**Author:** Aaron Valoroso

**Title:** Understaind OAuthentication2

**Date:** January 12th, 2018

**OS:** Linux / Unix

**Preview:** You can find most of this information over at DigitalOcean’s well written article about OAuth2: <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2> . I will be inserting more topics and ideas of my own into certain places, and will provide exampels of code that

**What is OAuth2:** is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth2 provides authorization flows for web and desktop applications, and mobile devices.

**OAuth Roles:**

OAuth defines four roles:

* Resource Owner
* Client
* Sesource Server
* Authorization Server

**Resource Owner: User –** The resource owner is the user who authorizes an application to access their account. The application’s access to the user’s account is limited to the “scope” of the authorization granted.

**Resource / Authorization Server: API –** The resource server hosts the protected user accounts, and the authorization server verifies the identity of the user then issues access tokens to the application. From an application developer’s point of view, a service’s API fulfills both the resource and authorization server roles. We will refer to both of these roles combined, as the Service or API role.

**Client: Application –** The client is the application that wants the user’s account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.\_



**Application Registration:** Before using OAuth with your application, you must register your application with the service. This is done through a registration form in the “developer” or “API” portion of the service’s website, where you will provide the following information (and probably details about your application):

* Allpication Name
* Application Website
* Redirect URI or Callback URL

The redirect URI is where the service will redirect the user after they authorize (or deny) your application, and therefore the part of your application that will handle authorization codes or access tokens.

**Client ID and Client Secret:** Once your application is registered, the service will issue “client credentials” in the form of a client identifier and a client secret. The Client ID is a publicly exposed string that is used by the service API to identify the application, and is also used to build authorization URLs that are presented to users. The Client Secret is used to authenticate the identity of the application to the service API when the application requests to access a user’s account, and must be kept private between the application and the API.

**Authorization Grant:** In the abstract protocol flow above, the first four steps cover obtaining an authorization grant and access token. The authorization grant type depends on the method used by the application to request authorization, and the grant types supported by the API. OAuth2 defines four grant types, each of which is useful in different cases:

* **Authorization Code:** used with server-side Applications.
* **Implicit:**  used with mobile Apps or Web Applications (applications that run on the user’s device).
* **Resource Owner Password Credentials:** used with trusted Applications, such as those owned by the service itself.
* **Client Credentials:** used with Application API access.

**Grant Type: Authorization Code:** The authorization code grant type is the most commonly used because it is optimized for server-side applications, where source code is not publicly exposed, and Client Secret confidentiality can be maintained. This is a redirection-based flow, which means that the application must be capable of interacting with the user-agent (i.e. the user’s web browser) and receiving API authorization codes that are routed through the user-agent.

Here is the authorization code flow:

