목차 ✚

1. OAuth2 구성요소

OAuth2.. 결국은 인증을 위해서 사용하는 건데, 얼마나 어렵겠어라고 생각했는데 "어려웠다". 적어도 나에게는 어려웠다. 내가 머리가 나빠서인건가 라는 생각을 하기도 했는데, 다행히 "복잡한게 OAuth2"의 단점이라는 평가도 있어서 나름 안도하고 있는 중.

그래서 OAuth2의 구성요소들을 먼저 정리하기로 했다. 뭔가 의미가 애매모호한 것들이 많아서, 개념을 잡아 놓지 않으면 나중에 상당히 헷갈릴 수 있다.

1.1. Resource Server (Service Provider)

자원(protected resource)를 제공하는 서버다. Client는 **access token**을 이용해서 자원에 접근할 수 있다.

1.2. Resource Owner

Resource server에 있는 자원의 소유자. 사람이 소유자가 될 필요는 없다. 애플리케이션이나 기계가 자원의 소유자가 될 수도 있다.

1.3. Client

Cleint는 유저 계정에 접근을 시도하는 애플리케이션이다. 애플리케이션의 사용을 위해서 로그인이 필요한 경우도 있다.

1.4. Authorization Server

인증을 수행하는 서버로 로그인 및 접근허가 여부를 검사해서 Authorization code과 access token을 발행하는 일을 한다..

1.5. Protected resource

접근허가를 필요로 하는 **자원**이다. 자원에 접근하기 위해서는 Authorization server를 통해서 access token을 받아야 한다. 자원은 접근제어를 필요로하는 모든 인터넷 객체다. 즉 문서파일, 음악, 오디오, 사용자 정보, API들이다.

1.6. Authorization grant

Authorization grant는 client가 획득한 access token에 부여된 리소스 소유자의 권한을 의미한다.

1.7. Authorization code

Authorization code는 authorization server가 발급해주는 문자열로 client와 resource owner를 중개하기 위해서 사용한다. Client는 Authorization code를 이용해서 Access token을 얻을 수 있다.

1.8. Access token

Access token은 자원(protected resources)에 접근하기 위해서 사용하는 **증명서**로 접근을 요청하는 client의 권한과 범위(scope)를 알아낼 수 있는 일련의 문자열로 구성된다.

서버/클라이언트 인증 모델의 경우 3rd-pary 애플리케이션이 "아이디/패스워드"를 가짐으로써 보안문제를 가지고 있다. OAuth2는 아이디/패스워드 대신에 access token을 교환함으로 보안성을 높일 수 있다.

1.9. Refresh token

유저가 접근승인을 받으면 **access token**을 발급받는다. 이 access token은 만료기간을 가지고 있어서, 만료기간을 넘어가게 되면 더 이상 사용할 수 없게 된다. 이 경우 access token을 발급받기 위한 과정을 다시 진행해야 한다.

많은 경우 token endpoint(아래 설명한다)는 **access token**과 **refresh token**을 함께 제공한다. 유저는 refresh token을 이용해서 access token을 재 발급받을 수 있다. 대략 다음과 같은 방식이다.

Client 요청이다.

```
import com.google.appengine.api.urlfetch.*
import groovy.json.*
import java.net.URLEncoder

def client_id = 'mobile_android'
def client_secret = 'secret'
def redirect_uri = 'http://localhost:8080/oauth2_callback'

if (!params.token)
{
    out << "No refresh token given..."
    return
}

def token = params.token

//exchange code for real oauth token
URL tokenURL = "http://localhost:9001/rest/oauth/token".toURL()
HTTPResponse res = tokenURL.post(deadline: 30, payload:"refresh_token=${
out << res.text
```

서버 응답은 다음과 같다.

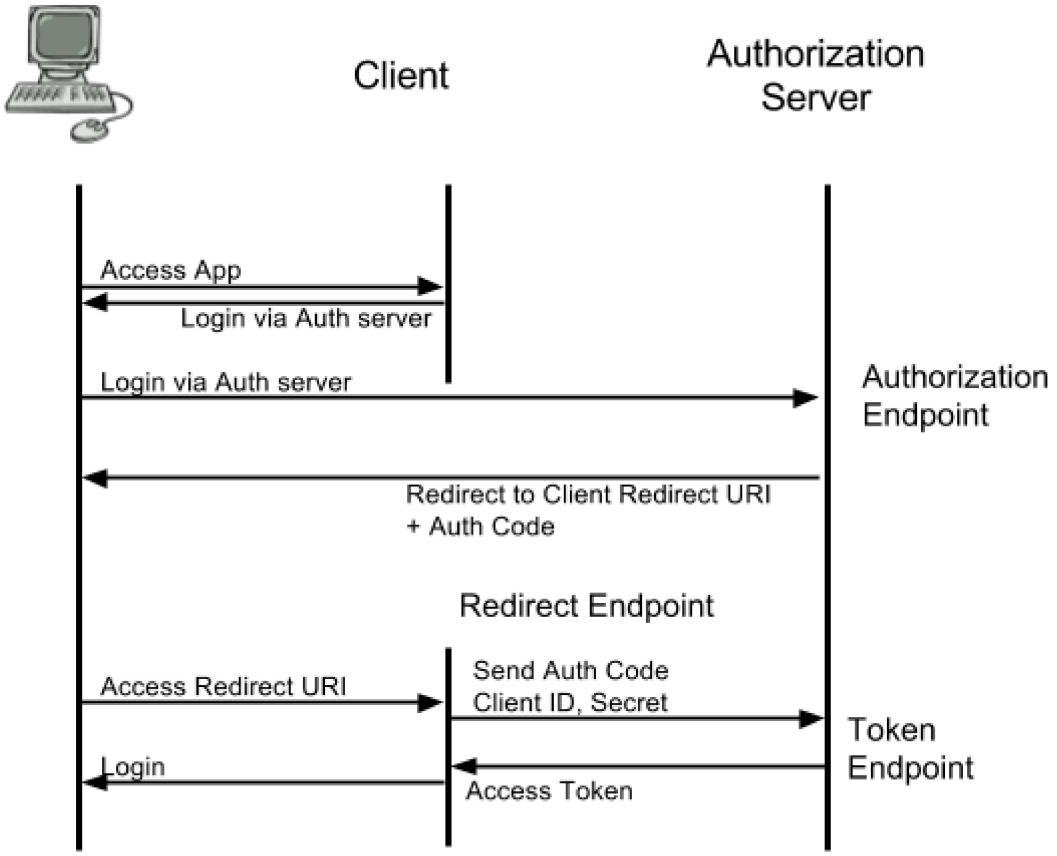
```
{
    "access_token":"78a10516-4b13-4442-90b1-050acb7226d6",
    "refresh_token":"3427f260-5101-4139-a094-75e32d4bbb2c",
    "expires_in":43199,
    "scope":"tcloud"
}
```

1.10. scope

접근하고자 하는 데이터셋 영역이다. 예를들어 멀티미디어 데이터를 서비스하고 있다면, "사진", "음악", "동영상" 별로 따로 scope를 지정해서 접근요청을 할 수 있다.

1.11. OAuth2 endpoint

OAuth2는 몇 개의 endpoint 셋을 정의하고 있다. Endpint는 web server의 URI로 나타낼 수 있다. 여기에서는 OAuth2에서 정의하고 있는 endpoint 셋을 살펴보려고 한다.



- Authorization endpoint
- token endpoint
- Redirection endpoint

1.11.1. Authorization endpoint

Authorization endpoint는 authorization server에 있는 endpoint로 client 애플리케이션에 (resource owner로의) 로그인, grant, authorization 서비스를 제공한다.

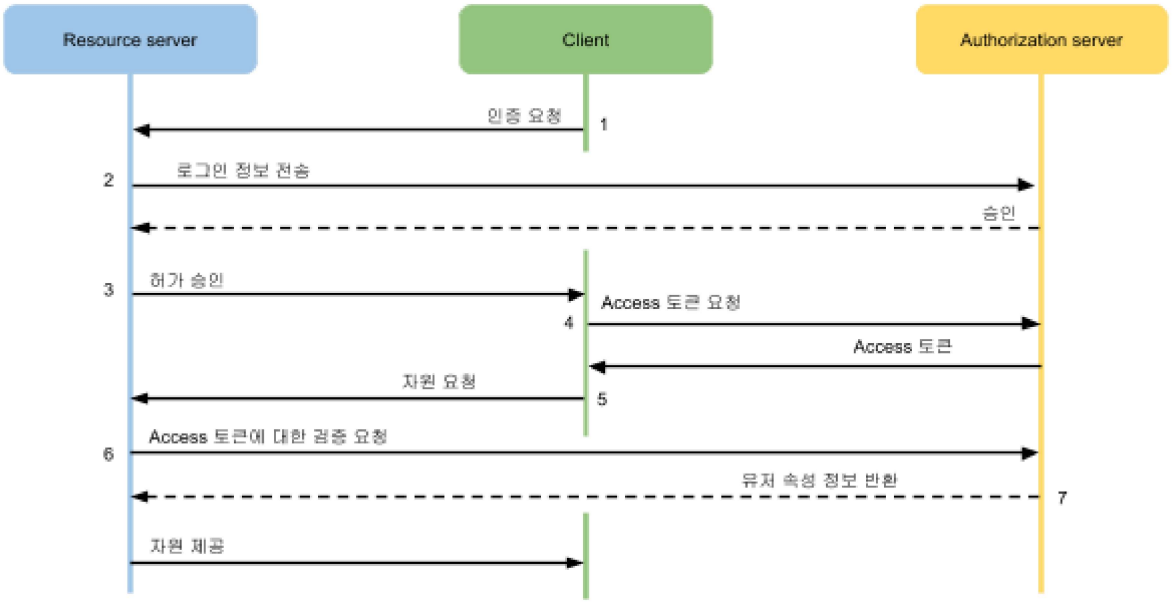
1.11.2. Token endpoint

Token endpoint는 authorization server에 있으며, client의 **authorization code**와 client ID, client secret를 받아서 access token을 발행하는 서비스를 제공하는 endpoint다.

1.11.3. Redirection endpoint

Authorization endpoint에서 인증과정을 거친후 resource owner로 리다이렉트 서비스를 제공하는 endpoint다. Redirection endpoint는 client 애플리케이션이 제공한다.

2. OAuth2 flow



1. Client가 Resource server에 대해서 인증을 요청한다.
2. Client는 아이디와 패스워드 정보를 입력할 테고, Resource server는 이 정보로 Authorization server에 승인 요청을 한다.
3. Resource server는 승인허락이 떨어지면, 이 정보를 클라이언트에 전달한다.
4. Client는 승인허락 정보를 가지고 Authorization server에 **Access 토큰**을 요청한다.
5. Client는 access 토큰을 이용해서 resource server에 자원을 요청한다.
6. Resource server는 client 의 access 토큰이 유효한지를 authorization server에 물어 본다.
7. Authorization server는 access 토큰이 유효한지와 기타 유저의 권한등을 포함한 정보를 resource server에 반환한다.
8. Resource server는 client에 자원을 제공한다.

뭐 쓸데 없이 복잡해 보이기도 하지만 아래와 같이 간단히 요약할 수 있다.

- Resource server에 대한 접근 정보는 Authorization server가 가지고 있다.
- 당연히 resource server는 client가 접근 요청에 대해서 Authorization server에 물어봐야 한다.
- Resource server는 client에게 Authorization server에 Access 토큰(인증및 권한 정볼를 가진)를 만들어 놨으니까. 가져가라고 한다.
- Client는 Access 토큰을 가지고 resource server에 자원을 요청할 수 있다.

3. OAuth2 인증방식

4가지의 인증방식을 제공한다. 주로 3-legged 방식인 **Authorization Code Grant**와 **Implicit Grant**를 사용하다.

3.1. Client 타입

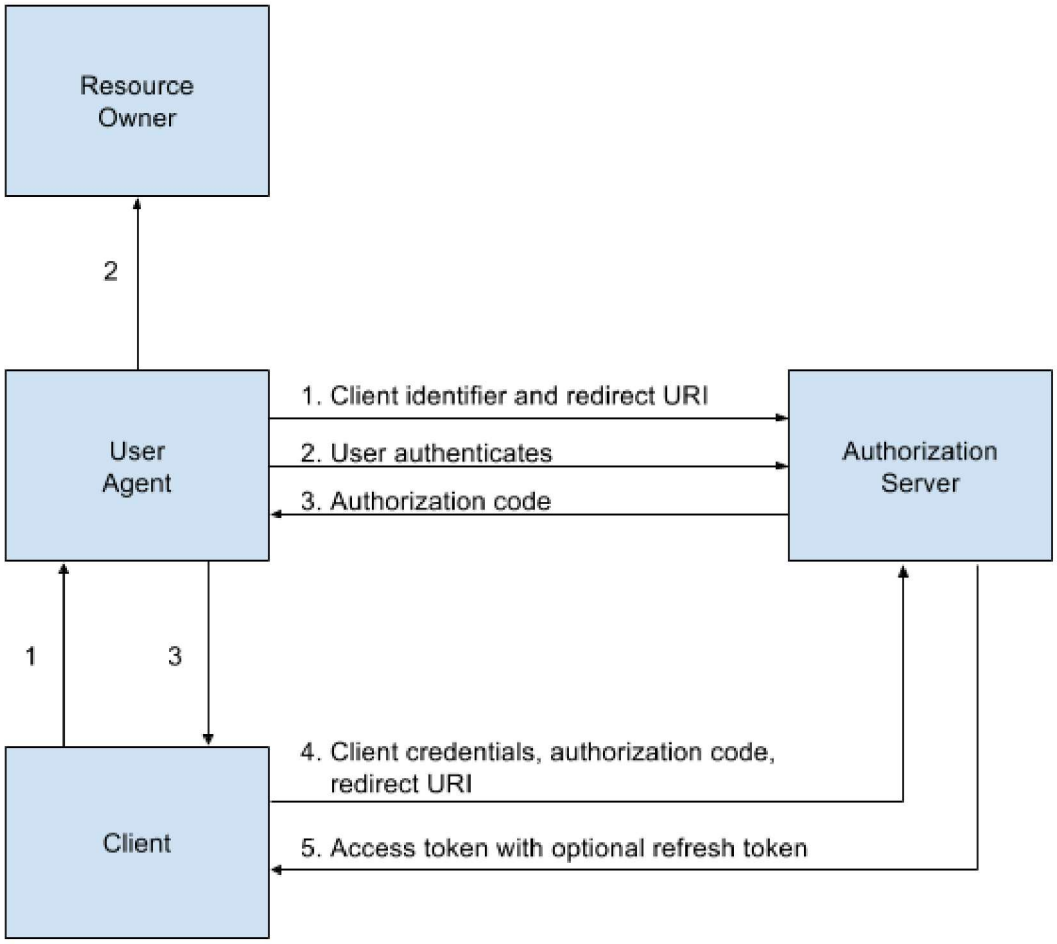
Client는 Confidential Client와 Public client 타입이 있다.

- Confidential Client : 접근하려는 client가 누구인지 확인(보증)할 수 있는 타입이다. 예를 들어 **웹 서버**가 **API 서버**를 호출하는 경우가 되겠다. API 서버에 접근하는 **웹 서버**는 특정할 수 있기 때문에 client_secret를 안전하게 보관할 수 있음을 보증할 수 있다.
- Public Client : 웹 브라우저나 모바일 애플리케이션 같은 client 타입이다. 이들 client는 임의의 client이기 때문에 client 증명서가 안전하다는 것을 보증할 수 없다. 이런 경우에는 redirect_uri를 이용해서 client를 인증한다.

3.2. Authorization Code Grant

Confidential Client가 사용하는 방식이다. OAuth service-side flow 라고 부르기도 한다.
`client_secret`를 보내는 것으로 인증한다. 로그인 할 때 URL에 `response_type=code`라고 명시한다.
Client는 user agent를 이용해서 resource owner에 자원을 요청한다.

Authorization Code grant 흐름은 다음과 같다.



1. Client는 user agent를 이용해서 Authorization server에 인증절차를 거치기 위한 일련의 정보들을 전달한다.
    - response_type : Authorization code grant 타입일 경우 "code"를 설정한다.
    - client_id : Client의 id
    - client_secret : client_id를 위한 secret 값이다. 이 값으로 Client를 인증하기 때문에, 다른 누구에게도 노출되면 안된다.
    - redirect_uri : 실제 인증을 수행할 authorization endpoint로 리다이렉트하기 위한 uri를 명시한다.
    - scopes : scopes는 접근 레벨을 조절하기 위해서 사용한다.
    - state :

    2. 이 과정은 때때로 두 단계로 이루어진다. 첫번째는 Basic Authentication으로 유저가 아직 메인 웹 애플리케이션에 로그인 하지 않았을 때, 로그인을 진행하는 과정이다.(페이스북에 로그인하지 않은 상태를 생각하면 되겠다.) 다음 단계는 client의 접근을 허용하는 단계다. 모바일 client를 사용하고 있다면 "이 앱을 모바일에서 사용하도록 허용하겠습니까? (Yes/No)"를 선택하는 단계다. 페이스북이나 구글 앱을 사용하고 있다면, 이 과정을 쉽게 이해할 수 있을 것이다.

3. Autohrization server는 User Agent로 authorization code를 반환한다. 나중에 authorization code를 이용해서 **access token**을 얻을 수 있다.

4. Client는 authorization code로 access token을 요청한다. Access token을 요청하기 위해서 사용하는 매개변수들은 다음과 같다.

- code : authorization code
- grant_type : 이 경우에는 authorization_code
- redirect_url :
- client_id 와 client_secret

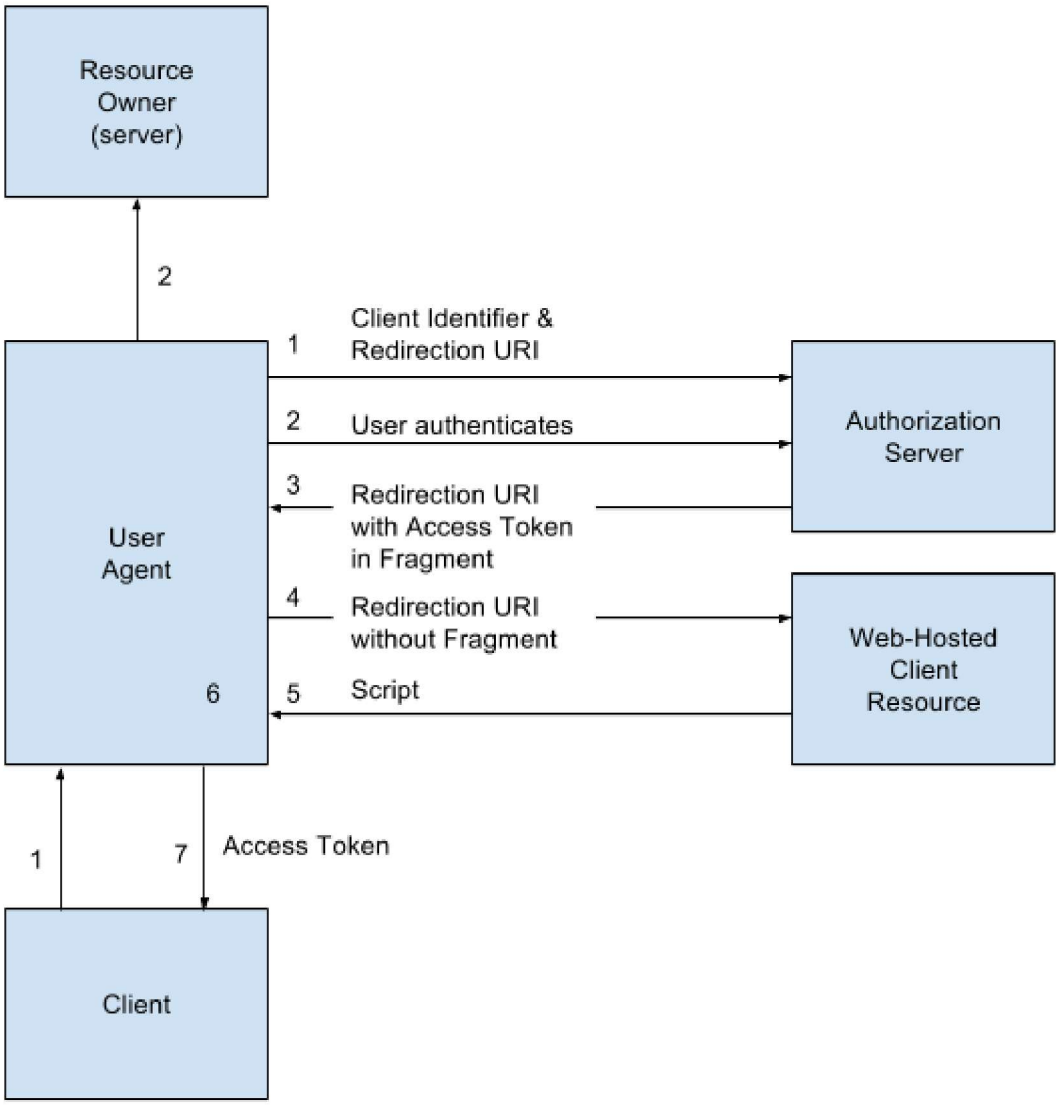
5. 드디어 access token을 얻었다. 다음은 access token 예제다.

```
1  {"access_token":"865b3ecf-54d3-413c-951e-fae0b00c906f","token_type":"
2  "refresh_token":"addc590a-c79c-4e7e-a1a7-f98921765c20","expires_in":4
```

3.3. Implict Grant Flow

Implict Grant Flow는 Client-Side Flow라고 부르기도 한다. 암시적(Implict) Grant flow 라고 부르는 이유는 client 애플리케이션이 access token을 얻기 위해서 authorization token을 필요로 하지 않기 때문이다. 이는 전체 인증/권한 프로세스를 좀 더 간단하게 만들어 주지만 (Authorization Code Grant에 비해서)보안성이 떨어진다.

이 방식은 보안성을 높이기 위해서, client측 웹 애플리케이션으로 하여금 짧은 시간동안(몇 시간 이내) 자원을 임시로 허용하는 식으로 사용해야 한다.



1. Authorization grant flow와 마찬가지로 service-side flow(authorization server)에 요청을 보내는 것으로 인증과정을 시작한다. 그림에서 처럼 client identifier과 redirection URI를 보낸다. 주목할 점은 **client_secret**를 전송하지 않는다는 점이다. Authorization server에 넘겨주는 파라메터들은 다음과 같다.
    - response_type : token으로 설정한다.
    - client_id : Client의 아이디
    - redirect_uri
    - scope
    - state : 옵션이긴 하지만 사용하는 걸 권장한다. 랜덤으로 state 값을 만든다음 authorization server에 넘겨주면, 나중에 이 값을 비교하는 것으로 XSS 공격을 어느 정도 막을 수 있다.

    아래는 authorization server로 보내는 예제 프로그램이다.

```
import java.net.URLEncoder

def client_id = 'client-side'
def redirect_uri = 'http://localhost:8080/oauth2_implicit_callb

def scopes = [
    'customer'
]

def state = new Random(System.currentTimeMillis()).nextInt().to
session.state = state

redirect "http://localhost:9001/rest/oauth/authorize?client_id=

def client_id = 'client-side'
def redirect_uri = 'http://localhost:8080/oauth2_implicit_callb

def scopes = [
    'customer'
]

def state = new Random(System.currentTimeMillis()).nextInt().to
session.state = state

redirect "http://localhost:9001/rest/oauth/authorize?client_id=
&redirect_uri=${URLEncoder.encode(redirect_uri, 'UTF-8')}
&response_type=token&scope=${URLEncoder.encode(scopes.join(' ')
&state=${URLEncoder.encode(state, 'UTF-8')}"
```

2. 이쯤에서 유저는 로그인 화면을 만날 거다.

3. Authorization server는 client web application으로 리다이렉트를 한다. **Access token**은 리다이렉트 URI에 함께 전송한다.

4. User-agent(보통 웹 브라우저)는 client-side web application을 (리다이렉트 URI로)로 접근한다. Client-side web application은 자바스크립트를 포함한 HTML 문서를 전송한다.

5. HTML 문서는 몇개의 자바스크립트를 포함한다. 대략 아래와 같은 문서를 전송할 것이다.

```
<!doctype html>
<html>
  <head>
    <title>Oauth2 Implicit Callback Page</title>
  <head>
<body>

<div id="oauthParams" data-state=${request.getAttribute('state'

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/
<script src="/js/oauth.js"></script>

</body>
</html>
```

- state : 이 정보는 session 정보로

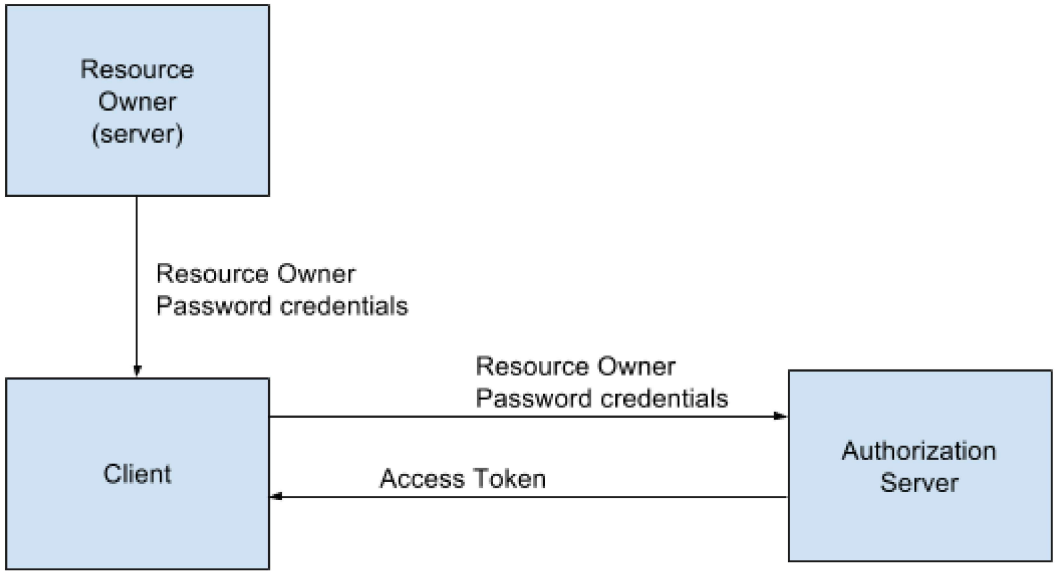
- script : 두 개의 스크립트 소스를 포함한다.
    1. CDN (/w/man/12/CDN)으로 호스팅되는 jquery(혹은 다른 자바스크립트 프레임워크)
    2. OAuth를 위한 javascript 코드

6. /js/oauth.js는 access token을 추출하고 state 값을 비교한다.

3.4. Password Credential Grant

OAuth2의 인증방식의 흐름은 **access token**을 얻는 방식에 따라 나뉜다고 보면 된다.

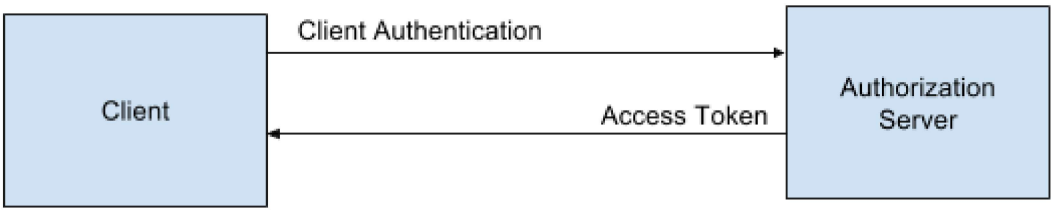
이 방식은 2-legged 방식의 인증으로 아이디와 패스워드를 이용해서 access token을 직접 얻어오는 아주 간단한 방법을 사용한다. 안드로이드 애플리케이션이 아이디와 암호를 직업 요청하는 방식이다. 서버/클라이언트 인증방식과 차이가 없다고 보면된다. Client에게 암호와 패스워드가 직접 노출되기 때문에 3rd-party 응용 프로그램 개발에는 사용할 수 없는 방식이다. 이 방식은 "공식적인 애플리케이션"에서만 사용할 수 있다.



1. Client는 유저 아이디와 패스워드를 묻는다. 유저는 아이디와 패스워드를 client 애플리케이션에 직접 입력한다.
2. Client는 유저 아이디와 패스워드로 authorization server에 token을 요청한다.

3.5. Client Credentials Grant

Client 자신이 resource에 대한 접근권한을 가지는 방식으로 OAuth1의 2-legged flow와 비슷하다. 특정 Client에게 backend API의 접근을 허용하기 위한 용도로 사용할 수 있다.



Client 측 코드는 아래와 같은 형태를 가진다.

```
import com.google.appengine.api.urlfetch.*
import groovy.json.*
import java.net.URLEncoder

if (!params.client_id)
{
    out << "No cliient_id given..."
    return
}

if (!params.client_secret)
{
    out << "No client_secret given..."
    return
}

def client_id = URLEncoder.encode(params.client_id, 'UTF-8')
def client_secret = URLEncoder.encode(params.client_secret, 'UTF-8

HTTPResponse res
URL tokenURL = "http://localhost:9001/rest/oauth/token".toURL()

if (params.basic)
{
   def headers = [Authorization:"Basic ${"${client_id}:${client_se
res = tokenURL.post(deadline: 30, headers: headers, payload:"grant
}
else
{
   res = tokenURL.post(deadline: 30,  payload:"client id=${client

```

요청에 대한 응답은 다음과 같다.

```
{ "access token": "31d9fda8-4694-427b-af57-90853907daf3", "token type
```

3.6. Extension

기타 필요에 따라서 추가적인 인증방식을 만들 수 있다. 보안성 문제로 이래 저래 이야기가 많은 것 같다. 그닥 권장하지 않는 다는 뜻.

4. 보안

OAuth2의 전신인 OAuth는 안전한 access token(oauth_signature 라고 한다.)를 만들기 위해서 아래의 과정을 거쳤다.

1. oauth_signature를 생성하기 위해서
2. 요청 매개변수를 모으고
3. 매개변수를 정규화 해서
4. Signature base string을 만든 다음
5. HMAC-SHA1등의 암호화 방법을 이용해서 암호화된 oauth_signature를 만든다.

엄청 복잡하다.

OAuth2는 그런거 없이 SSL(HTTPS)에 모든 걸 맡긴다. 그리고 access token의 만료일을 지정할 수 없는 OAuth1.0과는 달리 access token의 만료시간을 정할 수 있다.

5. OAuth를 지원하는 인터넷 서비스들

OAuth1.0과 OAuth2.0은 **OAuth**라는 이름만 같고 전혀 호환되지 않는 프로토콜이기 때문에, OAuth 응용 애플리케이션을 개발한다면, 버전을 신경써야 한다. 주요 인터넷 서비스들에 대한 OAuth 지원 버전을 정리했다. 정리한 현재시간은 이다.

Evernote	**1.0**
Yahoo!	1.0a
MySpace	1.0a
Netflix	1.0a
StatusNet	1.0a
Tumblr	1.0a
Vimeo	1.0a
Xero	1.0a
Google App Engine	1.0a
LinkedIn	1.0a, 2.0
Twitter	1.0a, 2.0
Plurk	1.0a, 2.0
Amazon	2.0
PayPal	2.0
Microsoft(Hotmail, Windows Live, Xbox)	2.0
Google	2.0
GitHub	2.0
Dropbox	2.0
Yammer	2.0
Basecamp	2.0
Reddit	2.0

Evernote	1.0
Viadeo	2.0
Facebook	2.0
Instagram	2.0

6. Regacy와의 통합

6.1. 서버/클라이언트 모델과의 통합

7. OAuth2와 OpenAPI

8. Ruby OAuth2 구현

댓글 0건 www.joinc.co.kr 1 로그인

추천 공유 인기순

토론 시작

1등으로 댓글 달기

WWW.JOINC.CO.KR 의 다른 댓글.

IP 자세히 보기
댓글 한 건 • 3달 전•
MJ — MTU, total length는 byte단위로 표시하고, fragment offset값은 (byte단위 offset위치)/8을 한 값이 들어가는 것으로 알고 있습니다.그래서

Constrained Application Protocol
댓글 3건 • 3달 전•
조용찬 — 아하 그렇군요^^ 음.. Resourcce Identifier를 제가생각할땐 URI를 만들수있나하는거였어요

Mesos
댓글 한 건 • 3달 전•
Daeyoung Kim — 2번 항목에서 Mesosphere가 'Datacenter Operation System을 표방하는 소프트웨어'라는 설명이 있는데 제가 알고 있는 사실

Tensorflow 시작
댓글 한 건 • 3달 전•
Sunghyun Park — 안녕하세요, 초보적인 질문 좀 하겠습니다.저는 윈도우10에서 도커를 이용해서 설치를 했습니다.마찬가지로 웹에서 8888을 이

구독 당신의 사이트에 Disqus 추가하기 Disqus 추가 추가 개인정보 처리방침