An Introduction to OAuth 2

Aaron Parecki • @aaronpk
O'Reilly Webcast • July 2012
A Brief History
Before OAuth
aka the Dark Ages

If a third party wanted access to an account, you’d give them your password.
Several Problems and Limitations

- Apps store the user’s password
- Apps get complete access to a user’s account
- Users can’t revoke access to an app except by changing their password
- Compromised apps expose the user’s password
Before OAuth 1.0

- Services recognized the problems with password authentication
- Many services implemented things similar to OAuth 1.0
- Each implementation was slightly different, certainly not compatible with each other
Before OAuth 1.0

- Flickr: “FlickrAuth” frobs and tokens
- Google: “AuthSub”
- Facebook: requests signed with MD5 hashes
- Yahoo: BBAuth (“Browser-Based Auth”)
“We want something like Flickr Auth / Google AuthSub / Yahoo! BBAuth, but published as an open standard, with common server and client libraries.”

*Blaine Cook, April 5th, 2007*
Click below to change your profile picture!

This will automatically update your Twitter profile picture! We will not post any tweets without your explicit confirmation.

We will not post anything to your Facebook wall without your explicit confirmation.
Authorize Put an Apple On It to use your account?

This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Authorize app  No, thanks

This application will not be able to:
- Access your direct messages.
- See your Twitter password.

Put an Apple On It
putanappleonit.com
Update your Twitter profile image to show your respect to Steve Jobs
← Cancel, and return to app

You can revoke access to any application at any time from the Applications tab of your Settings page.
By authorizing an application you continue to operate under Twitter’s Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.
Your Twitter profile picture was updated!

RIP Steve Jobs. Show your love by changing your profile picture: http://putanappleonit.com

Tweet This!
OAuth 1.0 Signatures

The signature base string is often the most difficult part of OAuth for newcomers to construct. The signature base string is composed of the HTTP method being used, followed by an ampersand ("&") and then the URL-encoded base URL being accessed, complete with path (but not query parameters), followed by an ampersand ("&"). Then, you take all query parameters and POST body parameters (when the POST body is of the URL-encoded type, otherwise the POST body is ignored), including the OAuth parameters necessary for negotiation with the request at hand, and sort them in **lexicographical order** by first parameter name and then parameter value (for duplicates), while ensuring that both the key and the value for each parameter are URL encoded in isolation. Instead of using the equals ("=") sign to mark the key/value relationship, you use the URL-encoded form of "%3D". Each parameter is then joined by the URL-escaped ampersand sign, "%26".

---

oauth_nonce="QP70eNmVz8jvdPevU3oJD2AfF7R7odC2XJcn4XlZJqk", oauth_callback="http%3A%2F%2Flocalhost%3A3005%2Fthe_dance%2Fprocess_callback%3Fservice_provider_id%3D11", oauth_signature_method="HMAC-SHA1", oauth_timestamp="1272323042", oauth_consumer_key="GDdmIQH6jhtmLUypg82g", oauth_signature="8wUi7m5HFQy76nowoCThusfgB%2BQ%3D", oauth_version="1.0"
OAuth 2: signatures replaced by https
Some Current Implementers

An application would like to connect to your account

The application Science Notes by Science Hack Day would like to connect to your Geoloqi account.

Science Notes wants to:
- see my exact last location
- leave me GeoNotes
- subscribe me to layers

Allow Science Notes access?

Deny  Allow

Windows Live

Windows Live for Developers

Announcing Support for OAuth2

by Dare Obasanjo

Members of the Windows Live Team are happy to announce support for OAuth2. Over the past few weeks, thought leaders in the internet identity space have been working to define OAuth2. OAuth is a protocol by which clients can request credentials and other information from OAuth servers. This means that site owners can simply configure the Windows Live Identity Platform to support OAuth2 and developers can request permission to access protected resources.

Request for Permission

My Great Website is requesting permission to do the following:

Access my basic information
Includes name, profile picture, gender, networks, user ID, list of friends, and any other information I've shared with everyone.

Send me email
My Great Website may email me directly at dmp@fb.com

Access posts in my News Feed

Report App

allow  Don't allow

Sign in with Geoloqi

Sign in with Google

Sign in with Foursquare

Connect with Gowalla
# The OAuth 2 Spec

http://oauth.net/2/

## Table of Contents

1. Introduction ................................................. 4
   1.1. Notational Conventions ................................. 5
   1.2. Terminology ........................................... 5
   1.3. Overview ............................................... 7
   1.4. Client Profiles ........................................ 10
      1.4.1. Web Server ......................................... 10
      1.4.2. User-Agent ......................................... 11
      1.4.3. Native Application ................................ 13
      1.4.4. Autonomous ......................................... 14
2. Client Credentials ........................................... 14
   2.1. Client Password Credentials ......................... 15
3. Obtaining End-User Authorization ............................ 16
   3.1. Authorization Response ............................... 18
   3.2. Error Response ........................................ 20
      3.2.1. Error Codes ....................................... 21
4. Obtaining an Access Token .................................... 21
   4.1. Access Grant Types ..................................... 23
      4.1.1. Authorization Code ................................. 23
      4.1.2. Resource Owner Password Credentials .......... 24
      4.1.3. Assertion .......................................... 24
      4.1.4. Refresh Token ..................................... 25
   4.2. Access Token Response .................................. 26
   4.3. Error Response ......................................... 27
      4.3.1. Error Codes ....................................... 28
5. Accessing a Protected Resource ............................... 29
   5.1. Authenticated Requests ................................ 29
      5.1.1. The Authorization Request Header Field .......... 30
      5.1.2. URI Query Parameter ................................ 30
      5.1.3. Form-Encoded Body Parameter ..................... 31
   5.2. The WWW-Authenticate Response Header Field .......... 32
      5.2.1. Error Codes ....................................... 33
6. Extensibility .................................................. 34
OAuth 2?!  
There are 29 versions!

| Versions: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
## Currently Implemented Drafts

<table>
<thead>
<tr>
<th>Provider</th>
<th>Draft</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foursquare</td>
<td>-10</td>
<td><a href="http://aaron.pk/2YS">http://aaron.pk/2YS</a></td>
</tr>
<tr>
<td>Google</td>
<td>-10</td>
<td><a href="http://code.google.com/apis/accounts/docs/OAuth2.html">http://code.google.com/apis/accounts/docs/OAuth2.html</a></td>
</tr>
<tr>
<td>Facebook</td>
<td>-10 (ish)</td>
<td><a href="https://developers.facebook.com/docs/authentication/oauth2_updates/">https://developers.facebook.com/docs/authentication/oauth2_updates/</a></td>
</tr>
<tr>
<td>Windows Live</td>
<td>-10</td>
<td><a href="http://aaron.pk/2YV">http://aaron.pk/2YV</a></td>
</tr>
<tr>
<td>Salesforce</td>
<td>-10</td>
<td><a href="http://aaron.pk/2YW">http://aaron.pk/2YW</a></td>
</tr>
<tr>
<td>Github</td>
<td>-07</td>
<td><a href="http://develop.github.com/p/oauth.html">http://develop.github.com/p/oauth.html</a></td>
</tr>
<tr>
<td>Geoloqi</td>
<td>-10</td>
<td><a href="http://developers.geoloqi.com/api">http://developers.geoloqi.com/api</a></td>
</tr>
</tbody>
</table>
So how does it work?
Definitions

- **Resource Owner**: The User
- **Resource Server**: The API
- **Authorization Server**: Often the same as the API server
- **Client**: The Third-Party Application
Use Cases

- Web-server apps
- Browser-based apps
- Username/password access
- Application access
- Mobile apps
Facebook’s OAuth Flow

User's Browser ➔ Your App ➔ Facebook

1. User clicks a call-to-action to login

2. GET Your app’s frontpage

3. GET OAuth Dialog

4. 302 Redirect including Access Token in URL

5. GET /me?access_token

6. API Response, render user data in page

7. GET /ajax_api.php?

8. Render user data in page

9. GET /me?access_token=...

10. API Response

Source: https://developers.facebook.com/docs/authentication/
Web Server Apps

Authorization Code Grant
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=code&client_id=YOUR_CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=code&client_id=YOUR_CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=code&client_id=YOUR_CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=code&client_id=YOUR_CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=code&client_id=YOUR_CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
User visits the authorization page

https://facebook.com/dialog/oauth?
response_type=code&client_id=28653682475872
&redirect_uri=everydaycity.com&scope=email
On success, user is redirected back to your site with auth code

https://example.com/auth?code=AUTH_CODE_HERE

On error, user is redirected back to your site with error code

https://example.com/auth?error=access_denied
Server exchanges auth code for an access token

Your server makes the following request

POST https://graph.facebook.com/oauth/access_token

Post Body:
grant_type=authorization_code
&code=CODE_FROM_QUERY_STRING
&redirect_uri=REDIRECT_URI
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET
Server exchanges auth code for an access token

Your server gets a response like the following

```json
{
    "access_token": "RsT5OjbzRn430zqMLgV3Ia",
    "token_type": "bearer",
    "expires_in": 3600,
    "refresh_token": "e1qoXg7Ik2RRua481XIV"
}
```

or if there was an error

```json
{
    "error": "invalid_request"
}
```
Browser-Based Apps

Implicit Grant
Create a “Log In” link

Link to:

https://facebook.com/dialog/oauth?
response_type=token&client_id=CLIENT_ID
&redirect_uri=REDIRECT_URI&scope=email
User visits the authorization page

https://facebook.com/dialog/oauth?
response_type=token&client_id=2865368247587&
redirect_uri=everydaycity.com&scope=email
On success, user is redirected back to your site with the access token in the fragment

https://example.com/auth#token=ACCESS_TOKEN

On error, user is redirected back to your site with error code

https://example.com/auth#error=access_denied
Browser-Based Apps

- Use the “Implicit” grant type
- No server-side code needed
- Client secret not used
- Browser makes API requests directly

aaron.pk/oauth2
Username/Password

Password Grant
Password Grant

Password grant is only appropriate for trusted clients, most likely first-party apps only.

If you build your own website as a client of your API, then this is a great way to handle logging in.
Password Grant Type

Only appropriate for your service’s website or your service’s mobile apps.
Password Grant

POST https://api.example.com/oauth/token

Post Body:
grant_type=password
&username=USERNAME
&password=PASSWORD
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET

Response:

{
  "access_token": "RsT5OjbzRn430zqMLgV3Ia",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "e1qoXg7Ik2RRua481XIV"
}
Application Access

Client Credentials Grant
Client Credentials Grant

POST https://api.example.com/1/oauth/token

Post Body:
grant_type=client_credentials
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET

Response:

```json
{
  "access_token": "RsT5OjbzRn430zqMLgV3Ia",
  "token_type": "bearer",
  "expires_in": 3600,
  "refresh_token": "e1qoXg7Ik2RRua481XIV"
}
```
Mobile Apps
Implicit Grant
Everyday City

Automatically post your current city to Facebook

Login with Facebook

Photo credit: flickr-kennymatic
Everyday City

This app may post on your behalf, including cities you arrived in and more.

About this app

You are logging into this app as Aaron Parecki.

By proceeding, you also agree to Everyday City's Terms of Service and Privacy Policy.
Redirect back to your app

Facebook app redirects back to your app using a custom URI scheme.

Access token is included in the redirect, just like browser-based apps.

fb2865://authorize/#access_token=BAAEEmo2nocQBAFFOeRTd
Everyday City runs in the background and automatically posts your current city to Facebook!

Even though you will see the location services icon in your phone's menu, the app is monitoring your location in a battery-safe way.

Automatic Updating

Off  On

Powered by Geoloqi

Logout

History  Current City
Mobile Apps

- Use the “Implicit” grant type
- No server-side code needed
- Client secret not used
- Mobile app makes API requests directly
Accessing Resources

So you have an access token. Now what?
Use the access token to make requests

Now you can make requests using the access token:

```
GET https://api.example.com/me
Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia
```

Access token can be in an HTTP header or a query string parameter:

```
https://api.example.com/me?
access_token=RsT5OjbzRn430zqMLgV3Ia
```
Eventually the access token will expire

When you make a request with an expired token, you will get this response

```
{
  "error": "expired_token"
}
```

Now you need to get a new access token!
Get a new access token using a refresh token

Your server makes the following request

POST https://api.example.com/oauth/token

grant_type=refresh_token
&refresh_token=e1qoXg7Ik2RRua48lXIV
&client_id=YOUR_CLIENT_ID
&client_secret=YOUR_CLIENT_SECRET

Your server gets a similar response as the original call to oauth/token with new tokens.

```
{
  "access_token":"RsT5OjbzRn430zqMLgV3Ia",
  "expires_in":3600,
  "refresh_token":"e1qoXg7Ik2RRua48lXIV"
}
```
Moving access into separate specs

Bearer tokens vs MAC authentication
Bearer Tokens

GET /1/profile HTTP/1.1
Host: api.example.com
Authorization: Bearer B2mpLsHWhuVFw3YeLFW3f2

Bearer tokens are a cryptography-free way to access protected resources.

Relies on the security present in the HTTPS connection, since the request itself is not signed.
Security Recommendations for Clients Using Bearer Tokens

- Safeguard bearer tokens
- Validate SSL certificates
- Always use https
- Don’t store bearer tokens in plaintext cookies
- Issue short-lived bearer tokens
- Don’t pass bearer tokens in page URLs
MAC Tokens

GET /1/profile HTTP/1.1
Host: api.example.com
Authorization: MAC id="jd93dh9dh39D",
nonce="273156:di3hvd8",
bodyhash="k9kbtCIyI3/FEfpS/oIDjk6k=",
mac="W7bdMZbv9UWOTadASIQHagZyirA=

MAC tokens provide a way to make authenticated requests with cryptographic verification of the request.

Similar to the original OAuth 1.0 method of using signatures.
OAuth 2 Clients

Client libraries should handle refreshing the token automatically behind the scenes.
Scope

Limiting access to resources
Limiting Access to Third Parties

Authorize Awesome App to use your account?

This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Authorize app  No, thanks

This application will not be able to:
- Access your direct messages.
- See your Twitter password.

Awesome App
example.com
Update your Twitter profile image to show your respect to Steve Jobs
← Cancel, and return to app

You can revoke access to any application at any time from the Applications tab of your Settings page. By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.
Limiting Access to Third Parties

Authorize Awesome App to use your account

This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Authorize app  No, thanks

This application will not be able to:
- Access your direct messages.
- See your Twitter password.

Awesome App
dinosaur.png
Update your Twitter profile image to show your respect to Steve Jobs
← Cancel, and return to app
Limiting Access to Third Parties

Authorize Awesome App to use your account?

This application will be able to:
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.

Authorize app  No thanks

Awesome App
example.com
Update your Twitter profile image to show your respect to Steve Jobs
← Cancel, and return to app

This application will not be able to:
- Access your direct messages.
- See your Twitter password.

You can revoke access to any application at any time from the Applications tab of your Settings page. By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.
OAuth 2 scope

- Created to limit access to the third party.
- The scope of the access request expressed as a list of space-delimited strings.
  - In practice, many people use comma-separators instead.
- The spec does not define any values, it’s left up to the implementor.
- If the value contains multiple strings, their order does not matter, and each string adds an additional access range to the requested scope.
OAuth 2 scope on Facebook

## OAuth 2 scope on Facebook

<table>
<thead>
<tr>
<th>User permission</th>
<th>Friends permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user_about_me</td>
<td>friends_about_me</td>
<td>Provides access to the &quot;About Me&quot; section of the profile in the about property</td>
</tr>
<tr>
<td>user_activities</td>
<td>friends_activities</td>
<td>Provides access to the user's list of activities as the activities connection</td>
</tr>
<tr>
<td>user_birthday</td>
<td>friends_birthday</td>
<td>Provides access to the birthday with year as the birthday property</td>
</tr>
<tr>
<td>user_checkins</td>
<td>friends_checkins</td>
<td>Provides read access to the authorized user's check-ins or a friend's check-ins that the user can see. This permission is superseded by user_status for new applications as of March, 2012.</td>
</tr>
<tr>
<td>user_education_history</td>
<td>friends_education_history</td>
<td>Provides access to education history as the education property</td>
</tr>
<tr>
<td>user_events</td>
<td>friends_events</td>
<td>Provides access to the list of events the user is attending as the events connection</td>
</tr>
<tr>
<td>user_groups</td>
<td>friends_groups</td>
<td>Provides access to the list of groups the user is a member of as the groups connection</td>
</tr>
<tr>
<td>user_hometown</td>
<td>friends_hometown</td>
<td>Provides access to the user's hometown in the hometown property</td>
</tr>
</tbody>
</table>
OAuth 2 scope on Github

https://github.com/login/oauth/authorize?
client_id=...&scope=user,public_repo

user
• Read/write access to profile info only.

public_repo
• Read/write access to public repos and organizations.

repo
• Read/write access to public and private repos and organizations.

delete_repo
• Delete access to adminable repositories.

gist
• write access to gists.
Proposed New UI for Twitter
by Ben Ward

http://blog.benward.me/post/968515729
Implementing an OAuth Server
Implementing an OAuth Server

- Find a server library already written:
  - A short list available here: http://oauth.net/2/

- Read the spec of your chosen draft, in its entirety.
  - These people didn’t write the spec for you to ignore it.
  - Each word is chosen carefully.

- Ultimately, each implementation is somewhat different, since in many cases the spec says SHOULD and leaves the choice up to the implementer.

- Understand the security implications of the implementation choices you make.
Implementing an OAuth Server

- Choose which grant types you want to support
  - Authorization Code – for traditional web apps
  - Implicit – for browser-based apps and mobile apps
  - Password – for your own website or mobile apps
  - Client Credentials – if applications can access resources on their own

- Choose whether to support Bearer tokens, MAC or both

- Define appropriate scopes for your service
OAuth 2 **scope** on your service

- Think about what scopes you might offer
- Don’t over-complicate it for your users
- Read vs write is a good start
Mobile Applications

- External user agents are best
  - Use the service’s primary app for authentication, like Facebook
  - Or open native Safari on iPhone rather than use an embedded browser

- Auth code or implicit grant type
  - In both cases, the client secret should never be used, since it is possible to decompile the app which would reveal the secret
Staying Involved
Join the Mailing List!

- https://www.ietf.org/mailman/listinfo/oauth

- People talk about OAuth

- Keep up to date on changes

- People argue about OAuth

- It’s fun!
An **open protocol** to allow **secure authorization** in a **simple** and **standard** method from web, mobile and desktop applications. Read the OAuth 2 specification »

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service.

For Consumer developers...

If you're building...

- web applications
- desktop applications
- mobile applications
- Javascript or browser-based apps
- webpage widgets

OAuth is a simple way to publish and interact with protected data. It's also a safer and more secure way for people to give you access. We've kept it simple to save you time.

For Service Provider developers...

If you're supporting...

- web applications
- mobile applications
- server-side APIs
- mashups

If you're storing protected data on your users' behalf, they shouldn't be spreading their passwords around the web to get access to it. Use OAuth to give your users access to their data while protecting their account credentials.

Get started...
oauth.net Website

- [http://oauth.net](http://oauth.net)
- Source code available on Github
  - [github.com/aaronpk/oauth.net](https://github.com/aaronpk/oauth.net)
- Please feel free to contribute to the website
- Contribute new lists of libraries, or help update information
- OAuth is community-driven!
The http://oauth.net website. Feel free to send pull requests with updates.

http://oauth.net

cleaned up server/client section, added new links and added a section...
Thanks.

More Info, Slides & Code Samples:

aaron.pk/oauth2

Aaron Parecki
@aaronpk
aaronparecki.com
github.com/aaronpk