OAUTH 2.0 OPEN API 인증을 위한 만능 도구상자

시작하며

모바일의 시대, 플랫폼의 시대가 도래하면서 IT기업에게 open API는 가장 중요한 자산으로 자리잡았다. 어떤 형식으로 API를구성하고 어떤 포맷으로 데이터를 주고받을 것인가에 대한 싸움도 치열했는데, SOAP & XML 과 REST & JSON이 경합을 벌인 끝에 REST & JSON의 승리로 끝났다. 이제 새로 생기는 모든 웹 서비스, 모바일 서비스들은 REST & JSON 기반으로 API를 제공하고 있으며 인증 방식으로는 OAuth 2.0을 택하고 있다.

얼마 전 OAuth 의 메인 에디터인 Eran Hammer 가 자신의 블로그에 자신이 OAuth 2.0 editor를 그만둘 것이며 OAuth 2.0 스펙에서 자신의 이름을 지워 달라는 글을 올렸다. OAuth는 SNS와 모바일 세상에서 가장 중요한 인증 방법으로 자리잡았고 OAuth 2.0은 아직 최종 스펙이 나오지는 않았지만 Facebook, Google 등에서 이미 널리 사용되고 있는 스펙이기에 이 소식은 많은 사람들의 관심을 받았다. 이 글에서는 OAuth에 대한 간단한 소개와 함께 OAuth 2.0에 대한 이야기를 해보려고 한다.

OAuth 1.0a API 인증을 위한 만능 칼

OAuth 를 만들고 활성화 시키는데 기여한 가장 유명한 회사는 트위터다. 트위터를 비롯한 웹 개발자 들이 API의 인증(authentication; 이 사용자가 누구인가)과 권한 부여(authorization; 로그인 한 사용자가 무엇을 할 수 있는가?)를 동시에 제공하는 인증 프로토콜을 찾다가, 결국 적당한 것이 없다고 결론 내리고 새로 만든 것이 OAuth 1.0 이다. OAuth 1.0은 2007년 10월에 확정되었으나 나중에 세션 고정 공격(session fixation attack) 보안 결함이 발견되어 2009년 6월에 이 문제가 개선된 OAuth 1.0a가 발표되었다. OAuth 1.0은 보안 결함이 있는 버전이기 때문에 더 이상 사용해서는 안되는 버전이다. OAuth 1.0a는 2010년 4월에 The OAuth 1.0 Protocol“ 이라는 이름으로 RFC5849 문서 번호를 부여 받으며 IETF 표준이 되었다. RFC 표준이름에는 1.0뒤에 a“가 붙지 않지만 많은 사람들이 아직도 OAuth 1.0a라고 부른다. 이 글에서도 명확하게 하기 위해 OAuth 1.0a라고 표현하기로 한다.
이 글의 주요 내용은 OAuth 2.0이긴 하지만, 1.0a에 대해서도 간단히 알고 넘어가려고 한다.

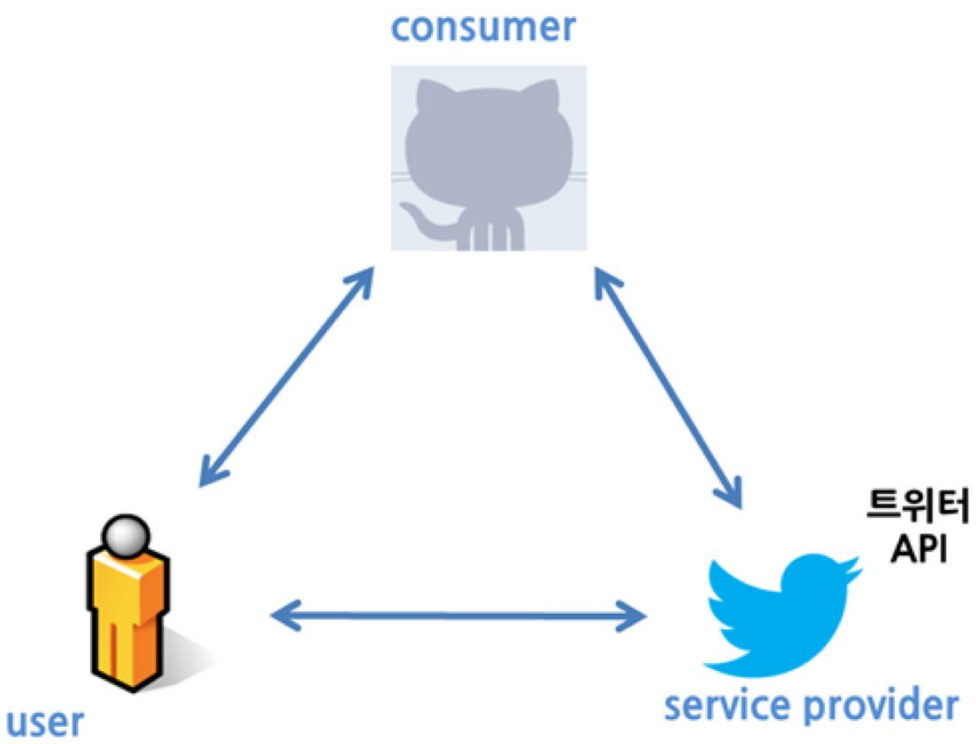
OAuth 1.0a의 장점

OAuth 1.0a가 기존의 다른 인증과 구분되는 특징은 크게 두 가지이다. 첫째, API를 인증함에 있어 써드파티 어플리케이션에게 사용자의 비밀번호를 노출하지 않고 인증 할 수 있다는 점. 둘째, 인증(authentication)과 API권한 부여(authorization)를 동시에 할 수 있다는 점이다. OAuth 1.0이 만들어지는 시점에는 써드파티에게 비밀번호를 노출하지 않고 인증하는 방법으로서 이미 Open ID가 있었다. 하지만 Open ID는 API의 권한 부여기능을 가지고 있지 않았고 인증 방법도 OAuth와는 방향이 많이 달랐다.

OAuth 1.0a 동작방식

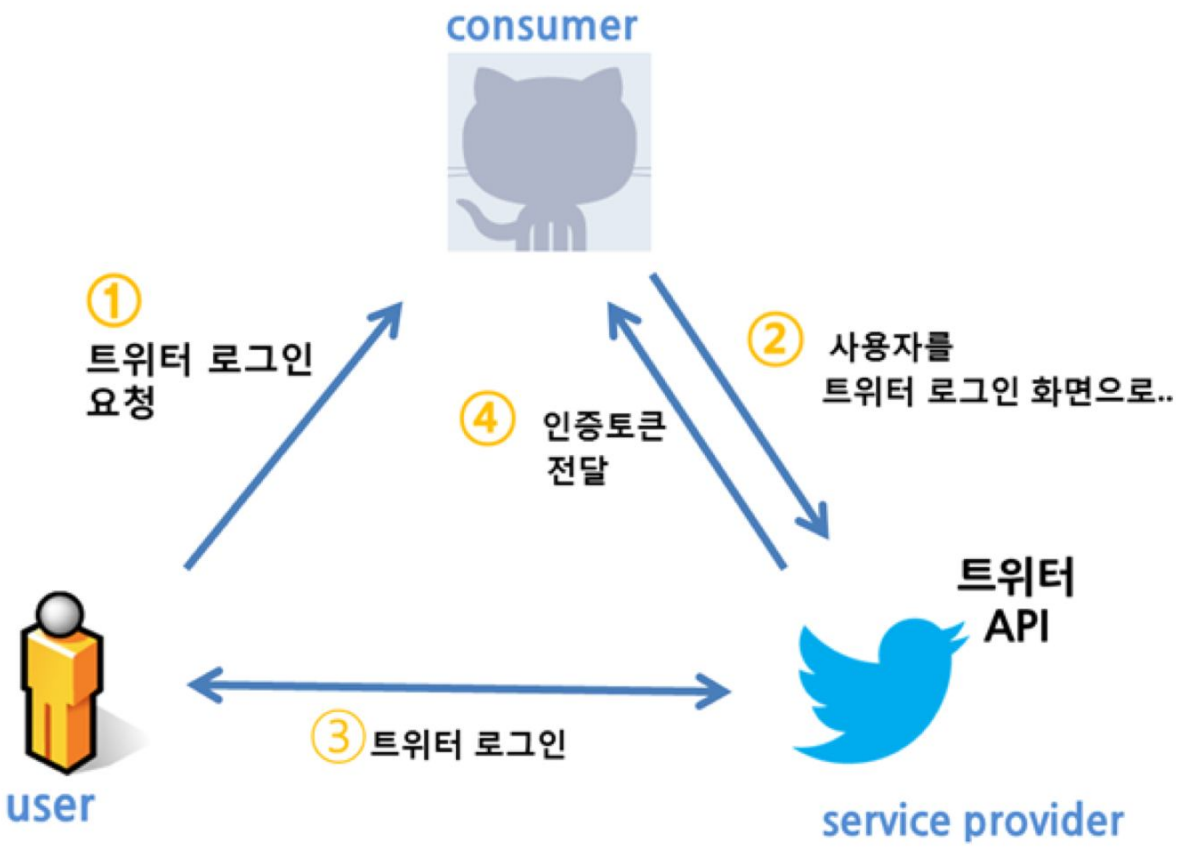
OAuth 1.0a 가 작동하기 위해서는 기본적으로 유저(user), 컨슈머(consumer), 서비스 프로바이더(service provider)가 있어야 한다. OAuth 1.0a 인증을 3-legged OAuth 라고 부르기도 하는데 OAuth는 둘이서 하는 것이 아니라 셋이서 하는 것이라는 말이다. 간단하게는 각각 유저는 트위터 사용자, 컨슈머는 트위터 단말 어플리케이션, 서비스 프로바이더는 트위터 API 서비스 라고 생각하면 쉽다.

< OAuth 1.0a 트라이앵글>

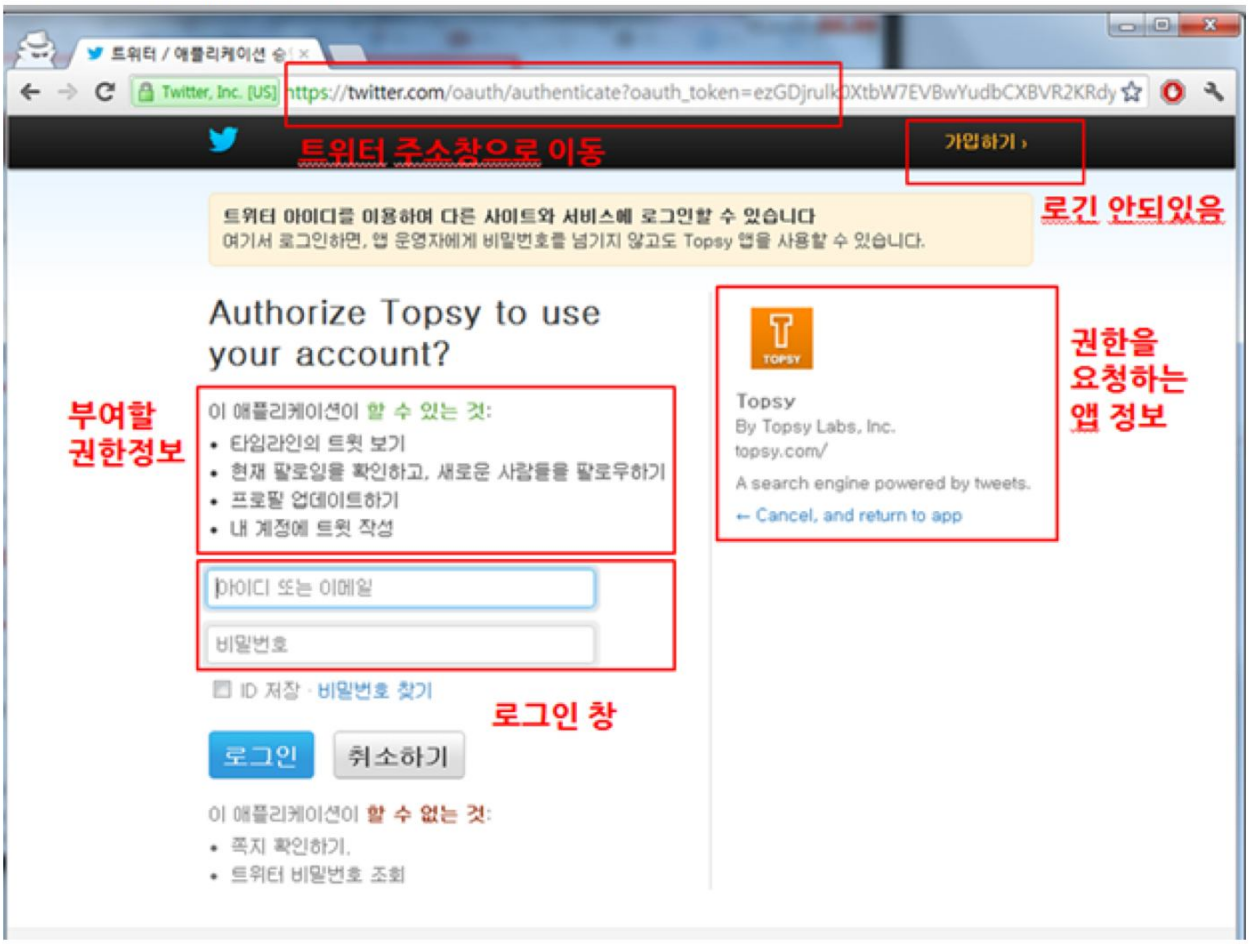


새로운 트위터 어플리케이션을 앱스토어에서 다운 받았지만, 아직은 어플리케이션을 신뢰할 수 없는 상황이라고 하자. 사용자는 이 어플리케이션에 아이디와 비밀번호를 저장하면 이 어플리케이션이 또 다른 어떤 짓(몰래 아이디/비밀번호를 수집하는 등)을 할 지 모르기 때문에, 어플리케이션에 비밀번호를 저장하고 싶지 않다. OAuth 1.0은 이 경우 트위터 단말 어플리케이션 (consumer)에게 인증토큰(access token)만을 전달하고 단말 어플리케이션이 인증토큰으로 트위터 API(service provider)를 사용할 수 있도록 해준다.

< OAuth 1.0a 인증 프로세스>



이 과정에서 사용자가 서비스 프로바이더의 API를 사용하겠다고 로그인 할 때, 컨슈머는 서비스 프로바이더의 로그인 화면으로 사용자를 리다이렉트 하게 된다. 이 서비스 프로바이더의 화면은 아래와 같은 정보를 제공한다.



사용자가 이 화면에서 로그인을 완료하면 다시 컨슈머로 리다이렉트 되고 동시에 컨슈머는 인증 토큰을 사용할 수 있게 된다. 이 인증 토큰은 컨슈머가 사용자의 아이디/패스워드 없이 허가 받은 API에 접근할 수 있게 해준다. 사용자는 컨슈머에 저장되어있는 인증 토큰이 유출 되더라도 서비스 프로바이더의 관리자 화면에서 언제나 그 인증토큰의 권한을 취소(revoke)할 수 있다.

OAuth 1.0a의 인증 과정에서 request token 등 추가적인 개념에 대해서는 이 글에서는 자세히 설명하지 않는다. OAuth 1.0a인증 과정에 대한 자세한 설명은 관련 자료가 많이 있으니 참고하길 바란다.

인증토큰

OAuth 1.0a 인증이 완료가 되면 컨슈머 (예를 들면 트위터 모바일 어플리케이션)은 사용자의 아이디/패스워드를 직접 저장하게 되는 것이 아니라, 인증토큰(access token)을 받게 된다. 이 인증 토큰은 OAuth 2.0에서도 같은 개념으로 사용된다. 인증토큰은 커베로스(Kerberos)의 티켓 개념과 비슷하다라고 할 수 있는데, 아래와 같은 특징을 가진다.

1. 컨슈머가 아이디/패스워드를 가지지 않고 API를 사용할 수 있음
2. 필요한 API에만 제한적으로 접근할 수 있도록 권한 제어 가능
3. 사용자가 서비스 프로바이더의 관리 페이지에서 권한 취소 가능
4. 패스워드 변경 시에도 인증 토큰은 계속 유효함.

자, 그러면 사용자의 스마트폰에 트위터를 인증해 놓은 상태에서, 스마트폰을 분실했을 때, 핸드폰을 습득한 사람이 내 트위터에 접근하지 못하게 하기 위해서 어떻게 해야 할까? 트위터 사이트에서 비밀번호를 바꿔도 트위터 어플리케이션은 계속 사용할 수 있다. 핸드폰을 분실 했다면, 아래와 같이 트위터 관리 페이지에서 어플리케이션 인증을 취소(revoke)하면 된다.
또한, 같은 원리로 트위터 비밀번호를 바꾸어도 내가 트위터 인증을 해놓은 수많은 어플리케이션들과 트위터 내보내기 설정해 놓은 서비스 들에서 다시 인증할 필요 없이 그대로 사용할 수 있다.

OAuth 2.0 API 인증을 위한 만능 도구상자

OAuth 2.0은 OAuth 1.0a 와 호환되지 않으며 사용되는 용어부터 시작해서 많은 것들이 다르다. OAuth 1.0a가 만들어지고 어느 정도 지난 후에 IETF 표준이 된 반면에 OAuth 2.0은 거의 초기부터 IETF 표준 프로세스 안에서 만들어지고 있다. OAuth 2.0의 첫 번째 draft가 등록된 것은 2010년 4월 이었고, OAuth 1.0a에서 불편하다고 느꼈던 모바일에서의 사용성 문제나 signature 생성과 같은 개발이 복잡하고 CPU를 많이 소비하는 기능의 단순화, 기능과 규모의 확장성 등을 지원하기 위해 만들어 졌다. 그러기 위해 표준이 매우 커지고 복잡해 졌는데, 이것은 OAuth 1.0a 표준의 정식 명칭이 OAuth 1.0 프로토콜(The OAuth 1.0 Protocol) 인데 반해 OAuth 2.0의 정식 명칙이 OAuth 인증 프레임웍(The OAuth 2.0 Authorization Framework)“ 인 것을 봐도 알 수 있다. 또한 OAuth 1.0은 하나의 RFC문서로 되어있지만, OAuth 2.0은 여러 개의 표준으로 작업이 이루어지고 있으며 그 표준들은 http://tools.ietf.org/wg/oauth/ 에서 확인 할 수 있다. OAuth 1.0a가 인증을 위한 만능 칼 이라면, OAuth 2.0은 인증을 위한 도구상자라고 할 수 있다. 다만 그 안에는 몇몇 위험한 도구들이 섞여있으니 검증된 도구만을 사용해야 하는데 그 안에 무엇이 있는지 알아보자.

OAuth 2.0이 1.0a에서 개선된 부분을 정리해보자

1. **간단해 졌다.**
   OAuth 1.0a에서는 https 가 필수가 아니었기 때문에 API를 호출할 때 signature를 생성해서 호출해야 했다. 때문에 OAuth 1.0a API를 테스트 하려면 curl등을 사용하기 힘들고 별도의 API 콘솔등을 사용해서 테스트 해야 했다. OAuth 2.0의 Bearer 토큰 인증 방식을 쓰면 더 이상 signature 가 필요 없기 때문에 API를 테스트하거나 예제를 만들 때 간단하게 curl등 직관적인 방법을 사용해서 문서화하고 개발 할 수 있게 되었다.
2. **더 많은 인증 방법을 지원**
   OAuth 1.0a는 한가지 인증 방식을 제공한다. HMAC을 이용한 암호화 인증 방식이다. 하지만 OAuth 2.0은 시나리오별로 여러가지 인증 방식을 제공하기 때문에 웹브라우저, 모바일 등의 다양한 시나리오에 대응할 수 있게 해준다.
3. **대형 서비스로의 확장성 지원**
   커다란 서비스를 만들기 위해서는 인증 서버를 분리할 수 있어야 하고 또한 인증 서버를 다중화 할 수 있어야 한다. OAuth 2.0에서는 실제 API 서비스를 하는 서버와 인증 역할을 하는 authorization server의 역할을 명확히 구분함으로서 인증서버의 분리와 다중화 등에 대한 고려가 되어있다.

또한 OAuth 1.0a에서는 사용자(user), 써드파티 어플리케이션(consumer), API 서버(service provider)로 불리우던 것이 OAuth 2.0에서는 용어도 바뀌었다.

- Resource Owner : 사용자
- Resource Server : API 서버
- Authorization Server : 인증서버 (API 서버와 같을 수도 있음)
- Client : 써드파티 어플리케이션

OAuth 2.0 spec 구성

위 홈페이지에서도 볼 수 있지만, 현재 OAuth 는 oauth-v2 와 oauth-v2-bearer 라는 2개의 표준이 가장 핵심적인 부분이며 현재 RFC에 등록되기 위한 과정을 밟고 있는 중이다. 대부분 OAuth 2.0 지원"이라고 하는 서비스 들은 이 2가지 spec을 지원한 다는 것을 의미한다. 이 외에도 SAML, JSON 웹 토큰, MAC 토큰 등의 방식이 있지만 아직 활발히 수정 중 이기 때문에 실제 서비스에서 사용되고 있지는 않다.

OAuth 2.0의 버전들

API 서비스를 일찍 시작한 회사들은 기존의 OAuth 1.0a 를 계속 유지하는 경우도 많이 있다. 1.0a 와 2.0을 사용하는 유명한 서비스 들은 아래와 같다.

OAuth 2.0의 최종 버전이 나오는데 오래 걸리고 있기 때문에 Facebook, Google 등은 OAuth 2.0 spec의 draft 버전이 올라가면서 그 개선사항들을 구현해 오고있다. 현재 최신 draft 는 31버전인데 주요 서비스들이 구현하고 있는 버전들은 아래와 같다.

서비스	draft	출처
Google	22	https://developers.google.com/accounts/docs/OAuth2
Facebook	12	http://developers.facebook.com/docs/authentication/
SalesForce	10	http://wiki.developerforce.com/page/Digging_Deeper_into_OAuth_2.0_on_Force.com

위 버전들은 서비스가 제공하는 OAuth 인증 페이지에서 수집한 것들이다. 보통 OAuth 2.0 spec 의 draft 10 ~ 22 정도를 많이 구현하였는데 중요한 변경사항은 아래와 같은 것이 있다.

1. HTTP 헤더 앞 부분에 bearer 토큰을 보내기 전에 OAuth“ 에서 Bearer“로 수정
2. oauth_token“ 이름이 access_token“ 으로 변경
   위 변경사항 때문에 아직까지는 두 가지를 모두 받아들이는 서비스들이 많다. RFC로 확정이 되면 더 이상은 변경이 불가능 하기 때문에 최종 안으로 정리가 될 것이다.

다양한 인증 방식 (Grant types)

앞에서 OAuth 1.0a가 동작하기 위해서는 사용자, 컨슈머, 서비스 프로바이더가 필요하고 3-legged OAuth 라고 불리우기도 한다고 하였지만, OAuth 2.0은 2-legged 모델 등 다양한 인증 방식을 지원한다. 3-legged 모델의 장점은 최종 사용자 뿐 아니라 개발자가 누구인지도 인증하기 때문에 어떤 어플리케이션이 API를 사용하는지 통계/과금 을 위한 필수적인 정보를 얻을 수 있다는 점이다. user-agent 나 referer 같이 변경될 수 있는 값이 아닌 인증을 통해 확실하게 구분할 수 있기 때문에 개발자가 API를 비정상적으로 호출하고 있다거나 할 때 개발자와 직접 연락하는 등의 조치를 취할 수도 있다. 그럼에도 불구하고 리다이렉트와 같은 OAuth에서 필요로하는 동작이 불가능한 시나리오, 둘이서만 인증하는 시나리오 등을 지원하기 위해 OAuth 2.0은 2-legged 모델도 지원하나, 그래도 OAuth 2.0에서 가장 기본이 되는 것은 3-legged 모델이다.

Client 는 기본적으로 Confidential Client 와 Public Client 로 나뉜다.

- Confidential 클라이언트는 웹 서버가 API를 호출하는 경우 등과 같이 client 증명서(client_secret)를 안전하게 보관할 수 있는 Client를 의미한다.
- Public Client는 브라우저기반 어플리케이션이나 모바일 어플리케이션 같이 client 증명서를 안전하게 보관할 수 없는 Client를 의미하는데 이런 경우 redirect_uri 를 통해서 client를 인증한다.

OAuth 2.0이 지원하는 인증방식은 client 종류와 시나리오에 따라 아래 4가지가 있다. 하지만 실제로 Authorization Code Grant와 Implicit Grant를 제외하고는 일반적인 3-legged OAuth 가 아니기 때문에 open API에서는 많이 사용되지 않는다.

1. **Authorization Code Grant**
   웹 서버에서 API를 호출하는 등의 시나리오에서 Confidential Client가 사용하는 방식이다. 서버사이드 코드가 필요한 인증 방식이며 인증 과정에서 client_secret 이 필요하다.
   로그인시에 페이지 URL에 response_type=code 라고 넘긴다.
2. **Implicit Grant**
   token과 scope에 대한 스펙 등은 다르지만 OAuth 1.0a과 가장 비슷한 인증방식이다. Public Client인 브라우저 기반의 어플리케이션(Javascript application)이나 모바일 어플리케이션에서 이 방식을 사용하면 된다. Client 증명서를 사용할 필요가 없으며 실제로 OAuth 2.0에서 가장 많이 사용되는 방식이다.
   로그인시에 페이지 URL에 response_type=token 라고 넘긴다.
3. **Password Credentials Grant**

이 방식은 2-legged 방식의 인증이다. Client에 아이디/패스워드를 저장해 놓고 아이디/패스워드로 직접 access token을 받아오는 방식이다. Client 를 믿을 수 없을 때에는 사용하기에 위험하기 때문에 API 서비스의 공식 어플리케이션이나 믿을 수 있는 Client에 한해서만 사용하는 것을 추천한다.
로그인시에 API에 POST로 grant_type=password 라고 넘긴다.

4. **Client Credentials Grant**
   어플리케이션 이 Confidential Client일 때 id와 secret을 가지고 인증하는 방식이다.
   로그인시에 API에 POST로 grant_type=client_credentials 라고 넘긴다.
5. **Extension**
   OAuth 2.0은 추가적인 인증방식을 적용할 수 있는 길을 열어놓았다. 이런 과도한 확장성을 메인 에디터인 Eran Hammer는 매우 싫어했다고 한다.

Password Credentials Grant와 Client Credentials Grant는 기본적으로 우리가 생각하는 OAuth의 프로세스를 따르지 않기 때문에 반드시 인증된 client에만 사용되어야 하며 가능하면 사용하지 않는 것이 좋다.

다양한 토큰 지원(Access token)

OAuth 2.0은 기본적으로 Bearer 토큰, 즉 암호화하지 않은 그냥 토큰을 주고받는 것으로 인증을 한다. 기본적으로 HTTPS 를 사용하기 때문에 토큰을 안전하게 주고받는 것은 HTTPS의 암호화에 의존한다. 또한 복잡한 signature 등을 생성할 필요가 없기 때문에 curl이 API를 호출 할 때 간단하게 Header 에 아래와 같이 한 줄을 같이 보내므로서 API를 테스트해볼 수 있다.
Authorization: Bearer

또한 OAuth 2.0은 MAC 토큰과 SAML 형식의 토큰을 지원할 수 있지만 현재 MAC 토큰 스펙은 업데이트 되지 않아 기한 만료된 상태이고 SAML 토큰 형식도 아직은 활발하게 수정중이기 때문에 사용할 수 없는 상태이다. 정리하자면, OAuth 2.0은 다양한 토큰 타입을 지원한지만 실질적으로는 Bearer 토큰 타입만 지원한다.

Refresh token

클라이언트가 같은 access token을 오래 사용하면 결국은 해킹에 노출될 위험이 높아진다. 그래서 OAuth 2.0에서는 refresh token 이라는 개념을 도입했다. 즉, 인증 토큰(access token)의 만료기간을 가능한 짧게 하고 만료가 되면 refresh token으로 access token을 새로 갱신하는 방법이다. 이 방법은 개발자들 사이에서는 논란이 있는데, 토큰의 상태를 관리해야 해서 개발이 복잡해질 뿐만 아니라 토큰이 만료되면 다시 로그인 하도록 하는 것이 보안 면에서도 안전하다는 의견이 있기 때문이다.

API 권한 제어 (scope)

OAuth 2.0은 써드파티 어플리케이션의 권한을 설정하기 위한 기능이다. scope의 이름이 스펙에 정의되어있지는 않으며 여러 개의 권한을 요청할 때에는 콤마등을 사용해서 로그인 시에 scope를 넘겨주게 된다.

http://example.com/oauth?....&scope=read_article,update_profile

OAuth 2.0은 위험한가?

Eran Hammer가 OAuth 2.0가 형편없다고하며 자신의 블로그에서 제시한 이유는 OAuth 2.0이 너무 복잡하고 안전하지 않다는 점이다. 그는 자신의 블로그에 이렇게 된 이유를 설명하고 있다. OAuth 2.0 스펙에 참가한 사람들은 주로 웹 커뮤니티 멤버들과 대형 기업멤버들로 되어있었다. 웹 관련 멤버들은 OAuth 1.0a와 비슷하되 단점을 개선한 버전을 만들기를 원하는데 반해, 기업쪽 사람들은 OAuth 2.0을 자신의 솔루션을 위한 인증 framework으로 사용할 수 있도록 유도하며 OAuth 2.0을 너무 복잡하게 만들었고 또 거의 무한한 확장성을 가질 수 있게 하면서 거의 모든 것에 "OAuth 2.0 호환' 딱지를 붙일 수 있게 되었다는 것이다. 그는 OAuth 2.0이 더 간결하고 제한된 스펙이 되기를 원했다. 실제 블로그에서는 OAuth 1.5 정도의 내용을 가진 새로운 OAuth 2.1이 10page 정도의 짧은 문서로 새로 나왔으면 좋겠다" 라고 했다.

그가 블로그 글에서 지적한 OAuth 2.0의 문제점은 어떤 것들 일까?

1. OAuth 1.0a에서는 token 과 함께 token 비밀번호를 같이 받는데 2.0에서는 이 토큰 비밀번호가 없어져서 실제로 클라이언트를 구분하기가 힘들어 졌다.
2. OAuth 1.0a의 signature 기능을 없애서 SSL 기능에 의존함으로서 더 위험해졌다. 또한 이렇게 된 것은 기업 솔루션에 적용 되기위한 것이었다.
3. OAuth 2.0에서 토큰 유효기간을 설정 할 수 있고 또 만료가 되면 갱신 되어야 한다. 이 변화는 OAuth 1.0 개발자들이 토큰 상태까지 관리해야 한다는 점에서 매우 커다란 변화이다. 이 기능은 자체적으로 인코딩 되어서 저장되는 토큰을 위한 것인데 이런 토큰은 권한을 취소(revoke) 하는데에 문제가 있기 때문에 또 다시 유효기간을 짧게 가져가게 된다.
4. 권한 부여 방식
   OAuth 2.0에서는 너무 많은 방식을 지원 하고있다. 이는 여러 보안상의 문제점의 가능성을 의미한다.
   또한 OAuth 2.0을 IETF에서 시작한 것을 후회하는 말도 하면서, 그렇다고 딱히 다른 대안이 있는 것도 아니라는 의견도 밝힌다. 현재 워킹 그룹이 합의를 하지 못하고 있는 사항들에 리스트도 적었는데, 그만큼 OAuth 2.0스펙이 너무 포괄적이며 스펙은 더 많은 제한을 가지고 있고 많은 것이 결정되어있어야 한다는 이야기라고 그는 계속 주장 한다

위 글에 대한 반박 글은 참 많지만 그 중에 유명한 것은 아래 2개이다.

- In Defense of OAuth 2.0
- The OAuth 2 Sky is NOT Falling

위 글에서는 OAuth 2.0의 장점들과 실질적으로는 OAuth 2.0 워킹 그룹이 잘 해나가고 있다는 내용의 글들이다. Eran Hammer는자기 자신의 첫 번째 글이 커다란 반향을 일으키자, 내용을 어느 정도 개선한 두 번째 글을 쓰기도 했다. 이 글에서 몇몇 OAuth 2.0을 잘 받아들인 단체들을 언급하면서, Facebook이나 Google은 OAuth 2.0이 직접 내렸어야 하지만 내리지 못한 결정들을 직접 내리고 OAuth 2.0을 성공적으로 구현했다고 이야기 하고있다. 하지만 그는 자신의 신념에는 변화가 없다는 뜻을 다시 한번 비추었다. 즉 OAuth 2.0이 명확하게 정의하지 않을 부분을 잘 구현한

다면 OAuth 2.0은 꽤 쓸만한 표준이나, Eran Hammer는 현재의 OAuth 2.0은 너무 많은 부분들을 허용하고 있고 복잡하기 때문에 안전하지 않다고 주장한 것이다.

정리하자면, 가장 많이 사용되는 Grant type인 Authorization Code 방식이나 Implicit 방식을 사용하고 access token으로서는 Bearer 토큰을 사용하며 HTTPS를 통해서 서비스하고 널리 사용되는 라이브러리들을 사용해서 구현한다면 OAuth 2.0은 안전하다고 할 수 있다. 다만 아직 표준 진행중인 토큰 방식을 사용하거나 자신만의 Extension을 정의해서 사용하는 경우도 OAuth 2.0이라고 부를 수는 있지만 아직은 안전하지 않다고 할 수 있다.

OAuth 2.0에 대한 예상질문

이상 OAuth 2.0에 대해서 알아보았다. 마지막으로 간단한 몇 가지 사항을 짚어보자

1. SSL을 사용하지 않는 bearer 토큰은 안전하지 않나?
안전하지 않다. MAC token을 사용할 수 있지만 OAuth 2.0에서는 실질적으로 관리되고 있지 않은 스펙이다. OAuth 2.0을 이용한 API 호출에는 반드시 HTTPS를 사용해야 한다.

2. API를 서비스하는데 HTTPS를 쓸 수 없는 상황이다. 어떻게 해야 하는가?
그런 경우라면 현 시점에서는 OAuth 1.0a 를 사용하는 것이 최선이다. 아직 OAuth 2.0 스펙은 자체 암호화를 지원하지 않는다.

3. OAuth 2.0의 대안이 있는가?
의미있는 대안은 현재 없다. Facebook, Google 등이 모두 API 인증방식으로OAuth 2.0을 채택했기 때문에 대세는 이미 OAuth 2.0이다.

4. API 사용자를 생성하는 좋은 방법은?
이 이슈는 API의 고전적인 이슈 중 하나이다. API인증을 하기위한 사용자를 생성하는 API는 어떻게 인증해야 하는가에 대한 이야기 이다. API를 사용해서 사용자를 생성한다면, CAPCHA 등의 방법을 사용할 수 없기 때문이다. 보통 다양한 조건체크, email 확인 등의 방법을 사용하기도 하지만, 가능하면 사용자 추가는 그냥 웹 브라우저상에서 직접 하도록 하는 것이 정답이다.

마무리 하며

암호 기술과 함께 인터넷상에서 인증 관련 기술들은 매우 빠르게 발전하고 있다. 최근에는 SHA-3 해쉬 알고리즘이 기나긴 선정과정을 걸쳐 최종 선정되는 커다란 뉴스가 있었고API인증에 있어서도 OAuth 2.0 뿐만 아니라 openid 진영의OpenID Connect 도 활발히 활동하고 있다. 많은 우여곡절 속에서 OAuth 2.0 spec은 최종안을 위해서 달려가고 있는데 예상보다 많은 시간이 걸리고 있고 또 잡음이 있다는 것은 그만큼 많은 사람들과 기업들에게 OAuth 2.0가 중요하고 많은 기대를 하고 있다는 것을 말해준다.

OAuth 2.0의 기본 스펙은 안전하다. 다만 스펙에 참여하고 있는 일부 기업들의 욕심으로 생긴 일부 추가적인 기능들과 방향성에 논란이 있는 상황이다. 이슈가 되는 Refresh 토큰이나 추가기능을 제공하는 extension, 그리고 아직 RFC 단계에 있지 않은 스펙 들 때문에 OAuth 2.0전체가 안

전하지 않다고 할 수는 없다. HTTP 1.1스펙의 PATCH 나 HEAD 메소드는 거의 사용되지 않지만 그렇다고 HTTP 1.1 의 GET/POST 메소드 까지 버릴 수는 없지 않을까?

AES나 SHA-3 등과 같은 암호화 알고리즘이 NIST같은 미국 정부기관의 주도로 정해지는 것과 달리 OAuth 는 IETF의 열린 프로세스에 따라서 스펙이 결정되고 메일링 리스트를 통해서 주로 토론이 이루어진다. 이 글을 쓰면서 인터넷 실명제 위헌 판결에 따라서 개인 인증에 대한 중요성이 부각되고 있는 우리나라에서 인터넷에서의 주민번호 대체품으로서 한국 정부가 OAuth 프로바이더를 만드는 것은 어떨까 하는 상상도 해보았다. 더 많은 사람들이 OAuth 2.0등의 열린 인증 방법에 관심을 가지고 더 많은 자료들이 API를 통해 공개되고, 또 안전하게 접근할 수 있는 날이 오기를 기대해 본다.

참고자료

- OAuth 2 OAuth 2 공식 홈페이지
- IETF OAuth 2 spec
- OAuth 2.0 and the Road to Hell (2012.7.26)
  모든 사건의 시작이된 OAuth 2.0 editor Eran Hammer의 폭탄선언
- On Leaving OAuth (2012.7.30)
  윗 글의 후속 글
- In Defense of OAuth 2.0
- The OAuth 2 Sky is NOT Falling
- An Introduction to OAuth 2
  OSCON 2012 발표자료
- OAuth 2 Simplified
- IETF OAuth Word Group 메일링 리스트