OAuth - The Standard That Isn't...

By

APImetrics, WA, USA
and OAuth.io, USA

The idea behind OAuth is simple - a standard way to enable access to protected resources (like a social network), so a user can sign into a web service, and, via a secure exchange of tokens, an app or 3rd party can access that service. All happening without the developer ever seeing the user's username or password. However, in reality, there is a lot of variation between implementations and many traps for the developer.

OAuth 1.0 was released in October 2007, revised in June 2009 (Revision A) and is still used by many services such as Twitter, Flickr and Yahoo. It was later published as RFC 5849. The protocol had some shortcomings though, as client developers found it hard to implement, and it did not allow for the expiry of tokens, or to control the level of access requested. Some implementations have tried to get around these problems, which causes interoperability issues, but OAuth 1.0 is mostly strictly adhered to, so developers of clients can now use one of the many standard library implementations to access an OAuth 1.0 protected resource.

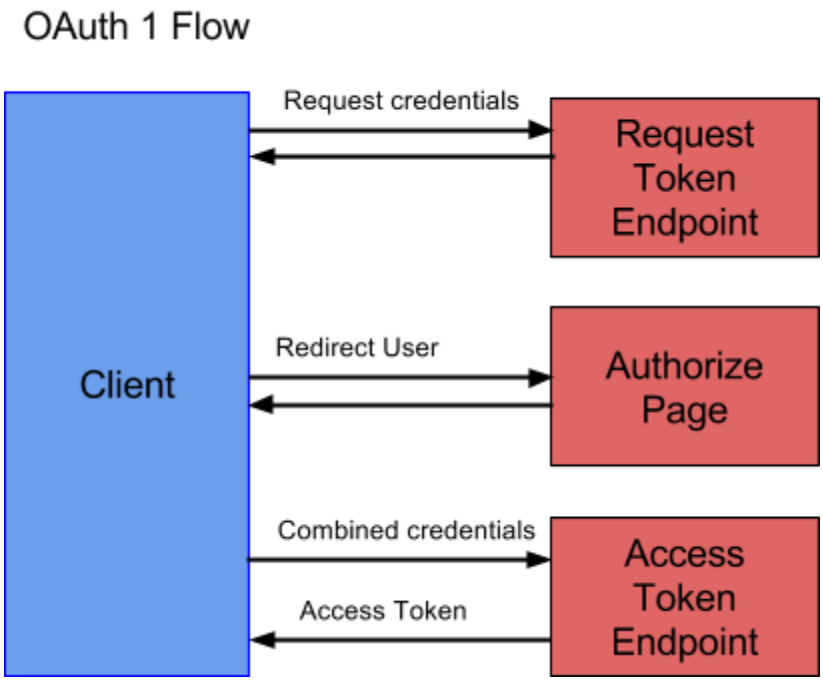
As an attempt to solve the OAuth 1.0's issues, OAuth 2.0 was developed as a non-backward compatible alternative. It was finally published as RFC 6749 in October 2012, but had been through several drafts going back to January 2010, and there were many popular APIs implementing the draft protocol long before it was finalized, most notably Facebook. As OAuth 2.0 is more flexible, the specification itself notes it "is likely to produce a wide range of non-interoperable implementations", a problem we regularly face when configuring tests. The challenge has become while OAuth 2.0 was aimed at being a protocol, like OAuth 1.0 it has ended up being a Framework. Additionally, OAuth 2.0 is also less secure than OAuth 1.0, relying on SSL connections rather than signatures to protect the user's access token, but this does make developing clients much easier.

In configuring tests, we've found 4 major areas of concern for developers, the first 3 are general developer focused problems, the 4th, as we'll show, is another potential problem.

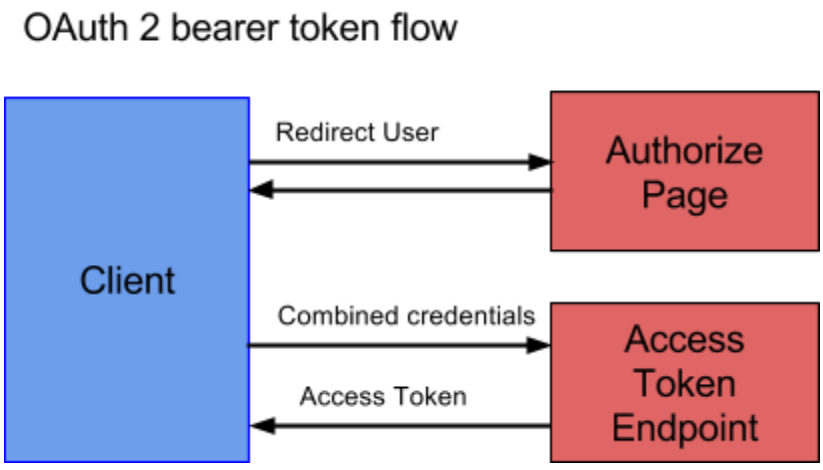
1. Documentation - can you easily find the user flow and the information required to implement the authentication?
2. API Implementation - does the token response look like what you expected?
3. Token Expiry and Renewal - does the token expire, and can we refresh it?
4. Speed - how fast is the process?

Documentation:

For OAuth 1.0 there are 3 distinct and separate calls that need to be made. First, your client calls the OAuth server and asks for temporary credentials. Then, the client opens a webpage dialog using those credentials so the user can sign in and give you access. The user is then redirected back to your client with a temporary token. Finally, your client calls the OAuth server again combining the temporary credentials and token to get the final access token.



For OAuth 2.0, it is simpler as you only need two calls - one to open the webpage dialog, and then another to swap the returned code for an access token.



Just finding the URI's for these services can be a challenge in developer documentation. In

others they hide the actual nuts and bolts of the process inside code examples where it's not clear which service is which. While they are trying to help developers, the net result can be confusing. There can also be naming issues - it can be difficult to work out whether a service is using OAuth 1.0 or 2.0 from the documentation.

API Implementation:

When you call the OAuth server to get the access token, we've found enormous variation in the way the service returns the data. There are services which don't follow the specification, or extend the specification for their own uses. Most return JSON (like Facebook), some XML (Concur), some just URL-encoded text (Google). If you're used to looking for a token in a particular format with one service, this may not help you with another service which returns the same data but formatted and presented very differently. Copying code over from a previous example doesn't really help very much either, as you'll find that almost every service you connect to will need to be handled differently.. As noted earlier, this is more frequent with OAuth 2.0, but non-standard OAuth 1.0 implementations do exist and if anything cause more difficulty.

OAuth 2.0 has the problem that early implementations only supported the access token as a parameter of the HTTP request, whereas the final specification says it should be in the HTTP header. The former made implementing code in javascript easier, for example, but the latter is the more standard way to handle authentication. Some services only support one of these options, and even then, the exact parameter name or header name can vary. For example Facebook uses "access_token" (as given in the RFC6749) while Google and more than 50% of the web use "oauth_token", mostly due to the old OAuth 2 draft 10.

When making the initial request for OAuth 2.0, the scope parameter has a huge variation in format: the separators should be a space " ", but often they use "," ";" or even "|". Moreover its degree of cardinality is unclear: you can have a single option (read only, read and write, ...) like Disqus or Heroku, or multiple (read access for X, write access for X, read access for Y, ...). Google has also a unique way to deal with the scope: the names are directly a URL. There are a lot of unique differences, that can be from adding prefix to scope elements (live.com) or accepting only uppercase characters (Odnoklassniki).

As well as being a problem for developers to call these APIs, the non-standard approaches can add security flaws. RFC 6819 has recently been released covering some of the security issues surrounding OAuth 2.0.

For OAuth 1, there are still some providers supporting only OAuth 1.0 (not OAuth 1.0a), like Dropbox (but also OAuth 2 now) or TripIt which does not return an oauth_verifier and requires an oauth_callback to the authorization url. This exposes these APIs to a CSRF vulnerability, so is a concern for security.

For OAuth 2, some providers are missing the useful "state" parameter, and without this , the API

can be exposed to a CSRF vulnerability (the same way as OAuth 1.0). For those that do not implement it, a common way to deal with a CSRF token is to include it in the redirect uri, but it is not always possible (AngelList).

Token Expiry and Renewal:

Tokens can and do expire and there is wild variation between services in how quickly this happens. There is also huge variation in whether or not you can easily refresh the token. With OAuth 2.0 this should be as simple as calling the OAuth server with a refresh token, which can happen without user interaction, but not all services support this for security issues.

Facebook does not use the refresh token flow as proscribed by the spec. Instead, it adds the "fb_exchange_token" grant type which exchanges a token for a new token with a new expiry. The standard proposes a refresh token method but only few providers implement it (Google, Github).

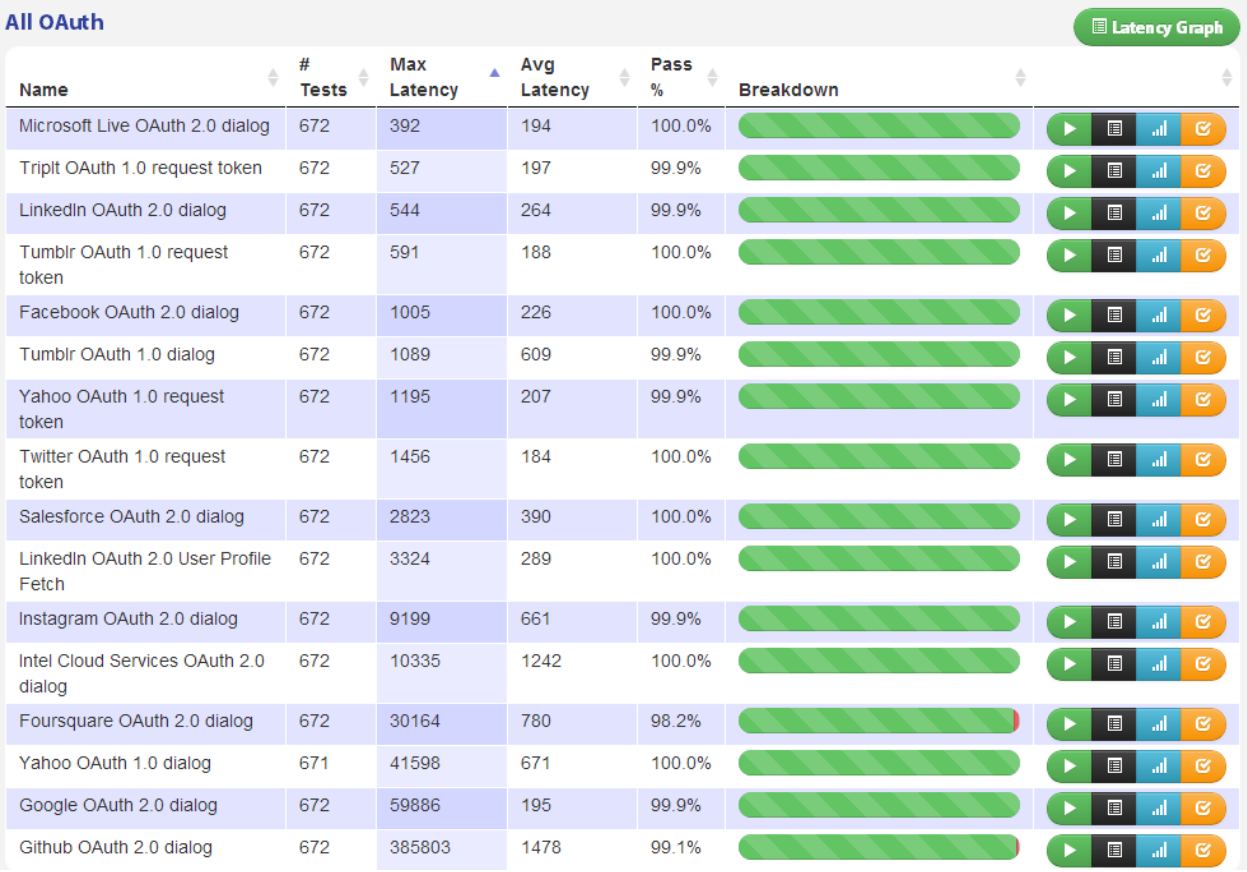
Some providers let the developer control the token expiry, either to set it longer or to request an access token without expiry. But they rarely have the same name or method to define it: Google adds a field "access_type" to the authorization url that can be "online" or "offline", but others add this option in the scope, where it can be "no_expiry" (StackExchange), "non-expiring" (Soundcloud), "ageless" (Meetup.com), "wl.offline_access" (Live.com), and so on.

If the services don't allow the automatic renewal, you must ask the user to sign in again to get a new valid token. During development when you're conducting individual tests of APIs this can be frustrating as we've seen timeouts as short as 2 hours. We've enabled numerous exception handling services in APImetrics to handle this but it's something that can and will affect you as a developer.

Speed:

As described earlier, the OAuth process requires up to 3 separate API calls. Thes table shows some of the variation in common OAuth servers when tested in a standardized way, from fastest to slowest.

All data has been provided by the APImetrics tool's public community tests which are available to developers free of charge.



All OAuth

Latency Graph

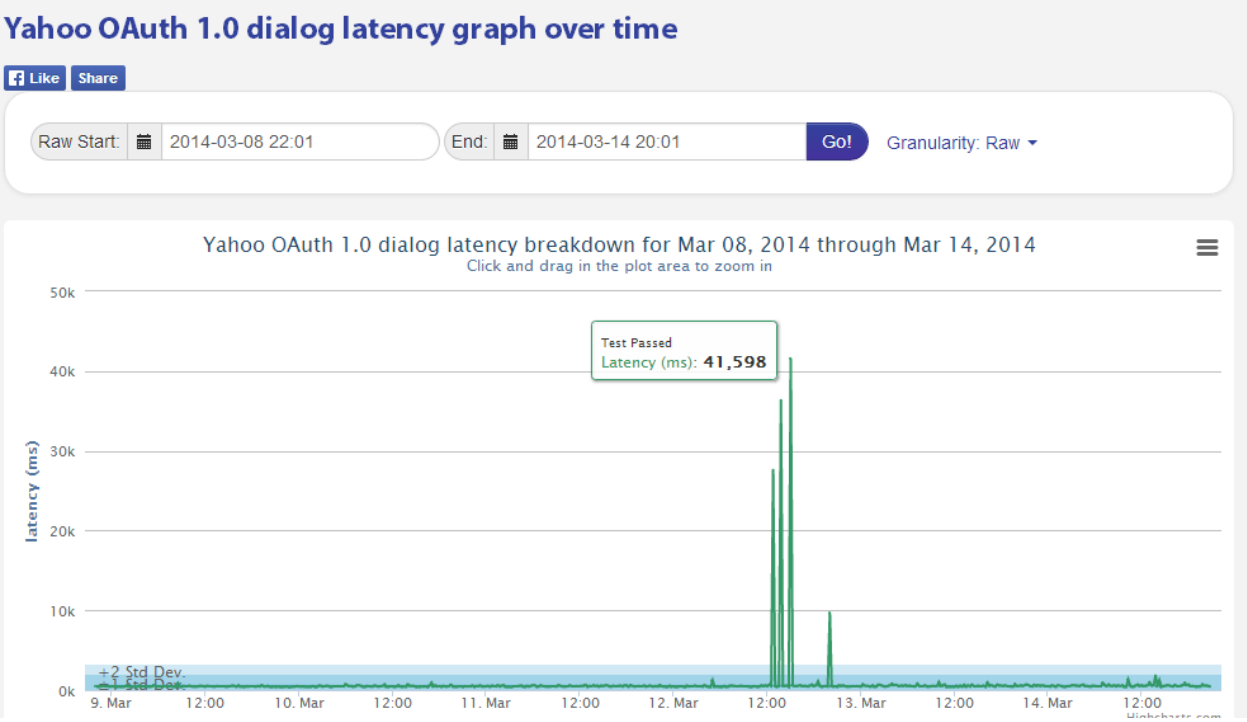
Name	# Tests	Max Latency	Avg Latency	Pass %	Breakdown
Microsoft Live OAuth 2.0 dialog	672	392	194	100.0%	
TripIt OAuth 1.0 request token	672	527	197	99.9%	
LinkedIn OAuth 2.0 dialog	672	544	264	99.9%	
Tumblr OAuth 1.0 request token	672	591	188	100.0%	
Facebook OAuth 2.0 dialog	672	1005	226	100.0%	
Tumblr OAuth 1.0 dialog	672	1089	609	99.9%	
Yahoo OAuth 1.0 request token	672	1195	207	99.9%	
Twitter OAuth 1.0 request token	672	1456	184	100.0%	
Salesforce OAuth 2.0 dialog	672	2823	390	100.0%	
LinkedIn OAuth 2.0 User Profile Fetch	672	3324	289	100.0%	
Instagram OAuth 2.0 dialog	672	9199	661	99.9%	
Intel Cloud Services OAuth 2.0 dialog	672	10335	1242	100.0%	
Foursquare OAuth 2.0 dialog	672	30164	780	98.2%	
Yahoo OAuth 1.0 dialog	671	41598	671	100.0%	
Google OAuth 2.0 dialog	672	59886	195	99.9%	
Github OAuth 2.0 dialog	672	385803	1478	99.1%	

These tests were all conducted over the same period of a week (ending March 14 2014) on the same test servers sharing a common internet connection. Tests were conducted concurrently.

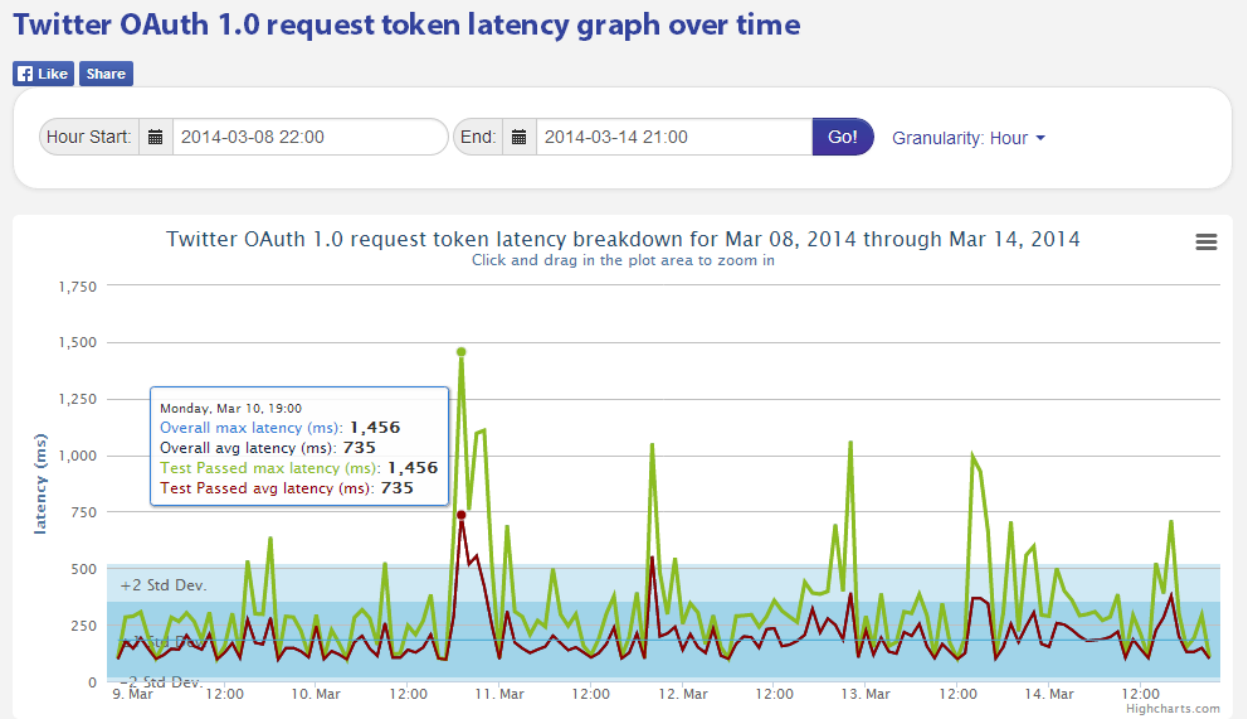
First thing to note is the variation in the Average Latency, from fastest to slowest there is over a second in time difference, with Github and Intel's services running 4-5 times slower than other OAuth services.

Secondly, the max latencies can have an impact on timeouts and overall reliability for users. For normal use, our tool will time out a connection after 60 seconds, but for other services where redirects are in-use, the tool will keep trying. In the case of Github, this took almost 6 minutes involving 10 redirects before we ended the test.

More interesting though is to look at services with 100% pass rates and the variation in the timescales. Looking, for example, at Yahoo OAuth dialog, the service had a 100% pass rate in providing users with the login page, but during one test took over 45 seconds..



Although with the dialog pages, the problems may also lie in the web servers upon which the services depend. If we look instead at an OAuth 1.0 Get Token call we can also see the high degree of variability in the speed of the OAuth servers themselves.



Twitter's OAuth Get Token service can take more than 7 times the average speed, or more than a second to return a token, when considering that this is on top of any delays in serving the dialog page for the user to sign into, this can represent significant time and opportunity for a user to encounter timeout issues in an application, or simply to get bored of the process and use something else.

It is estimated that a 1 second delay in the response of a mobile application can have a significant impact on whether users continue to use, or convert to premium users. Therefore, when picking 3rd party services for your critical applications, the speed of response and handling slow response is essential.

Conclusions:

There's huge variation in OAuth, not just in terms of implementation and documentation, but also in the week to week performance of the services. It's essential to understand this when designing your applications.

When developing, allow additional time to add new services requiring OAuth, and consider using services like those from OAuth.io to handle the challenges of the variable documentation for you.

Finally, plan and design your User Interface and User Experience around the potential for multi-second delays when signing in and have strategies for ensuring users remain engaged during any 3rd party authentication you depend on.

The Authors

About APImetrics

APImetrics is a Seattle based company focused on test and performance of APIs. With almost every app, service and technology depending on APIs, it's more critical than ever to understand what those APIs do and how they'll impact your customers and users. APImetrics products allow developers, enterprises and API providers to easily understand how their APIs work and perform.

Contact

David O'Neill
Phone: 206 972 1140
Email: david@apimetrics.io

About OAuth.io

OAuth.io's service allows developers to integrate OAuth in minutes instead of hours or days. OAuth.io removes the burden of dealing with the 30+ different implementations of this key protocol. OAuth.io supports 100+ hundred providers, including Twitter or Facebook. 4000+ of apps rely on OAuth.io to deal with authorizations. OAuth.io's core engine is open sourced.

Contact

Mehdi Medjaoui
Email: mehdi@oauth.io