RFC 5849 : The OAuth 1.0 Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 3 mai 2010

Date de publication du RFC : Avril 2010

https://www.bortzmeyer.org/5849.html

Le protocole OAuth, déjà fréquemment déployé, voit son développement officiellement passer du côté de l'IETF, avec ce RFC 5849 [1] qui reprend la description de la version 1 du protocole, pendant que le groupe de travail Oauth <http://tools.ietf.org/wg/oauth> travaille à la version 2 (dont le premier document publié a été le RFC 6749, en octobre 2012).

OAuth, parent de OpenID, est 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**. (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, Auto FollowFriday <http://www.autoFF.com/>, Twibbon <http://www.twibbon.com/> ou TwitterCounter <http://www.twittercounter.com/> - à noter que le premier des trois n'est pas encore passé à Oauth). 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... OAuth fournit une meilleure solution : le **serveur**, en recevant la requête du **client**, demande au **propriétaire** l'autorisation :

Tout se fait par redirections HTTP (RFC 2616, section 10.3). Bien sûr, pour que cela soit sûr, il y a de nombreux détails à prendre en compte, ce qui explique les quarante pages du RFC.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5849.txt

OAuth existe depuis plusieurs années et était géré par un groupe informel <http://oauth.net/> (voir la section 1 du RFC pour un historique). Il est désormais documenté dans ce RFC 5849 (la version documentée n'a pas subi de changement significatif à l'IETF, c'est surtout une officialisation, avec quelques corrections de bogues, l'annexe A détaille ces changements ; le plus gênant est la refonte complète de la terminologie) et les nouvelles versions sont maintenant développées à l'IETF.

Une fois que Twitter a adopté ce protocole <http://www.readwriteweb.com/archives/why_twitters_new_oauth_matters.php> 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, 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 que la terminologie OAuth a changé avec ce RFC (par exemple, le propriétaire était nommé « utilisateur » - "*user*").

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 l'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.ne` 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`).
- 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.

La requête initiale du client auprès du serveur pourra ressembler à :

```
POST /initiate HTTP/1.1
Host: photos.example.net
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131200",
        oauth_nonce="wIjqoS",
        oauth_callback="http%3A%2F%2Fprinter.example.com%2Fready",
        oauth_signature="74KNZJeDHnMBp0EMJ9ZHt%2FXKycU%3D"
```

Si le serveur renvoie le "*token*" `hh5s93j4hdidpola`, la redirection de Jane vers le serveur pourra se faire via un URL comme :

```
https://photos.example.net/authorize?oauth_token=hh5s93j4hdidpola
```

et, une fois l'authentification de Jane auprès du serveur et son autorisation de la requête effectuées, le client pourra demander son autorisation pour le "*token*" `hh5s93j4hdidpola` :

```
POST /token HTTP/1.1
Host: photos.example.net
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_token="hh5s93j4hdidpola",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131201",
        oauth_nonce="walatlh",
        oauth_verifier="hfdp7dh39dks9884",
        oauth_signature="gKgrFCywp7rO0OXSjdot%2FIHF7IU%3D"
```

Et enfin, après encore un nouveau "*token*", le client pourra demander la photo :

```
GET /photos?file=vacation.jpg&size=original HTTP/1.1
Host: photos.example.net
Authorization: OAuth realm="Photos",
        oauth_consumer_key="dpf43f3p2l4k3l03",
        oauth_token="nnch734d00sl2jdk",
        oauth_signature_method="HMAC-SHA1",
        oauth_timestamp="137131202",
        oauth_nonce="chapoH",
        oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

Voici pour le principe. Le reste du RFC est consacré aux détails. D'abord, comment obtenir son "*token*" ? Comme indiqué dans l'exemple, la méthode la plus courante est la redirection HTTP, normalisée en détail dans la section 2. Elle nécessite trois URL à connaître par le client. Par exemple, ceux de Twitter (apparemment documentés uniquement dans le source `<http://apiwiki.twitter.com/OAuth-Examples>`) sont :

— Demande d'un "*token*" initial (section 2.1 du RFC) : `https://twitter.com/oauth/request_token` (le terme de "*request token*" fait référence à l'ancienne terminologie, dans le RFC, vous trouverez le même concept sous le nom de "*temporary credentials*", cd. section 1.1),
— Renvoi du propriétaire vers la page d'autorisation (section 2.2) : `http://twitter.com/oauth/authorize`,
— Accès à la ressource convoitée via un autre "*token*" (section 2.3) : `https://twitter.com/oauth/access_token`.

Comment authentifier toutes ces requêtes ? C'est le but de la section 3. À noter qu'il existe plusieurs méthodes, dépendant du serveur. Le paramètre `oauth_signature_method` indique celle choisie, cela peut utiliser de la cryptographie asymétrique - cf. RFC 3447 -, un condensat incluant un secret partagé - cf. RFC 2104 -, voire même du texte brut, qui doit alors évidemment être emballé dans https. Les sections 3.4 à 3.6 détaillent la canonicalisation à pratiquer et autres formalités.

Le but de Oauth étant la sécurité, il n'est pas étonnant que la section résumant les questions de sécurité (section 4) soit longue. Parmi les points passés en revue :

— Le fait qu'un protocole, si parfait qu'il soit, ne protège pas contre les erreurs dues à de mauvaises procédures. Par exemple, si le serveur ne vérifie pas l'authenticité des propriétaires avant qu'ils donnent leur autorisation... (Cf. RFC 7235.)
— Si on utilise la signature RSA, le secret partagé n'est pas utilisé et la sécurité dépend donc du secret de la clé privée RSA (section 4.1). Par contre, si on utilise les secrets partagés, ils doivent être stockés en clair chez le client et le serveur. Si on ne prend pas de précautions suffisantes, un craqueur peut donc les lire et se faire alors passer pour l'une des deux parties (section 4.5).

— En lui-même, OAuth ne garantit que l'authentification, pas la confidentialité. Si les requêtes ne passent pas en https, elles sont visibles à tout espion (section 4.2).
— L'existence sur le trajet d'un relais/cache HTTP peut compliquer les choses. Par exemple, si la requête à la ressource est faite sans le champ HTTP `Authorization:` (ce qui est permis : section 3.5.1), le relais ne peut pas savoir que la ressource est protégée et risque donc de la distribuer après à d'autres clients. Il faut donc, dans ce cas, utiliser `Cache-control:` pour l'en empêcher (section 4.4).
— Le client s'authentifie (paramètre `oauth_consumer_key` et signature de la requête initiale), certes, mais le client peut être du logiciel distribué avec son code source et, de toute façon, un attaquant déterminé peut trouver les paramètres du client même sans le code source, puisqu'il tourne sur une machine que l'attaquant contrôle. Le serveur ne doit donc pas faire une confiance aveugle que le client est bien celui qu'il prétend être (section 4.6).
— Comme tout système reposant sur la redirection HTTP (le cas a été largement discuté pour OpenID), OAuth peut être sensible au hameçonnage. En effet, un méchant client peut rediriger le navigateur vers un faux serveur. Le propriétaire va alors s'authentifier auprès de ce qu'il croit être le serveur et peut-être donner alors des secrets réutilisables (par exemple un mot de passe). Le propriétaire humain doit donc faire attention à ce qu'il est bien sur le serveur attendu (section 4.7).
— OAuth, dans la phase d'autorisation par le propriétaire, est également vulnérable au "*clickjacking*" (section 4.14).

Un guide complet sur OAuth est disponible en <http://hueniverse.com/oauth/guide/>. Une autre bonne introduction est « "*Gentle introduction to OAuth*" <https://maqentaer.github.io/devopera-static-backup/http/dev.opera.com/articles/view/gentle-introduction-to-oaut index.html> ». 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 pour accéder à Twitter en Python figure dans mon article <https://www.bortzmeyer.org/twitter-oauth.html>. Fin 2012, la version 2 de OAuth est en cours de finalisation à l'IETF, mais de grosses incertitudes demeurent. Le premier RFC de la nouvelle version est le RFC 6749.