Bài Tập Lớn ATTT

Xây dựng tài liệu

Trường Tiểu Học Bách Khoa Hà Nội

Viện Công Nghệ Thông Tin Và Truyền Thông

**Mục Lục**

[1. Mục đích ra đời của Open Authentication 2](#_Toc37972383)

[2. Khái Niệm: Oauth2 3](#_Toc37972384)

[3. Lịch sử phát triển 3](#_Toc37972385)

[4. Nguyên Lý Hoạt Động 4](#_Toc37972386)

[4.1 Sơ đồ luồng hoạt động của OAuth2: 4](#_Toc37972387)

[4.2 Oauth 3 "chân" (3-legged) và 2 "chân" (2-legged) 6](#_Toc37972388)

[4.2.1 Oauth 3 "chân" (3-legged) 6](#_Toc37972389)

[4.2.2 Oauth 2 "chân" (2-legged) 7](#_Toc37972390)

[5. Các mô hình ủy quyền. 8](#_Toc37972391)

[5.1 Authorization code 8](#_Toc37972392)

[5.2 Implicit 10](#_Toc37972393)

[5.3 Resource Owner Password Credentials 11](#_Toc37972394)

[5.4 Client Credentials 12](#_Toc37972395)

[6 Tính bảo mật của ủy quyền. Ưu và nhược điểm của ủy quyền. 12](#_Toc37972396)

[1. Tính năng bảo mật của ủy quyền. 12](#_Toc37972397)

[1.1 Tokens 12](#_Toc37972398)

[1.2. Access Token 14](#_Toc37972399)

[1.3. Refresh Token 14](#_Toc37972400)

[1.4. Authorization "code" 15](#_Toc37972401)

[1.5. Redirect URI 15](#_Toc37972402)

[1.6. Tham số "state" 16](#_Toc37972403)

[1.7. Định danh client 16](#_Toc37972404)

[2. Ưu và nhược điểm của ủy quyền. 16](#_Toc37972405)

[2.1. Ưu điểm. 16](#_Toc37972406)

[2.2. Nhược điểm. 19](#_Toc37972407)

# Mục đích ra đời của Open Authentication

Trong thời đại công nghệ số hiện nay, số lượng ứng dụng đang phát triển chóng mặt, bao gồm các ứng dụng giải trí (nghe nhạc, xem phim, game, ...), ứng dụng đọc tin tức (báo điện tử, blog, ...) và còn nhiều loại ứng dụng khác nữa

Mỗi ứng dụng như trên đều có cơ chế quản lý người dùng riêng. Khi đó, việc đăng ký sử dụng nhiều ứng dụng khác nhau thật sự rất tốn thời gian, công sức và dễ gây nhầm lẫn nếu mỗi một ứng dụng lại phải có một username và password khác nhau. Việc đó yêu cầu phải lặp đi lặp lại các bước đăng ký username và password, rồi có thể phải active từng tài khoản với email nữa.

Có rất nhiều ứng dụng, diễn đàn yêu cầu đăng ký/đăng nhập để có thể sử dụng các chức năng của nó. Hướng giải quyết nào cho tình huống như vừa rồi? Tốn chút thời gian để đăng ký đăng nhập cho xong việc hay còn lựa chọn khác. Và một phương pháp cực ký thuận tiện đã ra đời để giải quyết vấn đề.

Nhìn nhận thấy các vấn đề, các "ông lớn" như Twitter, Facebook hay Google đã đàm phán với nhau và đưa ra một chuẩn mới có tên là **Open Authentication**. Nội dung của bài viết này sẽ giới thiệu với các bạn về chuẩn **OAuth** là gì? Cách thức hoạt động thế nào? Điểm mạnh và điểm yếu ra sao? Đồng thời phạm vi giới thiệu sẽ tập trung nhiều hơn vào chuẩn **OAuth** mới nhất hiện nay đó là [**Oauth2**](http://oauth.net/2/).

# Khái Niệm: Oauth2

Về cơ bản thì oauth là một *phương thức chứng thực*, mà nhờ đó một web service hay một application bên thứ 3 có thể **đại diện** cho người dùng để truy cập vào tài nguyên người dùng nằm trên một dịch vụ nào đó.

Từ “Auth” thực ra không phải là viết tắt của từ xác thực - “Authentification”, mà có nguồn gốc từ “Authorization” nghĩa là cấp phép. Authentification – Authorization, đây là 2 từ khá tương tự nhau nhưng lại là 2 từ khác biệt.

“Xác thực” nghĩa là xác nhận một đối tượng là ai. Cụ thể, authenfication là danh từ để chỉ việc xác nhận người sắp thực hiện thao tác là bản thân user mà không phải ai khác.Ví dụ tiêu biểu cho Authentification là mã PIN hay password của thẻ ngân hàng.

Còn Authorization - Cấp phép là gì?

Cấp phép nghĩa là cho phép đối tượng được cấp quyền được làm một việc nào đó, tức là sự chỉ định việc được phép làm.

Về chữ cái “O” trong OAuth, ngay cả trên trang chính thức cũng không có sự giải thích tường minh, tuy nhiên, vì OAuth là open standard nên ta có lẽ có thể hiểu rằng O là viết tắt của Open.

OAuth 2 không chỉ đơn thuần là một giao thức (protocol) mà nó còn là một nền tảng (framework) giúp xây dựng ứng dụng cả ở client và server.

# Lịch sử phát triển

Năm 2006, Twitter phát triển hệ thống OpenID phục vụ cho đăng nhập các ứng dụng trong hệ thống của Twitter, tuy nhiên hệ thống này yêu cầu người dùng phải cung cấp tên đăng nhập và mật khẩu, đây là một điểm yếu. Cũng chính năm này, các ông lớn trong về mạng xã hội như Facebook, Google, Twitter… đã cùng ngồi với nhau để phác thảo ý tưởng về một hệ thống xác thực mới giúp các ứng dụng bên thứ ba có thể tích hợp.

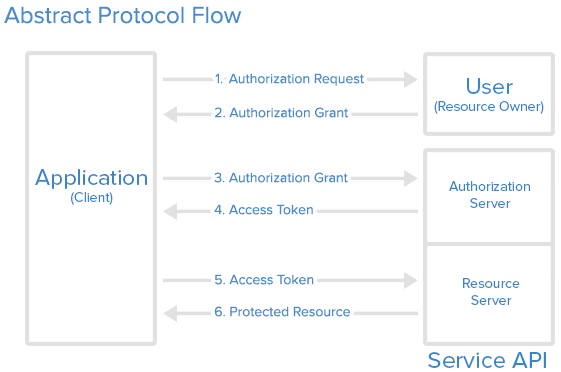
Năm 2008, IETF – tổ chức quản lý tiêu chuẩn mạng Internet (tránh nhầm với IEEE tổ chức quản lý tiêu chuẩn trong lĩnh vực điện và điện tử) đã quyết định hỗ trợ tiêu chuẩn OAuth này và bắt đầu cho xây dựng RFC 1.0.

Năm 2010, IETF phát hành phiên bản chính thức đầu tiên của OAuth 1.0 ([RFC 5849](https://tools.ietf.org/html/rfc5849)).  
Sau đó, lỗi bảo mật nghiêm trọng được phát hiện với tên gọi [Session Fixation](http://hueniverse.com/2009/04/23/explaining-the-oauth-session-fixation-attack/) xảy ra trên OAuth 1.0, cho phép các hacker lừa ứng dụng bên thứ ba trao quyền truy nhập vào tài khoản và dữ liệu người dùng.

Năm 2012, [phiên bản OAuth 2.0](http://tools.ietf.org/html/rfc6749) ra đời, tuy nhiên vẫn còn những lỗi bảo mật chết người các hacker có thể [Hack Facebook](http://homakov.blogspot.co.uk/2013/02/hacking-facebook-with-oauth2-and-chrome.html) thông qua trình duyệt Chrome. Từ đó đến nay chưa có thêm một phiên bản nào khác của OAuth ra đời thêm.

# Nguyên Lý Hoạt Động

## 4.1 Sơ đồ luồng hoạt động của OAuth2:



1. **Application** yêu cầu ủy quyền để truy cập vào **Resource Server** thông qua **User.**
2. Nếu **User** ủy quyền cho yêu cầu trên, **Application** sẽ nhận được giấy ủy quyền từ phía **User** (dưới dạng một token string nào đó chẳng hạn)
3. **Application** gửi thông tin định danh (ID) của mình kèm theo giấy ủy quyền của **User** tới **Authorization Server**
4. Nếu thông tin định danh được xác thực và giấy ủy quyền hợp lệ, **Authorization Server** sẽ trả về cho **Application** access\_token. Đến đây quá trình ủy quyền hoàn tất.
5. Để truy cập vào tài nguyên (resource) từ **Resource Server** và lấy thông tin, **Application** sẽ phải đưa ra access\_token để xác thực.
6. Nếu access\_token hợp lệ, **Resource Server** sẽ trả về dữ liệu của tài nguyên đã được yêu cầu cho **Application**.

Luồng hoạt động thực tế có thể sẽ khác nhau tùy thuộc vào việc ứng dụng sử dụng loại ủy quyền (authorization grant type) nào, trên đây chỉ là ý tưởng chung để thực hiện.

**Ở trên mô hình trên thì:**

**Resource owner** là người dùng

**Client** là application bên thứ 3, ví dụ là 1 ứng dụng sử dụng facebook api

**Authrization Server** và **Resource Server** là những dịch vụ mà phía bên facebook hay twitter phải implement để thực hiện oauth.

Đây chính là điểm cơ bản tạo nên khác biệt giữa Oauth2 và Oauth1, khi "tách biệt" giữa việc chứng thực (authorization) , và việc cung cấp thông tin người dùng (resource) thành 2 server riêng.

**Mô tả đơn giản thì:**

**Vấn đề**

* Tài khoản người dùng Facebook có các tài nguyên, profile cá nhân như số điện thoại, ảnh,..
* Một ứng dụng sử dụng Facebook Api muốn truy cập vào profile cá nhân của tài khoản đó để lập trình

**Cách Oauth giải quyết vấn đề trên**

Như vậy chúng ta có thể hiểu có 3 bên, người dùng, Facebook và nhà lập trình ứng dụng, oauth chính là 1 cách kết nối , để 3 bên nói chuyện với nhau một cách mềm dẻo, sao cho người dùng có thể trao cho bên ứng dụng quyền được truy cập vào các tài nguyên của họ trên facebook.

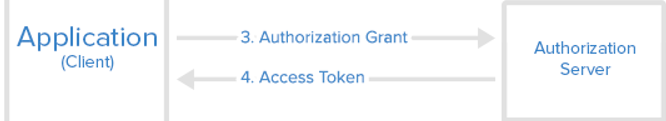
Thật sự thì việc oauth sẽ làm hoàn toàn cực kì tự nhiên:

* Step1: Bên ứng dụng xin quyền truy cập tài nguyên của người dùng
* Step2: Người dùng trả lời có thông qua một hình thức nào đó mà facebook kiểm soát được (ví dụ như một màn hình chứng thực do facebook cung cấp chẳng hạn).
* Step3: Facebook nhận được sự đồng ý đó, và sẽ trao cho bên ứng dụng một cái "chìa khoá" để có thể truy cập vào một số tài nguyên nhất định của người dùng đó.
* Step4: Nhà quản lý ứng dụng sẽ dùng "chìa khoá" đó để truy cập vào tài nguyên của người dùng.

## 4.2 Oauth 3 "chân" (3-legged) và 2 "chân" (2-legged)

Trước khi đi vào cụ thể thì chúng ta phải làm quen với một khái niệm gọi là flow chứng thực Client Credentials . Flow này có thể hiểu đơn giản là Step 1 và 2 ở trên, tức là: làm thế nào để người dùng nói cho facebook được là họ trao quyền cho nhà quản lý ứng dụng access vào profile của họ?

Phần thứ 2 trong mô hình ở trên



phần này chính là **Client Credentials Flow.**

Mục đích chính của flow này là để cho bên cung cấp dịch vụ (facebook) biết được là người dùng "đã" cung cấp cho ứng dụng quyền được truy cập thông tin của họ. Thông tin này có thể được cung cấp tại lúc mà ứng dụng yêu cầu người dùng, hoặc có thể được cung cấp "trước" tại một thời điểm nào đó. Chính sự khác biệt này tạo ra 2 cách làm khác nhau là **3-legged** và **2-legged** oauth.

### 4.2.1 Oauth 3 "chân" (3-legged)

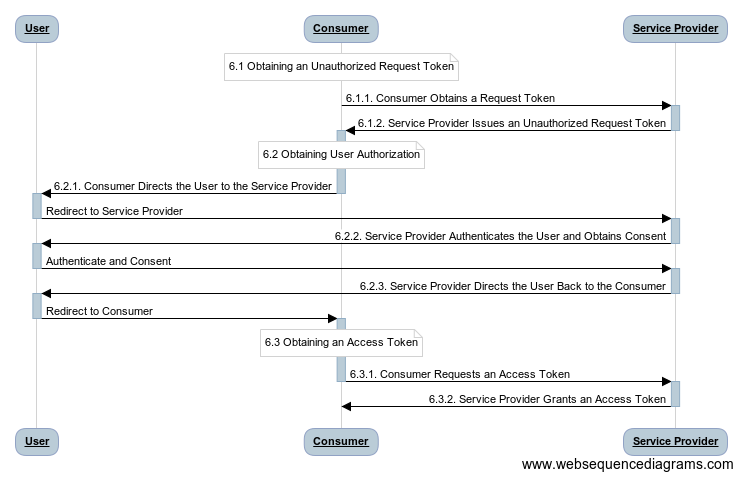
Đây chính là flow phổ thông cho mô hình oauth2 hiện nay. Client Credentials Flow sẽ được thực hiện thông qua việc redirect người dùng đến website của bên cung cấp dịch vụ, người dùng sẽ lựa chọn có hoặc không đồng ý (họ có thể gợi nhớ lại khi họ đăng nhập vào facebook app lần đầu tiên, facebook sẽ cho họ một form để chọn có/không).

Sau đó dựa vào redirect\_uri mà ứng dụng cung cấp trước đó cho facebook, facebook sẽ redirect-back lại dịch vụ của nó, kèm theo một token đã được chứng thực.

Sau cùng ứng dụng sẽ sử dụng token đó như "chìa khoá" để truy cập vào profile cuả người dùng.

Flow chứng thực này yêu cầu cả 3 bên: người dùng, Facebook và nhà lập trình ứng dụng, do đó mà nó được gọi là 3-legged.

Mô hình 3-legged oauth2:



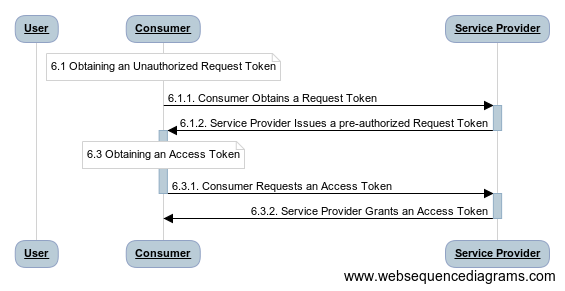
**Mô hình 3-legged oauth2**

### 4.2.2 Oauth 2 "chân" (2-legged)

Vậy nếu ứng dụng đó được facebook tin tưởng "hoàn toàn" thì sao. Ví dụ instagram chẳng hạn =)). Khi đó người dùng phải accept cho ứng dụng truy cập tài nguyên của họ trên facebook là không cần thiết. Hoặc một ví dụ khác là khi làm một widget của google app và người dùng cho phép ứng dụng quyền sử dụng gmail account của họ.

Trong những trường hợp trên thì flow redirect đi redirect lại như ở trên là không cần thiết, khi đó thì chỉ cần ứng dụng nói chuyện trực tiếp với bên cung cấp dịch vụ là họ cần thông tin của người dùng thôi là đủ. Việc cấp quyền của người dùng cho bên ứng dụng có thể được thực hiện tại một thời điểm bất kỳ như lúc bạn cài đặt app của tôi vào desktop chẳng hạn (pre-installed app)

Mô hình này không cần phía người dùng , chỉ cần phía ứng dụng và facebook, do đó mà nó được gọi là 2-legged.



# Các mô hình ủy quyền.

Loại ủy quyền phụ thuộc vào phương thức mà ứng dụng sử dụng để yêu cầu ủy quyền, Oauth2 định nghĩa ra 4 loại:

**Authorization Code**: sử dụng với các server-side Ứng dụng.

**Implicit**: sử dụng với các Mobile App (ứng dụng chạy trên thiết bị của User) hoặc Web App (có thể hiểu là Browser App cũng được, VD: Chrome Extension).

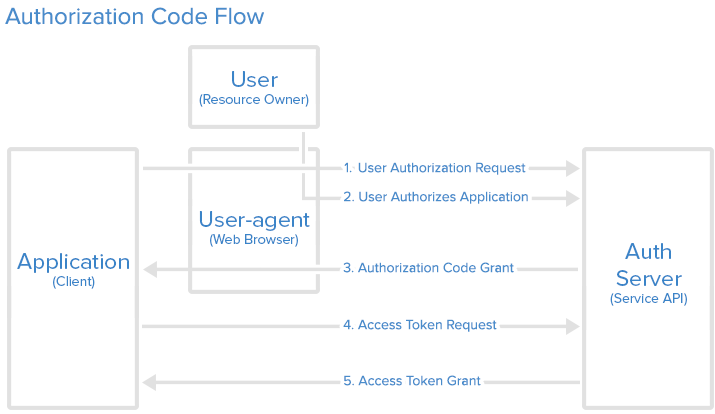
**Resource Owner Password Credentials**: sử dụng với các Trusted Ứng dụng, kiểu như những ứng dụng của chính Service.

**Client Credentials**: sử dụng với các ứng dụng truy cập thông qua API.

## 5.1 Authorization code

Sử dụng cho những ứng dụng có độ tin cậy không cao (ứng dụng của bên thứ 3 yêu cầu truy cập vào hệ thống của người dùng).

The flow includes the following steps:



Ứng dụng gửi một link đến authorization server cho người dùng để bắt đầu quá trình nhận authorization\_code. Link này bao gồm các thông tin cho phép authorization server định danh và phản hồi lại cho ứng dụng.

Người dùng điền thông tin đăng nhập.

Thông tin đăng nhập được gửi đến authorization server.

Authorization server xác thực thông tin của đăng nhập và điều hướng người dùng đến uri của ứng dụng cùng với một mã authorization. Tại đây ứngdụng sẽ phải thực hiện thao tác lưu lại authorization\_code

Ứng dụng gửi yêu cầu đến authorization server cùng mã authorization để nhận access token cùng refresh token (nếu có).User không cần thao tác thêm gì nữa.

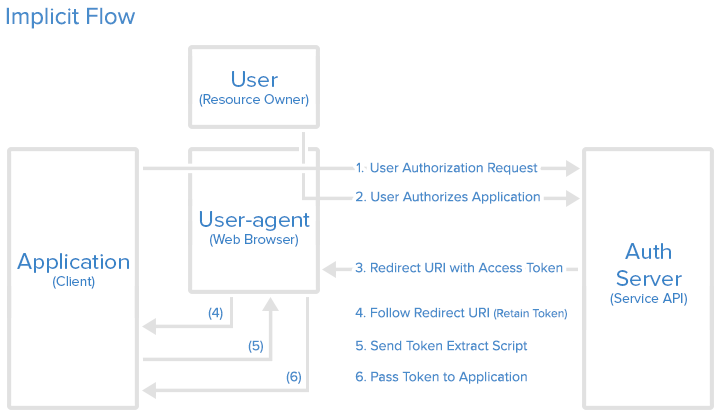
Tới đây Ứng dụng đã được User ủy quyền truy cập. Ứng dụng có thể sử dụng access\_token để truy cập những tài nguyên của dịch vụ mà User cho phép, ví dụ thông tin email, ảnh avatar, ... cho tới khi access\_token hết hạn sử dụng. Nếu phía API hỗ trợ và gửi về thêm cả thông tin refresh\_token thì ứng dụng có thể sử dụng để đổi lấy access\_token mới khi access\_token cũ hết hạn

## Implicit

Thường được sử dụng trong các ứng dụng mobile hay các ứng dụng chạy trên web, nơi mà thông tin bí mật của client không thể lưu trữ bảo mật. Do lo ngại bảo mật, trong flow này, ứng dụng sẽ không nhận authorization\_code từ Authorization server, thay vào đó, Authorization server sẽ trả trực tiếp access token cho ứng dụng.Loại grant này không hỗ trợ refresh\_token.

Ta có thể hiểu đơn giản luồng hoạt động như sau: User nhận được yêu cầu ủy quyền cho Ứng dụng, sau đó Authorization Server truyền thẳng access\_token tới Browser và sau đó truyền lại cho Ứng dụng.

Sơ đồ yêu cầu ủy quyền của ứng dụng:



The flow includes the following steps:

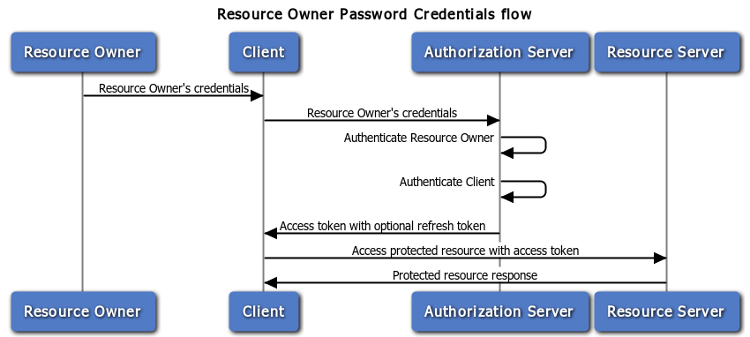
* + Ứng dụng gửi một link đến authorization server cho người dùng để bắt đầu quá trình nhận authorization\_code. Link này bao gồm các thông tin cho phép authorization server định danh và phản hồi lại cho ứng dụng.
  + Người dùng điền thông tin đăng nhập.
  + Thông tin đăng nhập được gửi đến authorization server.
  + Sau đó AuthorizationServer truyền thẳng access\_token ( không có access token cùng refresh ) tới Browser và sau đó truyền lại cho Ứng dụng.

 Ứng dụng có thể sử dụng access\_token để truy cập những tài nguyên của dịch vụ mà User cho phép, ví dụ thông tin email, ảnh avatar, ... cho tới khi access\_token hết hạn sử dụng.

## Resource Owner Password Credentials

Với loại ủy quyền này, User sẽ phải cung cấp thông tin username và password trực tiếp cho Ứng dụng sử dụng để lấy access\_token. Cần lưu ý là loại ủy quyền này chỉ nên sử dụng cho những ứng dụng thực sự được tin tưởng (VD: những ứng dụng của chính Service hay những ứng dụng mặc định của hệ điều hành chẳng hạn).

Sơ đồ ủy quyền của ứng dụng:



Flow ở đây hết sức đơn giản, với một request ta có thể lấy luôn được access\_token:

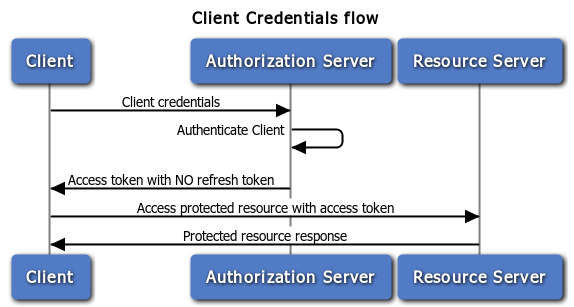
https://OAUTH\_SERVER.DOMAIN/token?grant\_type=password&username=USERNAME&password=PASSWORD&client\_id=CLIENT\_ID

The flow includes the following steps:

* + Ứng dụng đưa ra một form cho phép người dùng nhập thông tin đăng nhập (ví dụ: username/password).
  + Ứng dụng gửi thông tin đăng nhập cùng thông tin định danh của mình lên authorization server. Authorization server xác thực thông tin, trả lại access token và refresh token (nếu có).
  + Ứng dụng sử dụng access token truy cập tài nguyên trên resource server.

## Client Credentials

Loại ủy quyền này phục vụ cho việc truy cập vào chính thông tin tài khoản của **Ứng dụng** tại Service. Có thể hiểu nôm na là **Ứng dụng** mong muốn thay đổi thông tin description hoặc redirect\_uri hay lấy thông tin của chính **Ứng dụng** thông qua API. Loại ủy quyền này không có sự tham gia của \*User.



The flow includes the following steps:

* Ứng dụng gửi thông tin xin cấp quyền truy cập tài nguyên đến authorization server.
* Authorization server sau xác thực thông tin gửi đến từ ứng dụng. Sau đó authorization server sẽ gửi access token (không đi kèm refresh token) đến ứng dụng.
* Ứng dụng có thể sử dụng mã này để truy cập tài nguyên của server.

# Tính bảo mật của ủy quyền. Ưu và nhược điểm của ủy quyền.

## Tính năng bảo mật của ủy quyền.

Đây là một số tính năng bảo mật đã được tích hợp trong giao thức OAuth 2.0 để giảm thiểu các cuộc tấn công và các vấn đề liên quan đến bảo mật.

### Tokens

OAuth sử dụng rộng rãi nhiều loại token (access token, refresh token, "mã" ủy quyền- authorization "codes"). Nội dung thông tin của một token có thể được trình bày theo hai cách như sau:

Xử lý (Handle) : Một handle là một tham chiếu đến một số cấu trúc dữ liệu nội bộ trong máy chủ ủy quyền (authorization server); cấu trúc dữ liệu nội bộ chứa các thuộc tính của token, chẳng hạn như id người dùng (UID), phạm vi, v.v. Các handle cho phép thu hồi đơn giản và không yêu cầu cơ chế mã hóa để bảo vệ nội dung token khỏi bị sửa đổi. Mặt khác, các handle yêu cầu sự giao tiếp giữa thực thể phát hành và tiêu thụ (ví dụ: máy chủ ủy quyền và máy chủ tài nguyên) để xác thực token và lấy dữ liệu token-bound. Giao tiếp này có thể có tác động tiêu cực đến hiệu suất và khả năng mở rộng nếu cả hai thực thể ở trên các hệ thống khác nhau. Do đó, handle thường được sử dụng nếu thực thể phát hành và tiêu thụ là như nhau. Một 'handle' token thường được gọi là một token mã hóa mờ đục vì máy chủ tài nguyên không cần phải giải nghĩa token một cách trực tiếp; nó chỉ đơn giản là sử dụng token.

Xác nhận (Assertion - còn gọi là mã thông báo độc lập - self-contained token): Một assertion là một token có thể phân tích cú pháp. Một assertion thường có thời lượng, có đối tượng và được chữ ký số để đảm bảo tính toàn vẹn dữ liệu và xác thực nguồn gốc. Nó chứa thông tin về người dùng và khách hàng. Ví dụ về các định dạng assertion là các Security Assertion Markup Language (SAML) [OASIS.saml-core-2.0-os] và thẻ Kerberos [RFC4120]. Các assertion thường có thể được xác nhận trực tiếp và được sử dụng bởi một máy chủ tài nguyên mà không cần tương tác với máy chủ ủy quyền. Điều này dẫn đến hiệu suất và khả năng mở rộng tốt hơn trong các triển khai nơi các thực thể phát hành và tiêu thụ ở trên các hệ thống khác nhau. Việc assertion thực hiện thu hồi token khó khăn hơn so với handle.

Các token có thể được sử dụng theo hai cách để gọi yêu cầu trên các máy chủ tài nguyên, đó là:

* bearer token: là token có thể được sử dụng bởi bất kỳ client nào đã nhận được token (ví dụ: [RFC6750]). Vì chỉ sở hữu là đủ để sử dụng token, điều quan trọng là giao tiếp giữa các điểm cuối phải được bảo mật để đảm bảo rằng chỉ các điểm cuối được ủy quyền mới có thể nắm bắt token. bearer token thuận tiện cho các ứng dụng khách, vì nó không yêu cầu họ làm bất cứ điều gì để sử dụng chúng (chẳng hạn như bằng chứng nhận dạng). bearer token có đặc điểm tương tự như cookie đăng nhập một lần (SSO) được sử dụng trong trình duyệt.
* proof token: là token mà chỉ một khách hàng cụ thể có thể sử dụng. Mỗi lần sử dụng token yêu cầu khách hàng thực hiện một số hành động chứng minh rằng đó là người dùng được ủy quyền của token. Ví dụ về điều này là các MAC-type access tokens, (ví dụ: [OAuth-HTTP-MAC]).

#### 1.1.1. Scope (Phạm vi)

Scope thể hiện ủy quyền truy cập được liên kết với một token cụ thể đối với các máy chủ tài nguyên, tài nguyên và phương thức trên các tài nguyên đó. Scope là cách để OAuth quản lý rõ ràng sức mạnh liên quan đến một access token. Một scope có thể được kiểm soát bởi máy chủ ủy quyền và / hoặc người dùng cuối nhằm hạn chế quyền truy cập vào tài nguyên cho các client OAuth mà các bên này cho rằng kém an toàn hoặc đáng tin cậy. Theo tùy chọn, client có thể yêu cầu phạm vi áp dụng cho token nhưng chỉ với scope nhỏ hơn mức được cấp, ví dụ, để giảm tác động nguy hại nếu token này được gửi qua các kênh không an toàn. Một scope thường được bổ sung bởi một hạn chế về thời gian tồn tại của token.

#### 1.1.2. Access Token giới hạn thời gian tồn tại

Tham số giao thức "expires\_in" cho phép máy chủ ủy quyền (dựa trên chính sách của nó hoặc nhân danh người dùng cuối) để giới hạn thời gian tồn tại của token truy cập và truyền thông tin này cho client. Cơ chế này có thể được sử dụng để phát hành token ngắn hạn cho các client OAuth mà máy chủ ủy quyền cho là kém an toàn hơn hoặc gửi token qua các kênh không an toàn.

### 1.2. Access Token

Access token được sử dụng bởi client để truy cập tài nguyên. Access token  thường có tuổi thọ ngắn (phút hoặc giờ) bao gồm vòng đời phiên thông thường. Access token có thể được làm mới thông qua việc sử dụng một refresh token.  Tuổi thọ ngắn của access token, kết hợp với việc sử dụng một refresh token, cho phép khả năng thu hồi thụ động ủy quyền truy cập khi hết hạn access token hiện tại.

### 1.3. Refresh Token

Một refresh token thể hiện sự ủy quyền lâu dài của một client nhất định để truy cập tài nguyên thay mặt cho chủ sở hữu tài nguyên. Các token như vậy chỉ được trao đổi giữa máy khách và máy chủ ủy quyền. Client sử dụng loại token này để có được ("làm mới") access token mới được sử dụng cho các yêu cầu máy chủ tài nguyên.

Một refresh token, cùng với access token giới hạn thời gian tồn tại, có thể được sử dụng để cấp quyền truy cập lâu hơn cho các tài nguyên mà không liên quan đến ủy quyền của người dùng cuối. Điều này mang lại lợi thế khi máy chủ tài nguyên và máy chủ ủy quyền không phải là cùng một thực thể, ví dụ: trong môi trường phân tán, vì refresh token luôn được trao đổi tại máy chủ ủy quyền. Máy chủ ủy quyền có thể thu hồi refresh token bất cứ lúc nào, khiến quyền truy cập được cấp sẽ bị thu hồi sau khi access token hiện tại hết hạn. Do đó, access token giới hạn thời gian tồn tại rất quan trọng nếu việc thu hồi kịp thời được ưu tiên.

Refresh token cũng là một bí mật được ràng buộc với định danh client và phiên bản client ban đầu yêu cầu ủy quyền; refresh token cũng đại diện cho cấp chủ sở hữu tài nguyên ban đầu. Điều này được đảm bảo bởi quy trình ủy quyền như sau:

1. Chủ sở hữu tài nguyên và tác nhân người dùng cung cấp "mã" ủy quyền (authorization "code") cho phiên bản client ban đầu yêu cầu ủy quyền.

2. Client sử dụng nó ngay lập tức trong các giao tiếp cấp vận chuyển an toàn đến máy chủ ủy quyền và sau đó lưu trữ một cách an toàn refresh token tồn tại lâu dài.

3. Client luôn sử dụng refresh token trong các giao tiếp cấp vận chuyển an toàn đến máy chủ ủy quyền để nhận một access token.

### 1.4. Authorization "code"

"Mã" ủy quyền biểu thị kết quả trung gian của quy trình ủy quyền người dùng cuối thành công và được client sử dụng để có được access token và refresh token. "Mã" ủy quyền được gửi đến URI chuyển hướng của client thay vì token cho hai mục đích:

1. Các luồng dựa trên trình duyệt hiển thị các tham số giao thức cho những kẻ tấn công tiềm năng thông qua các tham số truy vấn URI (tham chiếu HTTP), bộ đệm của trình duyệt hoặc các mục nhập tệp nhật ký và có thể được phát lại. Để giảm mối đe dọa này, "mã" ủy quyền tồn tại trong thời gian ngắn được thông qua thay vì token và đổi lấy các token qua kết nối trực tiếp an toàn hơn giữa máy khách và máy chủ ủy quyền.

2. Việc xác thực client trong khi yêu cầu trực tiếp giữa máy khách và máy chủ ủy quyền sẽ đơn giản hơn nhiều so với yêu cầu ủy quyền gián tiếp. Sau này sẽ yêu cầu chữ ký số.

### 1.5. Redirect URI

URI chuyển hướng giúp phát hiện các client độc hại và ngăn chặn các cuộc tấn công lừa đảo từ các client đang cố lừa người dùng tin rằng kẻ lừa đảo là client. Giá trị của URI chuyển hướng thực tế được sử dụng trong yêu cầu ủy quyền phải được trình bày và được xác minh khi "mã" ủy quyền được trao đổi với các token. Điều này giúp ngăn chặn các cuộc tấn công trong đó "mã" ủy quyền được tiết lộ thông qua các bộ chuyển hướng và client giả mạo ứng dụng web. Máy chủ ủy quyền phải yêu cầu các public client và confidential client sử dụng loại cấp quyền ngầm để đăng ký trước URI chuyển hướng của họ và xác thực đối với URI chuyển hướng đã đăng ký trong yêu cầu ủy quyền.

### 1.6. Tham số "state"

Tham số "state" được sử dụng để liên kết các yêu cầu và lời gọi lại để ngăn chặn các cuộc tấn công giả mạo yêu cầu giữa các trang web trong đó kẻ tấn công cho phép truy cập vào tài nguyên của chính mình và sau đó lừa người dùng tiếp tục một chuyển hướng với token của kẻ tấn công. Tham số này phải liên kết với trạng thái được xác thực trong tác nhân người dùng và, theo thông số OAuth cốt lõi, tác nhân người dùng phải có khả năng giữ nó ở vị trí chỉ có thể truy cập bởi client và tác nhân người dùng, nghĩa là được bảo vệ bởi chính sách cùng nguồn gốc.

### 1.7. Định danh client

Các giao thức xác thực thường không tính đến danh tính của thành phần phần mềm thay mặt cho người dùng cuối. OAuth thực hiện điều này để tăng mức độ bảo mật trong các tình huống ủy quyền và vì client sẽ có thể hành động mà không cần sự có mặt của người dùng.

OAuth sử dụng định danh client để đối chiếu các yêu cầu liên quan đến cùng một người khởi tạo, chẳng hạn như:

* một quy trình ủy quyền của người dùng cuối và yêu cầu tương ứng trên điểm cuối của token để trao đổi "mã" ủy quyền cho các token
* ủy quyền ban đầu và phát hành một token của người dùng cuối cho một client cụ thể và các yêu cầu tiếp theo của client này để nhận token mà không cần sự đồng ý của người dùng (tự động xử lý ủy quyền lặp lại)

Mã định danh này cũng có thể được sử dụng bởi máy chủ ủy quyền để hiển thị thông tin đăng ký có liên quan cho người dùng khi yêu cầu đồng ý cho một phạm vi được yêu cầu bởi một client cụ thể. Mã định danh client có thể được sử dụng để giới hạn số lượng yêu cầu cho một client cụ thể hoặc tính phí cho client theo yêu cầu. Ngoài ra, có thể hữu ích để phân biệt quyền truy cập của các client khác nhau.

## 2. Ưu và nhược điểm của ủy quyền.

### 2.1. Ưu điểm.

**Bảo vệ API**

Hãy tưởng tượng, một nhà phát triển cung cấp API cho một số application (có thể là RESTful) Để truy cập vào nó một cách an toàn, có một cách là sử dụng xác thực cơ bản, trong đó username và password được gửi bằng Base64- nhưng không được mã hóa hoặc băm nhỏ và SSL được sử dụng để bảo mật việc truyền dữ liệu. Hạn chế ở đây là, trong loại xác thực này, người dùng phải nhập cả username và password của mình.

Vậy điều này có thể cải thiện bằng cách nào? Để bảo mật với ủy quyền truy cập (OAuth), mỗi khi yêu cầu được gửi tới API, thay vì username và password, một access token được gửi. Application (client) nhận được token này trước khi thực hiện các yêu cầu, thay mặt người dùng sử dụng API.

Chẳng hạn, một application đang sử dụng API, bắt đầu hoạt động sai và sử dụng nó không đúng cách. Thì tất cả những gì phải làm là thu hồi access token, làm cho tất cả các yêu cầu trong tương lai không hợp lệ (nếu chúng được thực hiện với cùng một token).

Một điều quan trọng khác cần lưu ý là access token có thể đặt theo phạm vi truy cập, được gọi là scope. Vì vậy, một application có thể không có quyền truy cập API đầy đủ khi sử dụng access token. Ngược lại, nếu application đang sử dụng username và password thay vì access token, và các thông tin đăng nhập này đã bị đánh cắp, toàn bộ API có thể bị sử dụng sai cho đến khi ngăn chặn được việc tấn công tài khoản.

**Các ứng dụng doanh nghiệp nội bộ (Internal enterprise applications)**

Một ví dụ khác, một công ty sử dụng nội bộ một bộ ứng dụng doanh nghiệp. Các nhân viên trong công ty phải nhập username và password theo cách thủ công trong mỗi ứng dụng riêng biệt. Điều này đặt ra rủi ro an ninh trong một số lĩnh vực; cần một cơ sở dữ liệu để lưu trữ các password, các password được gửi nhiều lần...

Với ủy quyền truy cập (OAuth 2.0), những gì xảy ra là có một application mà người dùng đã đăng nhập bằng username và password của mình (nhà cung cấp dịch vụ dựa trên OAuth 2.0) và trong tất cả các application khác, người dùng chỉ cần chuyển hướng đến nhà cung cấp- nơi người dùng đã đăng nhập và xác nhận rằng anh ta muốn được ủy quyền. Bằng cách này, thay vì lưu trữ mật khẩu, các application này đang lưu trữ token cho người dùng. Lợi ích của điều này là nếu như mật khẩu bị đánh cắp, người dùng phải đặt lại mật khẩu của mình thì khi token bị đặt cắp, token bị thu hồi (không hợp lệ).

Một ví dụ liên quan khác là bảo mật trong các công ty BYOD(bring your own device). Nếu người dùng mang điện thoại thông minh của mình và sử dụng ứng dụng nội bộ của công ty trên đó. Khi điện thoại thông minh này bị đánh cắp hoặc bị xâm phạm, password của người dùng sẽ không bị lộ và chỉ các token đang sử dụng sẽ bị vô hiệu hóa. Người dùng có thể nhận được một thiết bị mới và khi anh ta ủy quyền cho nó, anh ta sẽ nhận được một token mới.

***Tích hợp dịch vụ và ủy quyền***

Trong trường hợp người dùng có tài khoản trên dịch vụ chia sẻ ảnh, và cả dịch vụ đặt hàng in.

Nếu hai dịch vụ này có tích hợp OAuth2.0 giữa chúng, người dùng có thể ủy quyền cho dịch vụ in để truy cập tài nguyên từ dịch vụ chia sẻ ảnh. Ví dụ: người dùng sẽ đăng nhập vào dịch vụ in và yêu cầu tích hợp, sau đó người dùng được chuyển hướng trở lại dịch vụ chia sẻ ảnh, nơi người dùng chấp thuận yêu cầu ủy quyền.

Từ thời điểm này, dịch vụ in có một access token thay mặt cho người dùng và có thể nhận biết ảnh nào mà người dùng đã tải lên. Và người dùng có thể truy cập ảnh trên dịch vụ in mà không cần tải lại chúng ở đó, nhưng được truy xuất thông qua dịch vụ chia sẻ ảnh.

Ưu điểm ở đây là người dùng không cẩn cung cấp username và password cho dịch vụ in, nhưng thay vào đó, dịch vụ in đã nhận được một access token khi được người dùng ủy quyền.

Điều này cho thấy một trong những thế mạnh chính của việc sử dụng ủy quyền truy cập. Với ủy quyền truy cập (OAuth 2.0), bạn có thể cấp quyền truy cập vào dữ liệu / tài nguyên của mình trên một số dịch vụ cho một dịch vụ khác và dễ dàng thu hồi quyền truy cập này khi bạn đổi ý. Điều này áp dụng ở mọi nơi, từ các ứng dụng doanh nghiệp nội bộ đến các dịch vụ khác nhau như ứng dụng truyền thông xã hội.

***Nhận dạng liên kết***

Một ưu điểm khác của ủy quyền là nhận dạng liên kết. Với nhận dạng liên kết, nhận dạng số và thông tin của một người (như là email, tên, họ và giới tính) có thể được liên kết giữa một số dịch vụ riêng biệt.

Hai ví dụ trước đã cho thấy điều này, nhưng hãy xem một ví dụ khác; Người dùng có thể đăng nhập vào một nhà cung cấp OAuth 2.0 (ví dụ: Facebook hoặc LinkedIn) và sau đó đăng nhập vào các dịch vụ web khác thông qua nhà cung cấp này mà không cần nhập username và password mới. Nếu người dùng muốn để lại nhận xét trên một số bài đăng trên blog và cần được xác định để nhận xét được chấp nhận.

Trường hợp này còn được gọi là sử dụng OAuth 2.0 để xác thực. Application (ví dụ: nền tảng blog) chỉ yêu cầu thông tin của người dùng sau khi được ủy quyền mà không đòi hỏi gì hơn.

***Giám sát dịch vụ dễ dàng hơn***

Cuối cùng nhưng không kém phần quan trọng, doanh nghiệp có thể theo dõi và giám sát dễ dàng hơn access token nào đang thực hiện yêu cầu gì; dựa trên điều này, họ có thể thực hiện các tính toán và hiểu rõ hơn về dịch vụ nào được khách hàng sử dụng thường xuyên hơn và tối ưu hóa.

### 2.2. Nhược điểm.

Đối với OAuth, nếu bạn thêm nhiều tiện ích mở rộng ở phần cuối trong đặc tả, nó sẽ tạo ra một loạt các triển khai không thể tương tác, có nghĩa là bạn phải viết các đoạn mã riêng cho Facebook, Google, v.v.

Nếu các trang web yêu thích của bạn được kết nối với trung tâm và tài khoản trung tâm bị hack, thì nó sẽ dẫn đến các ảnh hưởng nghiêm trọng trên một số trang web thay vì chỉ một.

Chẳng hạn, khi Facebook, Google, Twitter trở thành một điểm duy nhất nếu có tình huống xảy ra . Nếu bạn mất quyền truy cập vào bất kỳ các tài khoản, bạn sẽ mất quyền truy cập vào tất cả các trang web dựa trên tài khoản đó.

Mặc dù vẫn còn tồn tại lỗ hổng bảo mật khi duyệt web ở chrome,  nhưng người dùng có thể hạn chế bằng cách sử dụng HTTPS thay thì HTTP thông thường.