an attacker to steal a user's account under certain conditions. This hole is often encountered and also in many known websites (such as Pinterest, SoundCloud, Digg, ...) that have not properly implemented the flow.

Example:

- Your victim has a valid account on a website called A.
- The A website allows a user to login or register with Facebook and is previously registered as a client in Facebook OAuth2 authorization server.
- You click on the Facebook Connect button of website A but do not follow the redirection thanks to Firefox **NoRedirect** addon or by using **Burp** for example (callback looks like this: <http://site-internet-a.com/facebook/login?code=OGI2NmY2NjYxN2Y4YzE3>).
- You get the url (containing the authorization code) to which you would be redirected (visible in Firebug).
- Now you have to force your victim to visit this url via a hidden iframe on a website or an image in an email for example.
- If the victim is logged in website A, jackpot! Now you have access to the victim's account in website A with your Facebook account. You just have to click on the Facebook Connect button and you will be connected with the victim's account.

Workaround:

There is a way to prevent this by adding a "state" parameter. The latter is only recommended and not required in the specifications. If the client sends this parameter when requesting an authorization code, it will be returned unchanged by the authorization server in the response and will be compared by the client before the exchange of the authorization code against the access token. This parameter generally matches to a unique hash of a random number that is stored in the user session. For example in PHP: `sha1(uniqid(mt_rand(), true))`.

In our example, if the website A was using the parameter "state", he would have realized in the callback that the hash does not match the one stored in the session of the victim and would therefore prevented the theft of victim's account.

More information: [RFC 6749 — Cross-Site Request Forgery](#).

Vulnerability in Implicit Grant

This type of authorization is the least secure of all because it exposes the access token to client-side (Javascript most of the time). There is a widespread hole that stems from the fact that the client does not know if the access token was generated for him or not (**Confused Deputy Problem**). This allows an attacker to steal a user account.

Example:

- An attacker aims to steal a victim's account on a website A. This website allows you to connect via your Facebook account and uses implicit authorization.
- The attacker creates a website B allowing login via Facebook too.
- The victim logs in to the website B with his Facebook account and therefore implicitly authorized the generation of an access token for this.
- The attacker gets the access token via his website B and uses it on website A by modifying the access token in the URI fragment. If website A is not protected against this attack, the victim's account is compromised and the attacker has now access to it.

Workaround:

To avoid this, the authorization server must provide in its API a way to retrieve access token information. Thus, website A would be able to compare the client_id of the access token of the attacker against its own client_id. As the stolen access token was generated for the website B, client_id would have been different from client_id of website A and the connection would have been refused.

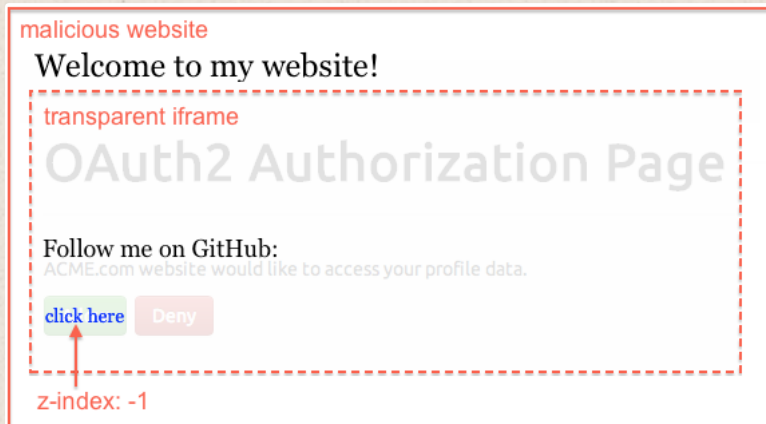
Google describes this in its API documentation: <https://developers.google.com/accounts/docs/OAuth2Login#validatingtoken>.

More information in RFC: <http://tools.ietf.org/html/rfc6819#section-4.4.2.6>

Clickjacking

This technique allows the attacker to cheat by hiding the authorization page in a transparent iframe and getting the victim to click a link that is visually over the "Allow" button of the authorization page.

Example:



Workaround:

To avoid this, it is necessary that the authorization server returns a header named X-Frame-Options on the authorization page with the value DENY or SAMEORIGIN. This prevents the authorization page to be displayed in an iframe (DENY) or requires consistency between the domain name of the main page and the domain name specified in the iframe "src" attribute (SAMEORIGIN).

This header is not standard but is supported in the following browsers: IE8+, Firefox3.6.9+, Opera10.5+, Safari4+, Chrome 4.1.249.1042+.

More information: <https://developer.mozilla.org/en-US/docs/HTTP/X-Frame-Options>.

Here is the RFC that lists the potential vulnerabilities in the protocol implementations and the countermeasures: <http://tools.ietf.org/html/rfc6819>.

Conclusion


Like it or not, (troll from a former contributor of the protocol), OAuth2 seems to impose itself for several years as a standard solution for the delegation of authority between different applications.

In any case, I hope I have helped you understand its workings. We will see in another article how to create your own OAuth2 authorization server with Symfony2.

++ and sorry for the broken English.

By Johann Reinke — In OAuth — Tags: [oauth](#), [oauth2](#)

48 Responses to “Understanding OAuth2”


1.  Eric says:

Thursday May 16th, 2013 at 11:30 AM

Un grand merci pour cet article. Ca fait vraiment plaisir de trouver du contenu d'une telle qualité, claire, bien expliqué, et en plus en Français. Bravo!

J'attends avec impatience la suite avec OAuth2 et Symfony2, car je pense que beaucoup galère sur ce sujet... moi le premier.

[Reply](#)

2.  bebshow says:

Monday July 8th, 2013 at 01:31 PM

Salut ! J'aime beaucoup cet article, clair, précis, avec citations, liens etc.

parfait merci and

Deeply the next article on symfony2 !

[Reply](#)

3.  adrien says:

Friday July 12th, 2013 at 12:12 AM

Merci beaucoup !

Je cherchais à réaliser des web services REST utilisant OAuth avec Symfony 2 sans vraiment avoir de connaissances sur le sujet et cet article m'a bien éclairci les idées !

Est-il possible de savoir où en est le tuto "comment créer son propre serveur d'autorisation OAuth2 avec Symfony2" ? Comme mon prédécesseur, je galère un peu sur le sujet et si un tuto de qualité égale à cet article était mis en ligne il simplifierait le travail de beaucoup de personnes !

[Reply](#)




Johann Reinke says:

Friday July 12th, 2013 at 07:22 AM

Merci à tous,

J'ai commencé l'article sur l'API Symfony2 avec serveur OAuth2. Pas encore eu le temps de terminer mais je n'oublie pas 😊

[Reply](#)

4.  JC says:


Monday September 30th, 2013 at 10:13 AM

Bravo !

C'est un article clair et très facile à comprendre, pourtant le sujet ne l'était pas.

Super sympa d'avoir pris du temps pour expliquer tout cela.

[Reply](#)


5.  Uneak says:

Thursday February 20th, 2014 at 09:53 PM

Sujet difficile, mais malgré tout SUPER clair !

Félicitation !!


[Reply](#)

6.  David says:

Wednesday March 12th, 2014 at 02:15 PM

Cool post, thanks!


[Reply](#)

7.  Flo63 says:

Tuesday April 22nd, 2014 at 01:45 PM

Merci ! tout est très clair !

[Reply](#)


8.  Laurent says:

Tuesday June 3rd, 2014 at 06:13 PM

Bonjour

Bravo pour cet article. Par contre, qualifier de "troll" le "coming-out" du leader des spécifications me semble inapproprié. Il ne faut pas minorer ce fait, qui montre qu'il y a des problèmes.


[Reply](#)

9.  Narayan says:

Tuesday July 1st, 2014 at 09:27 PM

Many thanks for this great article (and the flows).

[Reply](#)

10.  Max says:


Thursday October 16th, 2014 at 03:09 AM

Salut,

Quel est l'intérêt d'échanger un autorization code par un token, je veux dire pourquoi ne pas retourner directement le token puisque c'est le même serveur et le même client qui communiquent ?

Merci !


[Reply](#)

11.  Jean-Marc says:

Sunday October 19th, 2014 at 12:37 AM

Enfin un article clair sur le sujet. J'ai particulièrement apprécié les diagrammes de séquence. Continue ce super boulot !


[Reply](#)

12.  Sébastien says:

Tuesday October 21st, 2014 at 07:56 PM

Très bon article, merci.

[Reply](#)

13.  Zouhair says:

Tuesday November 18th, 2014 at 11:05 PM

Bon article, super

[Reply](#)

14.  jbn says:

Wednesday December 17th, 2014 at 01:51 PM

Merci pour l'article.

Je ne trouve pas l'article suivant sur comment réaliser son propre serveur oauth ?

J'ai trouvé le bundle Symfony2 FOSAuthServerBundle mais une explication claire m'aiderait à l'intégrer dans notre environnement.

Cordialement et bonne continuation !

[Reply](#)


15.  Xavier says:

Tuesday January 27th, 2015 at 08:40 PM

Excellent article.

Merci 😊

[Reply](#)


16.  Séyram says:

Wednesday April 15th, 2015 at 06:47 PM

C'est l'article est très bien écrit, le sujet est difficile mais on en ressort bien éclairé après la lecture. Merci beaucoup

Bonne continuation.


[Reply](#)

17.  BlackDev'S says:

Thursday April 16th, 2015 at 11:24 PM

Je te remercie pour cet article sur l'OAuth 2.0 j'apprends actuellement à utiliser les API et grâce à toi j'ai beaucoup avancé !

[Reply](#)

18.  Kashouf says:


Tuesday April 28th, 2015 at 10:19 AM

Bonjour, merci pour ce superbe article. Tout est bien expliqué et j'en apprend enfin un peu plus sur ce protocole. Les diagrammes de séquences sont bien utiles aussi. Tout ce qu'il manque à la limite, c'est un peu de code pour illustrer son utilisation...

Je suis confronté à un problème de modification et de copie de documents dans le drive, si vous avez des exemples du Client Credentials Grant je suis preneur ^^.

Merci


[Reply](#)

19.  hordanh ongouo says:

Tuesday July 7th, 2015 at 07:45 PM

merci énormément pour cet article car ça m'a beaucoup aidé.

[Reply](#)


20.  Pierre de LESPINAY says:

Thursday July 23rd, 2015 at 05:18 PM

Les graphiques les plus parlants que j'ai pu voir au sujet d'OAuth2 jusqu'à maintenant.

Merci.


[Reply](#)

21.  Thierry Templier says:

Monday August 10th, 2015 at 09:54 AM

Super article! Très bien expliqué et clair! Merci beaucoup

[Reply](#)

22.  Sébastien B says:

Wednesday August 26th, 2015 at 05:13 PM

Bonjour,

En effet un grand merci, très bon travail de vulgarisation.

En revanche j'avoue que je n'arrive pas à comprendre l'intérêt de ce protocole en terme de sécurité, même suite à la lecture de cet article


En particulier la multiplicité des "méthodes" possibles.

J'ai travaillé avec OAuth v1 et j'ai cru comprendre que l'intérêt résidait dans la "sécurité" qu'apportait le navigateur utilisé comme étant "neutre" pour l'utilisateur et rendant fiable le mécanisme.

Si j'ai bien compris, mis à part facilité, OAuth2 permet notamment d'être exploité sur application mobile native ce qui n'est pas le cas d'OAuth v1, mais dans ce cas, il semble que niveau sécurité ce ne soit "que" du https, rien de plus, l'autorisation de partage pourra être bypassée par l'appli qui controle totalement l'environnement

Qu'est ce que j'ai manqué?


[Reply](#)

23.  VaN says:

[Wednesday September 2nd, 2015 at 12:00 PM](#)

Toujours pas de nouvelles de l'article sur l'intégration de OAuth dans Symfony2 ? Je ne me fais pas trop d'illusions, au bout de 2 ans :p


[Reply](#)

24.  Jean Mardochée JOSEPH says:

[Thursday September 3rd, 2015 at 09:50 AM](#)

Bravo, c'est très explicite et très pédagogique, vivement la suite "OAuth2 avec Symfony2", bon courage et bonne continuation 😊

[Reply](#)

25.  Nam's says:

[Thursday September 24th, 2015 at 05:31 PM](#)

Bonjour et merci pour tes explications,

j'ai cependant quelques questions.

Je récupère en PHP via Curl les info relatives a mon compte google analytics et je passe donc mes parametres dans l'URL.

```
$url = 'https://www.googleapis.com/analytics/v3/data/realtime?ids=ga%3A104738580&metrics=rt%3AActiveUsers&access_token=ya29.-AF73Jm4hjBtqlVrK1o8RNhaJot1S4rY98GU6md0PHK9AW-Jve9jreSJzSNBvbXHZ7xr';
```

Le probleme est que ce token n'a une durée de vie que d'une heure.

Je me suis donc généré un refresh-token mais je ne sais pas comment l'utiliser dans mon code.

Où le placer dans l'url ?

Quelle manipe faire avec ?

Si quelqu'un peut m'aider ca serait top.

Merci d'avance.

Nam's.

[Reply](#)

26.  GM_CED says:

[Monday October 5th, 2015 at 02:48 PM](#)

Merci pour cette explication!

[Reply](#)

27.  nanou says:

[Friday November 6th, 2015 at 08:48 PM](#)

ça mériterait de figurer sur un wiki, excellent travail, merci pour le partage !

[Reply](#)


28.  techtech says:

[Friday January 15th, 2016 at 09:24 AM](#)

Super article;

Le diagramme des séquences est super clair et est le parfait complément !

[Reply](#)

29.  L. Truong says:

[Thursday March 3rd, 2016 at 05:11 PM](#)

Merci pour les explications qui font bien le lien avec les exemples d'implémentation en Java Servlet chez IBM pour les rois types d'autorisations (sauf implicit) :

<http://www.ibm.com/developerworks/library/se-oauthjavapt3/>

Si quelqu'un trouve des meilleurs exemples, merci de bien vouloir partager.

[Reply](#)


30.  aload says:

[Friday April 22nd, 2016 at 09:35 PM](#)

Bonjour,
Super tuto merci, mais un truc m'échappe, il faut s'enregistrer en tant que client auprès du serveur d'autorisation. OK
Mais le client == utilisateur ??? ou client == application (JS, android, php)
car je ne sais pas si je doit générer une seul paire de client_id + secret (ayant qu'une appli JS) ou alors, chaque utilisateur se voit obtenir une paire de clef.

ha oui "ressource owner" c'est bien un utilisateur ?

[Reply](#)

31.  Pavan Vadlmudi says:

[Tuesday May 3rd, 2016 at 12:32 PM](#)

Very Useful POST

[Reply](#)

32.  FL says:

[Tuesday May 10th, 2016 at 03:51 PM](#)

Merci pour les explications.

Mais je ne comprend pas une partie:
Lorsque le "Authorization Server" délivre un token, comment le "Ressource Server" sait que le token est valide ? Quelles méthode sont utilisées pour que ces deux la soient d'accords sur la validité (et l'expiration) d'un token ?

Pour les tokens d'accès (exemple : "Authorization: Bearer MzJmNdc3M2VjMmQzN"), en quoi ça change d'une Basic Auth ?

[Reply](#)

33.  TOMASIAN says:

[Wednesday May 18th, 2016 at 03:01 PM](#)

Excellent article. Clair et précis. Merci pour ce superbe travail.

Cordialement,

[Reply](#)


34.  fred says:

[Thursday June 9th, 2016 at 05:25 PM](#)

Merci pour votre super article, très didactique.

Par contre, je n'ai pas bien compris l'exemple de faille que vous décrivez.

[Reply](#)


35.  Praveen says:

[Wednesday June 22nd, 2016 at 09:51 AM](#)

Great work . Nicely explain the terminology with examples.

Thanks a lot Johann Reinke.

[Reply](#)

36.  Gonzalo says:

[Friday July 15th, 2016 at 08:38 PM](#)

Great article, When it should be used? made the difference from other articles in the internet.

Do you have anything written about OAuth1?

[Reply](#)

37.  Lala says:

[Sunday July 17th, 2016 at 08:12 PM](#)

Il se peut que je ne saisit pas l'idée de l'auteur mais il semble que la phrase ci dessous

Ce code est échangé (entre le site internet et Google) par un token d'accès de façon transparente pour vous.


devrait se lire ainsi

Ce code est échangé (entre le site internet et Google) par un token d'accès de façon opaque pour vous.

En effet on ne voit pas ce qui s'échange entre ces 2 sites alors c'est plutot opaque que transparente...

Merci de me corriger si besoin svp...

[Reply](#)

38.  Brian Chandler says:

[Tuesday July 19th, 2016 at 09:16 PM](#)

Thank you very much for this excellent explanation. The diagrams and when each flow tends to be used really cut through the fog.


[Reply](#)

39.  Kavita says:

Thursday September 15th, 2016 at 06:02 PM

Good explanation

[Reply](#)

40.  Michael says:

Wednesday September 21st, 2016 at 10:13 AM


Hi Johann,

thanks for this useful POST. I guess there is a little mistake in Chapter Roles:

Authorization Server: server issuing access token to the client. This token will be used for the client to request the resource server. This server can be the same as the authorization resource server (same physical server and same application), and it is often the case.

Regards,
Michael


[Reply](#)

41.  Syed says:

Thursday October 20th, 2016 at 10:03 AM

One of the best post so far on OAuth2 explanantion. Thanks for such a informative and clear post about OAuth.


[Reply](#)

42.  Gimhani says:

Thursday November 3rd, 2016 at 03:21 AM

A very useful article
Thank you

[Reply](#)

43.  Bhuvan Doshi says:

Sunday November 6th, 2016 at 02:01 PM

Thanks for useful post. salute for your knowledge 😊

[Reply](#)

44.  Mantas says:


Thursday November 24th, 2016 at 11:30 AM

With regards to definition of the Roles:

...
Authorization Server: server issuing access token to the client. This token will be used for the client to request the resource server. This server can be the same as the authorization server (same physical server and same application), and it is often the case.
...

Am I missing something in the above or does it say "... Authorization server ... can be the same as the authorization server" ?

[Reply](#)

 hakan says:

Tuesday November 29th, 2016 at 02:45 PM

Focus on "This server". It refers to "the resource server" mentioned in the preceding sentence.

[Reply](#)

45.  Ashok says:

Wednesday November 30th, 2016 at 11:09 AM

Excellent explanation. Many of my confusions regarding grant types got cleared. Thank you man

[Reply](#)

46.  Ritwik says:

Monday December 5th, 2016 at 11:48 PM

Finally I could found a post – that clears all the flows. Thanks a lot.

[Reply](#)

Leave a Reply

Name *

Mail (will not be published) *

Website

Comment *

* Required fields

Languages

- [English](#)
- [Français](#)

Categories

- [elasticsearch](#) (1)
- [Javascript](#) (1)
- [OAuth](#) (1)
- [PHP](#) (25)
 - [Magento](#) (23)
 - [Silex](#) (1)

© 2016 BubbleCode, by Johann Reinke