OAuth 2.0

What Is OAuth?

- OAuth is not an API or a service: it’s an open standard for authorization and anyone can implement it.
- OAuth is a standard that apps can use to provide client applications with “secure delegated access”. OAuth works over HTTPS and authorizes devices, APIs, servers, and applications with access tokens rather than credentials.
- There are two versions of OAuth: OAuth 1.0a and OAuth 2.0. These specifications are completely different from one another, and cannot be used together: there is no backwards compatibility between them.
- Nowadays, OAuth 2.0 is the most widely used form of OAuth

Facebook Graph API Demo

https://developers.facebook.com/tools/explorer

Why OAuth?

- OAuth was created as a response to the direct authentication pattern
- Basic Authentication is still used as a primitive form of API authentication for server-side applications: instead of sending a username and password to the server with each request, the user sends an API key ID and secret
- Before OAuth, sites would prompt you to enter your username and password directly into a form and they would login to your data (e.g. your Gmail account) as you. This is called the password anti-pattern.

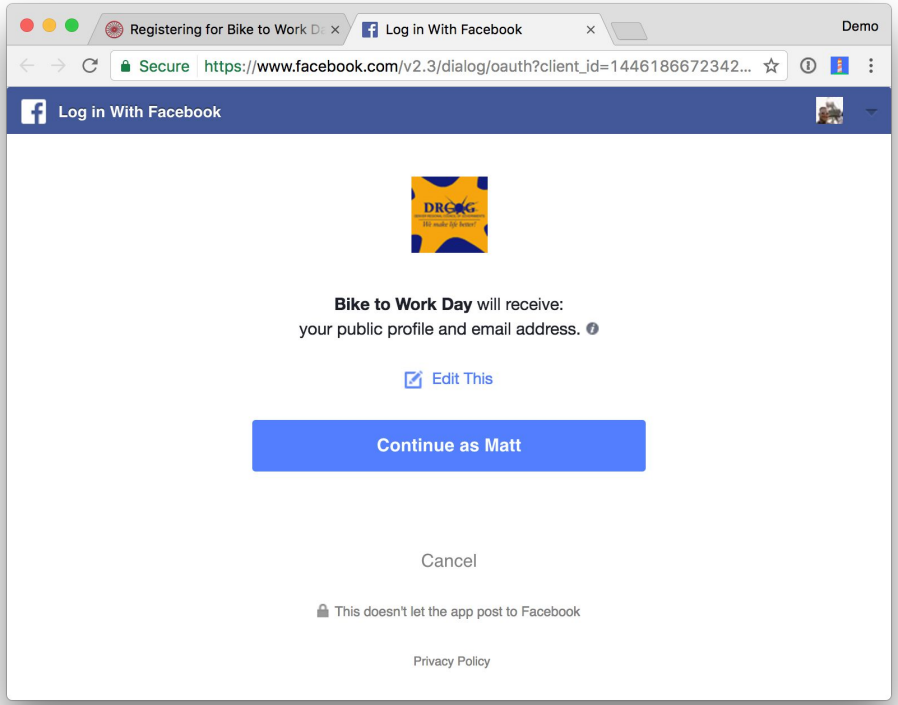
The Password Anti-Pattern

OAuth and APIs

In the old days, you'd enter your username/password directory and the app would login directly as you. This gave rise to the delegated authorization problem.

"How can I allow an app to access my data without necessarily giving it my password?"

This is OAuth



OAuth Definition

OAuth is a delegated authorization framework for REST/APIs. It enables apps to obtain limited access (scopes) to a user's data without giving away a user's password. It decouples authentication from authorization and supports multiple use cases addressing different device capabilities. It supports server-to-server apps, browser-based apps, mobile/native apps, and consoles/TVs.

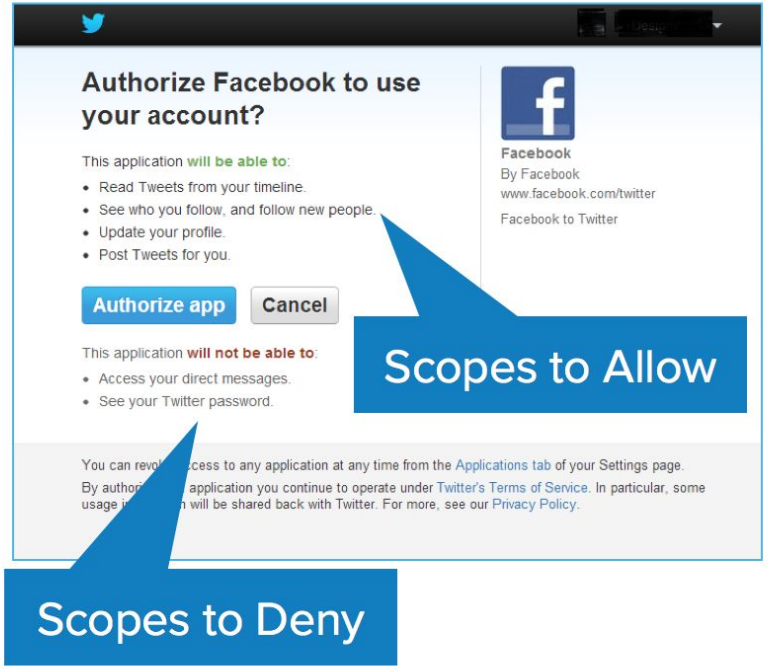
1. App requests authorization from User
2. User authorizes App and delivers proof
3. App presents proof of authorization to server to get a Token
4. Token is restricted to only access what the User authorized for the specific App

OAuth Central Components

- Scopes and Consent
- Actors
- Clients
- Tokens
- Authorization Server
- Flows

OAuth Scopes

Scopes are what you see on the authorization screens when an app requests permissions. They're bundles of permissions asked for by the client when requesting a token. These are coded by the application developer when writing the application.



Scopes decouple authorization policy decisions from enforcement. This is the first key aspect of OAuth. The permissions are front and center. They're not hidden behind the app layer that you have to reverse engineer. They're often listed in the API docs: here are the scopes that this app requires.

OAuth Scopes

- One thing to watch for when you consent is that the app can do stuff on your behalf - e.g. LinkedIn spamming everyone in your network
- OAuth is an internet-scale solution because it’s per application. You often have the ability to log in to a dashboard to see what applications you’ve given access to and to revoke consent.

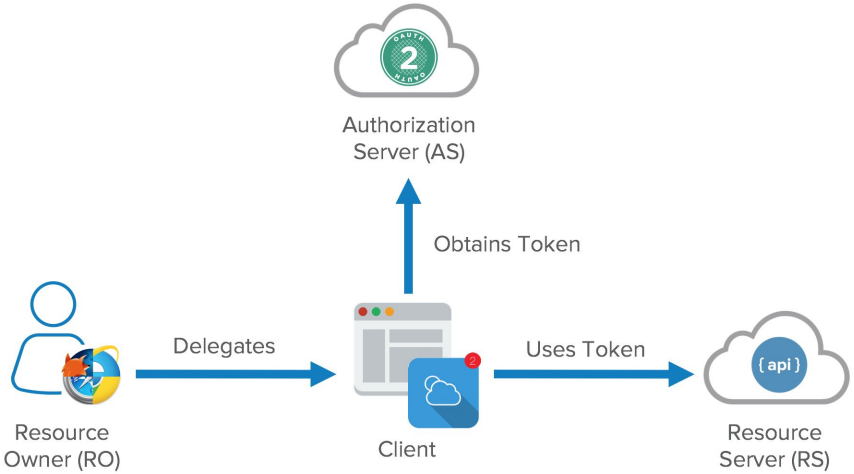
OAuth Actors

The actors in OAuth flows are as follows:

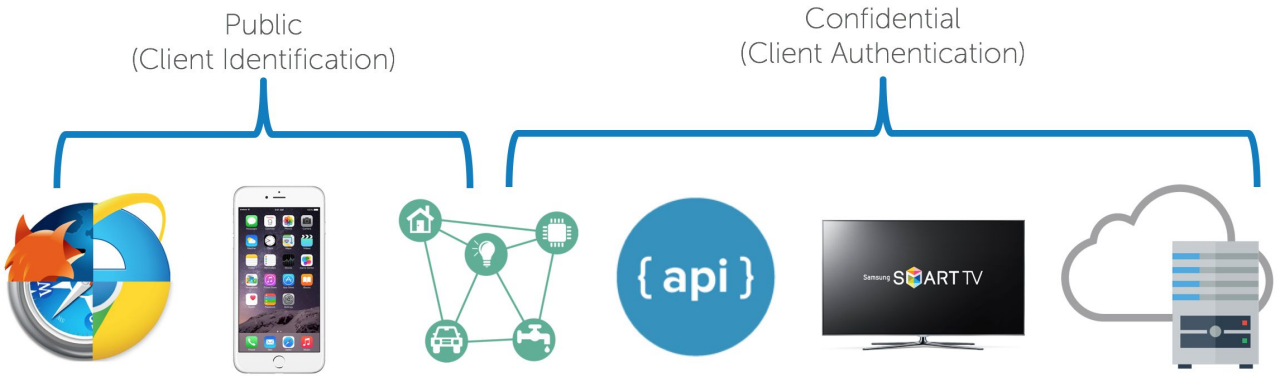
- **Resource Owner**: owns the data in the resource server. For example, I'm the Resource Owner of my Facebook profile.
- **Resource Server**: The API which stores data the application wants to access
- **Client**: the application that wants to access your data
- **Authorization Server**: The main engine of OAuth

OAuth Actors

The resource owner is a role that can change with different credentials. It can be an end user, but it can also be a company.



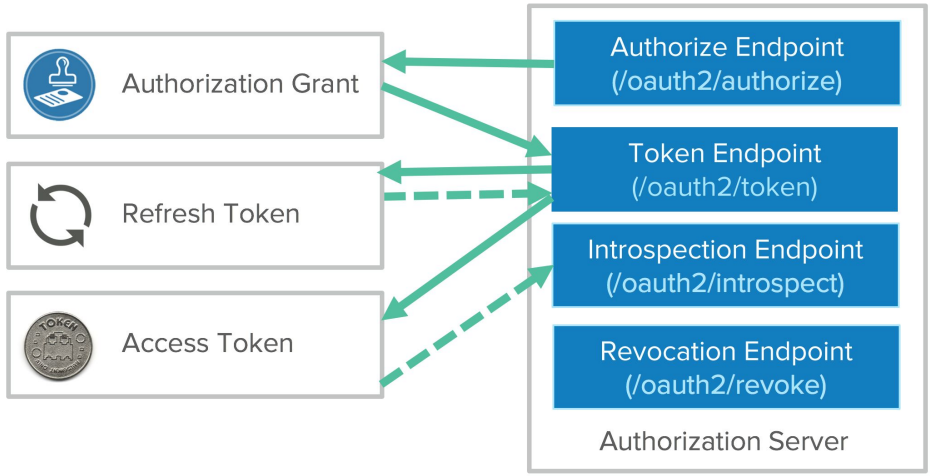
OAuth Clients



Clients can be public and confidential. There is a significant distinction between the two in OAuth nomenclature. Confidential clients can be trusted to store a secret. They're not running on a desktop or distributed through an app store. People can't reverse engineer them and get the secret key. They're running in a protected area where end users can't access them.

OAuth Tokens

Access tokens are the token the client uses to access the Resource Server (API). They're meant to be short-lived. Think of them in hours and minutes, not days and month. You don't need a confidential client to get an access token. You can get access tokens with public clients. They're designed to optimize for internet scale problems. Because these tokens can be short lived and scale out, they can't be revoked, you just have to wait for them to time out.



Refresh Tokens

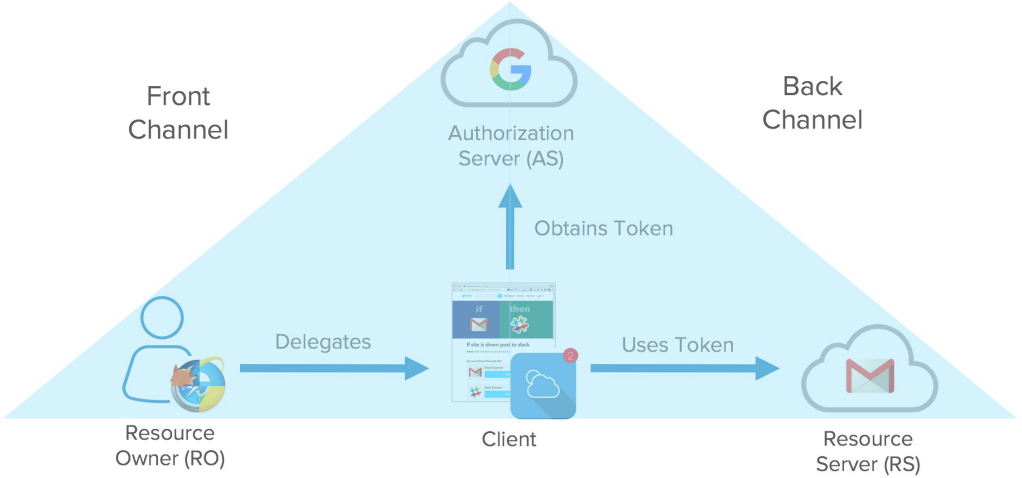
This is much longer-lived; days, months, years. This can be used to get new tokens. To get a refresh token, applications typically require confidential clients with authentication.

Refresh tokens can be revoked. When revoking an application’s access in a dashboard, you’re killing its refresh token. This gives you the ability to force the clients to rotate secrets. What you’re doing is you’re using your refresh token to get new access tokens and the access tokens are going over the wire to hit all the API resources. Each time you refresh your access token you get a new cryptographically signed token. Key rotation is built into the system.

The OAuth spec doesn’t define what a token is. It can be in whatever format you want. Usually though, you want these tokens to be JSON Web Tokens

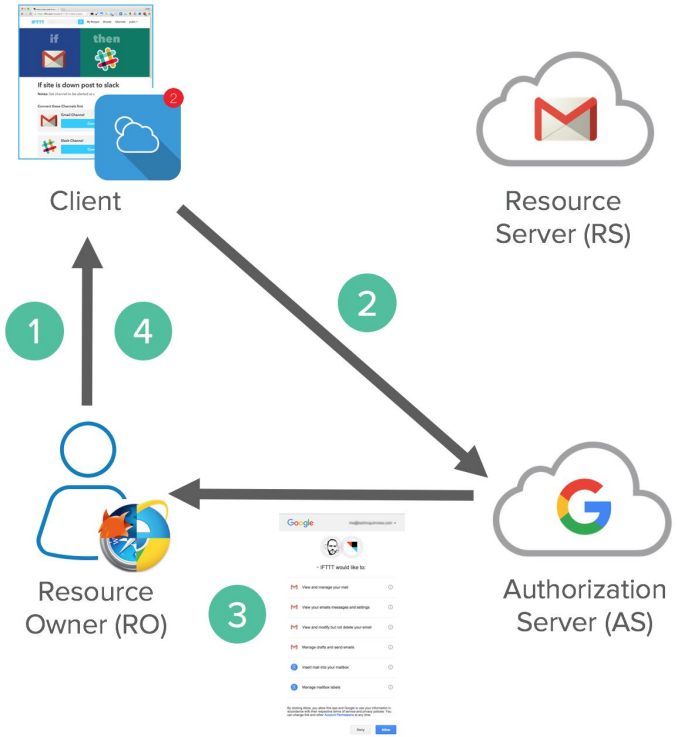
OAuth Flow Channels

There are two different flows: getting the authorization and getting the tokens. Those don't have to happen on the same channel. The front channel is what goes over the browser. The browser redirected the user to the authorization server, the user gave consent. This happens on the user's browser. Once the user takes that authorization grant and hands that to the application, the client application no longer needs to use the browser to complete the OAuth flow to get the tokens.



Front Channel Flow

1. Resource Owner starts flow to delegate access to protected resource
2. Client sends authorization request with desired scopes via browser redirect to the Authorize Endpoint on the Authorization Server
3. Authorization Server returns a consent dialog saying “do you allow this application to have access to these scopes?” Of course, you’ll need to authenticate to the application, so if you’re not authenticated to your Resource Server, it’ll ask you to login. If you already have a cached session cookie, you’ll just see the consent dialog box. View the consent dialog, and agree.
4. The authorization grant is passed back to the application via browser redirect. This all happens on the front channel.



Front Channel Flow Request

GET https://accounts.google.com/o/oauth2/auth?**scope**=gmail.insert gmail.send
&**redirect_uri**=https://app.example.com/oauth2/callback
&**response_type**=code&**client_id**=812741506391 &**state**=af0ifjsldkj

- Scopes are from Gmail's API
- The **redirect_uri** is the URL of the client application that the authorization grant should be returned to. This should match the value from the client registration process. You don't want the authorization being bounced back to a foreign application.
- Response type varies the OAuth flows. Client ID is also from the registration process. State is a security flag

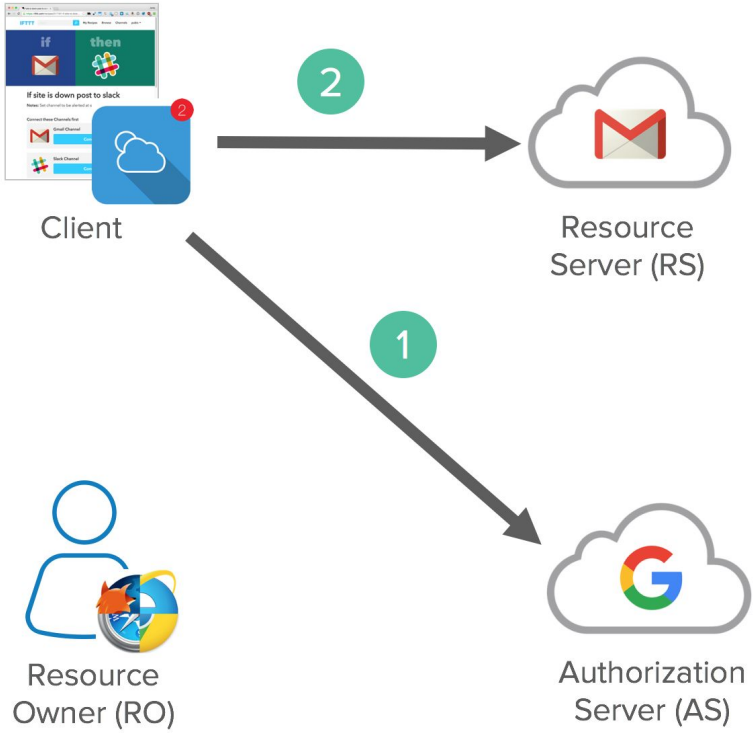
Front Channel Flow Response

HTTP/1.1 302 Found
Location: https://app.example.com/oauth2/callback?
code=MsCeLvIaQm6bTrgtp7&state=af0ifjsldkj

The `code` returned is the authorization grant and `state` is to ensure it's not forged and it's from the same request.

Back Channel Flow

The Client application sends an access token request to the token endpoint on the Authorization Server with confidential client credentials and client id. This process exchanges an Authorization Code Grant for an Access Token and (optionally) a Refresh Token. Client accesses a protected resource with Access Token.



Back Channel Flow Response

POST /oauth2/v3/token HTTP/1.1
Host: www.googleapis.com
Content-Type: application/x-www-form-urlencoded

code=MsCeLvIaQm6bTrgtp7&**client_id**=812741506391&**client_secret**={client_secret}&**redirect_uri**=https://app.example.com/oauth2/callback&**grant_type**=authorization_code

Back Channel Flow Response


```
{
    "access_token": "2YotnFZFEjr1zCsicMWpAA",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA"
}
```

Access Token Usage

```
curl -H "Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA" \
 https://www.googleapis.com/gmail/v1/users/1444587525/messages
```

OAuth Application Registration

Facebook - https://developers.facebook.com/

Google - https://console.developers.google.com/

Revoking Access - https://github.com/settings/applications