OAuth 2 Simplified

Facebook, Github, Google 과 같이 많은 서비스들이 이미 OAuth 2  서비스를 배치, 구현 중에 있다.

RFC 5849(July31, 2012) 는 OAuth 2.0 에 Spec 이며, 현재 최신 버전은 RFC 6749 이다.

<Resource>
ⅰ) The Third-Party Application: "Client"
Client는 User의 계정에 접근을 시도하는 Application 이다.
이것은 유저로부터 Permission을 얻어야 가능한 일이다.
ⅱ) The API: "Resource Server"
Resource Server는 User의 정보에 접근하는데 사용 되는 API Server이다.
ⅲ) The User: "Resource Owner"
Resource Owner는 User의 계정의 일부기능에 접근하고자하는 이들이다.

<Creating an App>
OAuth 프로세스를 시작하기 전에, 새로운 앱을 서비스에 등록해야 한다. 새로운 앱을 서비스에 등록할 때, 대게 앱 이름, 웹사이트, 로고 등 기본적인 정보를 등록하게 된다. 또 한, 웹 서버, 브라우저기반 혹은 모바일 앱의 리디렉션에 사용되는 Redirect URI를 등록해야 한다.

ⅰ) Redirect URIs
서비스는 여러 공격들을 예방하고자 오직 등록된 URI로만 사용자를 리디렉션 해준다. 모든 HTTP 리디렉션 URI들은 TLS 로 안전해야 하며, 그래서 서비스는 오직 "https"로 시작하는 URI로만 리디렉션 해야 한다. 이것은 권한부여(authorization) 프로세스 중 토큰이 가로막히는(intercepted) 것을 방지한다.

ⅱ) Client ID and Secret
앱을 등록한 이후, Client ID와 Client Secret을 받아야 한다. Client ID는 공개 정보이며 login URL을 구축하거나 Javascript 소스코드 등에 포함된다. Client Secret은 비밀리에 유지되어야 한다. 자바스크립트나 native 앱같이 배치된 앱이 Client Secret을 비밀리에 유지할 수 없다면 사용되지 않는다.

<Authorization>
OAuth 2.0의 첫 단계는 User로 부터 권한을 얻는 것이다. 브라우저 기반 혹은 모바일 앱에서 권한을 얻는 행위는 User에게 서비스가 제공하는 인터페이스를 보여줌으로써 수행된다.

OAuth 2.0은 여러개의 다른 선결형식(grant type)을 제공한다.
• Authorization Code : web server 상의 구동 되는 앱
• Implicit : 브라우저 베이스 혹은 모바일 앱
• Password : Username(아이디)와 Password(비밀번호)로 로그인
• Client credentials : 앱 접근(application access)
자세한 내용은 아래를 확인

<Web Server Apps>
Web server는 OAuth server를 다룰 때 가장 흔한 타입이다. 웹앱은 서버사이드 언어로 작성되며 그 소스코드가 공개적으로 이용될 수 없는(is not available ot the public) server 위에서 작동한다.

ⅰ) Authorization
① 유저에게 보낼 Log in 링크를 만든다.
https://oauth2server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=phtos
② 유저는 authorization 프롬프트를 보게 된다.

유저가 "Allow" 누른다면, 서비스는 auth code 가 있는 사이트로 user를 redirect 한다.
https://oauth2client.com/cb?code=AUTH_CODE_HERE
③ 서버는 Auth code를 Access 토큰으로 변경한다.
POST https://api.oauth2server.com/token
grant_type=authorization_code&
code=AUTH_CODE_HERE&
redirect_uri=REDIRECT_URI&
client_id=CLIENT_ID&
client_secret=CLIENT_SECRET
④ 서버는 access 토큰을 반환한다.
{
"access_token" : "RsT50jbzRn430zqMLgV3Ia"
}
혹은 에러가 발생한다.
{
"error":"invalid_request"
}
Security: 서버는 미리 등록된 redirect URI들만 수용해야 한다.

<Browser-Based Apps>
브라우저베이스 앱은 웹페이지의 모든 소스코드가 로딩된 이후에 브라우저 내에서 작동된다. 모든 소스코드는 브라우저내에 사용가능하기(available) 때문에, Client Secret을 비밀리에 유지할 수 없다. 따라서 Client Secert이 사용되지 않는다.

ⅰ) Authorization
① 유저에게 보낼 Log in 링크를 만든다
https://oauth2server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=phtos
② 유저는 authorization 프롬프트를 보게 된다.
유저가 "Allow" 누른다면, 서비스는 auth code 가 있는 사이트로 user를 redirect 한다.
https://oauth2client.com/cb#token=ACCESS_TOKEN
이 단계로 권한부여가 완료된다. Access 토큰의 파편으로부터 Javascript가 나올 수 있고 그것이 API request를 만들기 시작할 수 있다.
에러가 발생한다면
https://oauth2client.com/cb#error=access_denied

<Mobile Apps>
브라우저베이스 앱과 같이 모바일 앱은 Client Secret 을 비밀리에 유지할 수 없다. 이 때문에 모바일 앱은 Client Secret을 수용하지 않는 Oatuh 흐름을 사용해야 한다.

ⅰ) Authorization
Log in 버튼을 만든뒤, native app 상에서의 서비스 혹은 웹페이지를 통한 서비스를 User 에게 제공한다. iPhone 에서는 custom URI 프로토콜을 등록할 수 있다. "facebook://" 와 같은 프로토콜을 통해 native Facebook 앱을 실행시킬 수 있다. Android 에서는 native app을 실행시킬 수 있는 URL을 등록할 수 있다.

① iPhone
native Facebook 앱을 설치했다면 URL을 통해 전달된다.
fbauth2://authorize?response_type=token&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=email
이 경우에 redirect URL는 "fb00000000://authorize"같이 생겼으며 fb뒤에 붙는 프로토콜은 앱의 Client ID를 따른다. 이것은 앱이 해당 프로토콜에 등록되어야 한다는 것을 의미한다.

② Android or Others
iPhone 이 아니거나, facebook app이 설치되지 않은 iPhone의 경우에는 모바일 브라우저를 통해 authorization이 진행된다.
https://facebook.com/dialog/oauth?response_type=token&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=email
유저는 facebook의 프롬프트 화면을 보게 되며, "Okay"를 누르게 되면 앱으로 URL이 redirect 된다.
해당 URL에서 Access 토큰을 parse 할 수 있다.
fb00000000://authorize#token=ACCESS_TOKEN

<Others>
① Password
OAuth 2.0 username(아이디)와 password(비밀번호)와 Access 토큰을 교환할 수 있는 'password' 선결타입을 제공한다. 이것은 명백히 User의 password를 요구하기 때문에 오직 서비스 자체를 만든 앱에서 사용 된다. 예를 들어, native Twitter 앱은 password 선결타입을 모바일과 데스크탑 앱에서 사용할 수 있다.
'password' 선결타입을 사용하기 위해서 다음과 같은 POST request를 만들어야 한다.
POST https://api.oauth2server.com/token
grant_type=password&
username=USERNAME&
password=PASSWORD&
client_id=CLIENT_ID
서버는 다른 선결형식으로부터 같은형식의 Access 토큰을 반환한다.
② Application access
앱은 그 속의 웹사이트 URL 혹은 앱 아이콘 등의 정보를 갱신하거나  앱의 user에 대한 statistic을 얻기를 원한다. 또한 특정한 유저를 벗어나 Access 토큰을 계좌를 통해 얻을 방법이 필요하다. OAuth는 'client_credentials' 선결타입을 제공함으로써 이를 수행한다.
'client_credential'선결타입을 사용하기 위해서는 다음과 같은 POST request를 만들어야 한다.
POST https://api.oauth2server.com/token
grant_type=client_credentials&
client_id=CLIENT_ID&
client_secret=CLIENT_SECRET
다른 선결타입과 같이 같은 형식의 Access 토큰을 받을 수 있다.

<Making Authenticated Requests>
Access 토큰이 있다면, API를 향한 request를 만들 수 있다.
curl -H "Authorization: Bearer RsT50jbzRn430zqMLgV3Ia"₩
https://api.oauth2server.com/1/me
HTTPS 으로 보내는 것이 intercept와 modify를 보호할 수 있다.

<Differneces from OAuth 1.0>
OAuth 1.0은 Flickr's "FlickrAuth" 그리고 Google's "AuthSub" 와 같은 존재하고있는 소유권이 있는 프로토콜에 많은 기반을 두었다. 그 결과는 실제 구현 경험에 기반한 최적의 대안을 나타낸다. 그러나, 해당 프로토콜을 통해 몇년이 지난 뒤, OAuth 1.0이 제한되거나 혼란이 되는 세가지 중요 영역의 프로토콜을 향상시키고 재고할 필요성을 느끼게 되었다.

ⅰ) Authentication and Signatures
OAuth 1.0에서 개발자의 혼란과 골칫거리였던 것은 Client ID와 Client Secre 서명 요청의 암호화를 요구하는 것이였다. cURL을 쉽게 복사 붙여넣기할 수 있는 능력을 잃는 예제들은 더욱 시작하기 어렵게 만들었다. OAuth 2.0은 이러한 어려움을 인식하고 브라우저와 Client 그리고 API사이의 모든 통신을 HTTPS 를 요구하는 서명으로 교체하였

다.

ii) User Experience and Alternative Authorization Flows

OAuth는 Access 토큰을 얻는 것과 Access 토큰을 사용하여 request를 만드는 두가지 부분을 포함하고 있다. OAuth 1.0에서는 웹 브라우저에 적합하게 작동하였지만 모바일 앱, 혹은 다른 기기(TV 나 게임)에서는 좋은 사용자 경험을 제공하는데 실패하였다. OAuth 2.0에서는 미래의 기기들의 요구사항을 수용할 수 있는 프로토콜을 제공하고 native 앱에 대한 더 나은 사용자경험을 지원한다.

iii) Performance at Scale

큰 프로바이더가 OAuth 1.0을 사용하기 시작하였기에, 프토토콜이 잘 조절되지 않는 것을 확인하였다. 많은 단계들이 단계 관리와 데이터 센터를 건너 동기화 하는 어려움과 공유 저장소를 요구하는 일시적인 자격을 요구한다. OAuth 1.0은 또한 API 서버가 앱의 ID와 Secret에접근하기를 요구했다. 이것은 authorization 서버와 API 서버를 완전히 분리시키는 프로바이더의 구조를 깨트렸다. OAuth 2.0에서는 User의 Authorizaition을 얻는 것과 API를 호출하는 것의 역할을 분리하는 것을 지원한다. 이 확장성을 필요로 하는 큰 프로바이더는 자유롭게 구현할 수 있으며 원한다면 작은 프로바이더들이 같은 서버를 사용할 수 있다.