RFC 6749 : The OAuth 2.0 Authorization Framework

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 13 octobre 2012

Date de publication du RFC : Octobre 2012

https://www.bortzmeyer.org/6749.html

Le protocole OAuth a désormais sa seconde version, OAuth 2.0, normalisée dans ce RFC. OAuth permet de à un programme de s'authentifier sur le Web, pour accéder à un jeu limité de services, qu'un utilisateur lui a autorisé.

OAuth est donc un protocole d'authentification d'un tiers, le **client**, qui veut accéder à une ressource, par exemple un fichier, située sur un **serveur** et dont le contrôle revient à un **propriétaire**, le "*resource owner*". (OpenID vise à authentifier un utilisateur humain, OAuth à authentifier la requête d'un programme, agissant pour le compte d'un humain.) Prenons un exemple (vous en trouverez d'autres dans le RFC, comme l'exemple classique de l'impression de photos, mais celui que j'indique a l'avantage d'être un exemple réel) : le service de microblogging Twitter permet à des utilisateurs (les **propriétaires**, dans la terminologie OAuth) de faire connaître au monde entier leurs pensées profondes <http://www.henrymichel.com/humour/top10-relous-twitter/> en moins de 140 caractères. Le succès de ce service et l'existence d'une bonne (car très simple) API a poussé à l'arrivée de nombreux services tiers qui ont tous en commun de devoir accéder au compte Twitter de l'utilisateur (par exemple, Twibbon <http://www.twibbon.com/> ou TwitterCounter <http://www.twittercounter.com/>). Une façon possible, pour ces services, d'accéder au compte Twitter de l'utilisateur est de lui demander son nom et son mot de passe. On voit les conséquences que cela peut avoir pour la sécurité : le service doit stocker le mot de passe (le fera-t-il de façon sûre ?), le service peut abuser de son privilège, puisqu'il a tous les droits du propriétaire, le serveur doit gérer l'authentification par mot de passe (pas la meilleure), retirer un droit d'accès à un service veut dire changer le mot de passe, retirant ainsi le droit d'accès à tous les services tiers, etc.

OAuth fournit une meilleure solution : le **serveur**, en recevant la requête du **client**, demande au **propriétaire** l'autorisation :

Cette demande se fait typiquement via un formulaire HTML.

Le service tiers reçoit alors un "*token*", une autorisation qui lui est spécifique, et ne lui donne que certains droits. Il présentera ce "*token*" au serveur pour obtenir l'accès.

Tout se fait par redirections HTTP (RFC 7231 [1], section 6.4 et ce RFC 6749, section 3). Bien sûr, pour que cela soit sûr, il y a de nombreux détails à prendre en compte.

Si OAuth v1 (RFC 5849) était juste l'officialisaton à l'IETF d'un protocole qui avait été développé à l'extérieur, cette nouvelle version 2 marque des changements profonds, et incompatibles avec l'ancienne version. C'est d'abord une refonte de la norme, d'un seul document à un ensemble dont ce nouveau RFC 6749 n'est que le document de cadrage. Les documents décrivant effectivement le protocole ne sont pour la plupart pas encore parus (il y a par exemple un projet de RFC sur les scénarios d'usage, et un entièrement consacré à la sécurité, via une études des menaces portant sur OAuth). Seuls les RFC 6750, sur un type particulier de "*token*", et RFC 6755 sur des URN pour OAuth, accompagnent ce nouveau OAuth. C'est aussi le passage d'un protocole relativement simple à un ensemble de protocoles, partageant un cadre commun mais n'étant plus forcément interopérables (section 1.8), puisque OAuth v2 permet bien des variantes. C'est complètement contraire aux principes posés dans le RFC 6709, sur les extensions aux protocoles. Donc, attention, un service OAuth v1 ne peut pas interagir avec un serveur OAuth v2 (quand ils existeront) ou le contraire. Les programmeurs OAuth sont encouragés à repartir de zéro, vue la différence des deux protocoles.

Une fois que Twitter a adopté ce protocole <http://www.readwriteweb.com/archives/why_twitters_new_oauth_matters.php> (notez que, à la date de publication de ce RFC, Twitter utilise OAuth v1 et n'a pas annoncé d'intention de passer à la v2) et documenté son usage <http://apiwiki.twitter.com/OAuth-FAQ>, la plupart des services tiers l'ont intégré (par exemple Twibbon <http://blog.twibbon.com/oauth-is-live>).

Lisons maintenant le RFC. Premièrement, les concepts (section 1) et le vocabulaire (section 1.1). Ce qui se jouait autrefois à deux (le client et le serveur) est désormais fréquemment une partie à trois ("*OAuth Love Triangle*", dit Leah Culver <http://leahculver.com/>), le **client** ("*client*", Twibbon dans le cas plus haut), le **serveur** ("*server*", Twitter dans le cas plus haut) et le **propriétaire** ("*resource owner*", moi <http://twitter.com/bortzmeyer>). Notez qu'OAuth v2 a transformé le triangle en carré, en passant à quatre rôles, le serveur ayant été séparé en deux, le serveur qui contrôle l'accès à la ressource convoitée, et celui qui donne les autorisations. En pratique, on ne sait pas encore si des serveurs suivront cette séparation.

Pour voir le cheminement complet d'une authentification OAuth, on peut regarder la jolie image de Yahoo <http://developer.yahoo.com/oauth/guide/images/oauth_graph.gif> (mais attention, elle utilise une ancienne terminologie) ou bien suivre l'exemple de la section 1.2 du RFC. Si le client, le service d'impression `printer.example.com` veut accéder aux photos stockées sur le serveur `photos.example.net`, et dont le propriétaire est Jane :

— Le client et le serveur doivent déjà avoir un accord entre eux, préalable, avec un secret partagé pour s'authentifier. OAuth ne permet pas à un nouveau client inconnu d'en bénéficier.
— Le client doit avoir lu la documentation du serveur, qui indique entre autre les URL à contacter.
— Lorsque Jane visite le site du client, ici l'imprimeur, et demande l'impression de ses photos, le client contacte le serveur en HTTP pour demander un "*token*", une suite de nombres aléatoires qui identifie de manière unique cette requête (on peut voir ce "*token*" dans la copie d'écran ci-dessus, c'est le paramètre `oauth_token`).

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7231.txt`

— Le client doit alors utiliser les redirections HTTP pour envoyer Jane sur le site du serveur, afin qu'elle donne l'autorisation nécessaire. Les paramètres de la redirection indiquent évidemment le "*token*".
— Si Jane donne son accord (cela peut être un accord implicite, voir par exemple comment fait Twitter <http://apiwiki.twitter.com/Sign-in-with-Twitter>), le serveur signe la demande et renvoie le navigateur de celle-ci vers le client avec, dans les paramètres, une demande approuvée et signée (paramètre `oauth_signature`).
— Le client peut alors demander au serveur des autorisations pour les photos, puis demander les photos elle-mêmes.

Un guide complet sur OAuth v1 est disponible en <http://hueniverse.com/oauth/guide/>. Une autre bonne introduction à OAuth v1 est « "*Gentle introduction to OAuth*" <http://dev.opera.com/articles/view/gentle-introduction-to-oauth/> ». Parmi les applications amusantes d'OAuth, un curl OAuthisé en <http://groups.google.com/group/twitter-api-announce/browse_thread/thread/788a1991c99b66df>. Un exemple de programmation OAuth v1 pour accéder à Twitter en Python figure dans mon article <https://www.bortzmeyer.org/twitter-oauth.html>. Mais OAuth v2 ne bénéficie pas encore de telles documentations.

OAuth v2 a suscité de nombreuses critiques, notamment sur sa complexité. Mon pronostic est qu'OAuth v2, victime du "*Second System Effect*", ne sera guère déployé, la grande majorité des sites Web restant à la v1.

Une bonne critique qui synthétise la plupart des remarques qui ont été faites sur OAuth v2 est « "*OAuth 2.0 and the Road to Hell*" <http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/> ». Eran Hammer, l'auteur, est marqué dans l'annexe C de ce RFC comme ayant participé à OAuth v2 jusqu'à la version 26 du brouillon (la version publiée est la 31). Il a ensuite cessé sa participation au projet et s'en explique dans cet article. Ses principaux arguments :

— Le cadre est trop général, ne permettant pas de garantir l'interopérabilité de deux mises en œuvre d'OAuth v2 (ce point est admis dans la section 1.8 de ce RFC 6749). C'est au point qu'un très grand nombre de protocoles incompatibles entre eux pourraient légitimement se nommer « conformes à OAuth »..
— OAuth v1 était simple et utilisable, issu d'un monde de "*startups*" Web qui voulaient une solution « bien assez bonne » et tout de suite. OAuth v2 a été chargé par des exigences de PHB comme de mentionner SAML et autres technologies des « gens de l'entreprise » (par opposition aux « gens du Web »). C'est une tendance analogue qui a tué les "*Web Services*".
— Étant plus complexe et plus adaptable à des besoins spécifiques, OAuth v2 sera moins sûr.