|  |
| --- |
|  |
| |  | | --- | | OAuth2.0 Security Framework Implementation | |

Contents

[Version History 2](#_Toc45875354)

[Purpose 3](#_Toc45875355)

[Scope 3](#_Toc45875356)

[Guide 3](#_Toc45875357)

[API Provider Perspective: 3](#_Toc45875358)

[OAuth2.0 Security Framework Implementation 4](#_Toc45875359)

[1. The Concept of OAuth2.0 4](#_Toc45875360)

[2. OAuth Actors 4](#_Toc45875361)

[3. OAuth Endpoints 5](#_Toc45875362)

[4. OAuth Token 5](#_Toc45875363)

[5. OAuth Client Types: Confidential Clients and Public Clients 5](#_Toc45875364)

[5.1 Confidential Clients 6](#_Toc45875365)

[5.2 Public Clients 6](#_Toc45875366)

[6. OAuth Flows/Grant Types 6](#_Toc45875367)

[6.1 Authorization Code Flow (Three Legged OAuth/Access Code) 7](#_Toc45875368)

[Sequential Flow: 7](#_Toc45875369)

[6.2 Implicit Flow 10](#_Toc45875370)

[Sequential Flow: 11](#_Toc45875371)

[6.3 Password Flow/Resource Owner Password Credentials 13](#_Toc45875372)

[Sequential Flow: 13](#_Toc45875373)

[6.4 Application Flow/Client Credentials Flow 14](#_Toc45875374)

[Sequential Flow 15](#_Toc45875375)

# Version History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date Created | Author | Version | Approved by: | Comments |
| 07/17/2020 | Memar Demeke | 1.0 |  | Initial version |
| 07/22/202 | Memar Demeke | 2.0 |  | Second version |
|  |  |  |  |  |
|  |  |  |  |  |

# 

# Purpose

OAuth 2.0 provides a consistent, flexible identity and policy architecture to combat this password anti-pattern. OAuth 2.0 is becoming the default security framework to secure access to APIs. The main purpose of this document is to present a detail discussion of the concept OAuth2.0, OAuth actors, OAuth client types and the different grant types. It also discusses its implementation in IBM API Connect. It enables apps to obtain limited access (scopes) to a user's data without giving away a user's password.

# Scope

This guide will walk through the concept of OAuth2.0, flavors of OAuth2.0, OAuth actors interacting and involved in the generation of tokens and its implementation in API Connect

# Guide

### API Provider Perspective:

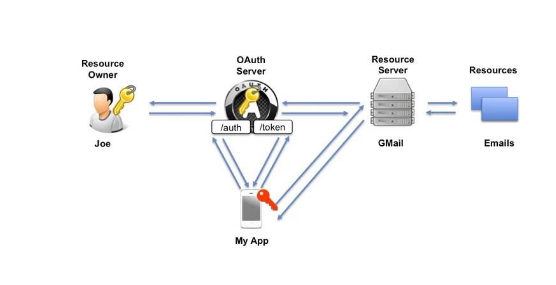
# OAuth2.0 Security Framework Implementation

## **The Concept of OAuth2.0**

* OAuth is a token-based authentication and authorization protocol allows third-party websites, cloud or applications to access user data without requiring the user to share personal information. In API Connect, you can secure an API with OAuth 2.0 security framework with different grant type.
* OAuth2.0 is a standard for delegating authorization for accessing resources via HTTP; the resource owner provide access rights to applications (mobile app, cloud app or Web app) without sharing the password. Instead, token handed to the app.
* The token represents the access rights for a subset of the data; for a short time frame. The generated token can be Authorization code, Access token or Refresh token.

## **OAuth Actors**

High level OAuth 2.0 actor’s interactions diagram



OAuth is designed for distributed system that consists of several actors with distinct roles.

1. **Authorization Provider**(OAuth server or Authorization server): It has:

* An **authentication** component such as login page and an identity provider
* A component (**consent server**) for requesting the authenticated user’s consent for the delegation of access rights to the client
* A **token management infrastructure** such as a database

1. **Resource Provider** (usually set of APIs): The resource provider makes a protected resource (data or a service) available. It allow APPs authenticated and authorized users can have access to data by checking the validity of the provided access token. Only requests with valid access token will be allowed to access resources.
2. **Resource Owner (User):** The resources owner is the owner of the protected resource. The resource owner delegates his access rights to the third party app. Through this delegation the resource owner allows the third party to access data.
3. **Client (Cloud app, web-app or mobile app):** The client is an application that attempts to access a protected resources from resources server in behalf of the resource owner.

## **OAuth Endpoints**

An OAuth endpoint is a service with a defined behavior and address. In a typical OAuth scenario, three endpoints are used which interact in a defined manner.

1. **Authorization Endpoint**: It authenticates the resource owner using any typical authentication method such as username and password. It sends a confirmation of the authentication and of access delegation to the redirect endpoint. This confirmation is called Authorization Code.
2. **Token Endpoint**: This endpoint produces OAuth tokens-access tokens and refresh tokens and returns them in the form of a JSON object to the requester. Token endpoints can only be called by clients that are known to OAuth provider. Clients have to register with the OAuth provider in order to get Client ID and Client Secret.
3. **Redirect Endpoint:** The response generated by the authorization endpoint is sent to the redirect endpoint. This endpoint is not called directly, but indirectly by a HTTP-redirect command (HTTP status code 302). The client needs the authorization code to request access token from the token endpoint.

## **OAuth Token**

OAuth tokens are identifiers used in OAuth flows. The holder of the token has the access rights associated with the token.

1. **Access Token:** used by client to access resources. The client sends the access token to the resources server when accessing the protected resource. Access tokens are bearer tokens. The identity of the holder of the token is not checked any further.
2. **Refresh Token:** It is requested to get a new access token after the access token is expired. It has a period of validity longer than that of the access token (e.g. 30 days).
3. **Authorization Code:** The OAuth provider creates an authorization code and sends it to the client after successfully authenticating the resource owner and getting the consent of the resource owner for delegating the access. Authorization code is thus a confirmation of successful authentication and of the resource owner’s consent. Its validity is limited to couple of minutes.

## **OAuth Client Types: Confidential Clients and Public Clients**

There are two types of clients defined by OAuth security protocol. These are: Confidential client and Public clients.

### **5.1** **Confidential Clients**

* Confidential clients are applications that are able to securely authenticate with the authorization server, for example being able to keep their registered client secret safe.
* Confidential applications can hold credentials in a secure way without exposing them to unauthorized parties. They require a trusted backend server to store the secret(s).
* Public clients are unable to use registered client secrets, such as applications running in a browser or on a mobile device.
* A confidential scheme is suitable when an application is capable of maintaining the secrecy of the client secret. This is usually the case when an application runs in a browser and accesses its own server when obtaining OAuth access tokens.
* Uses **Application Flow, Password flow and Access code flow**.

### **5.2 Public Clients**

* A public scheme is suitable when an application is incapable of maintaining the secrecy of the client secret. This is usually the case when the application is native on a computer or mobile where the secret would have to be stored on the user's device, likely inside the source code of the application. As such, these schemes do not make use of the client secret.
* Public applications **cannot** hold credentials securely.
* Uses **Implicit Flow, Password flow and Access code flow**.

## **OAuth Flows/Grant Types**

OAuth flow is an interaction between OAuth actors and components. It is described as a sequence of requests and responses exchanged among the OAuth actors and endpoints. During this interactions, credentials and tokens are requested, created, delivered, verified, updated and revoked. There are four OAuth flows identified in the standard.

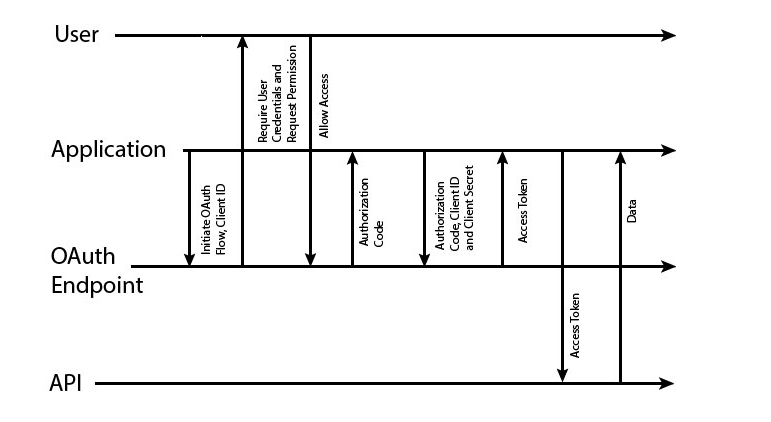
1. Authorization Code Flow (Three Legged OAuth)
2. Implicit Flow
3. Resource Owner Password Credential Flow
4. Client Credential Flow (Two Legged OAuth)-the client is also the resource owner.

**OAuth Security definition types**

|  |  |
| --- | --- |
| **OAuth Flow in API Connect** | **Corresponding OAuth Grant types** |
| Access Code | Authorization Code/Three Legged OAuth |
| Implicit | Implicit |
| Application | Client Credentials/Two legged OAuth |
| Password | Resource Owner Password Credentials |

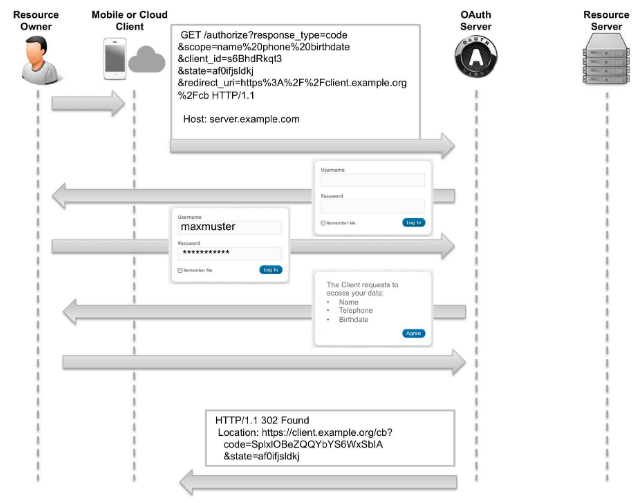
### **6.1 Authorization Code Flow (Three Legged OAuth/Access Code)**

* It enables checking the identity of the three involved actors: **OAuth server, resource owner and client.**
* The Authorization server authenticates the resource owner by the username and password that are provided interactively by the resources owner.
* The Authorization server authenticates the client by its Client ID and Client Secret which are transmitted in the HTTP headers.
* The client authenticates the OAuth server by checking its certificates and URL.

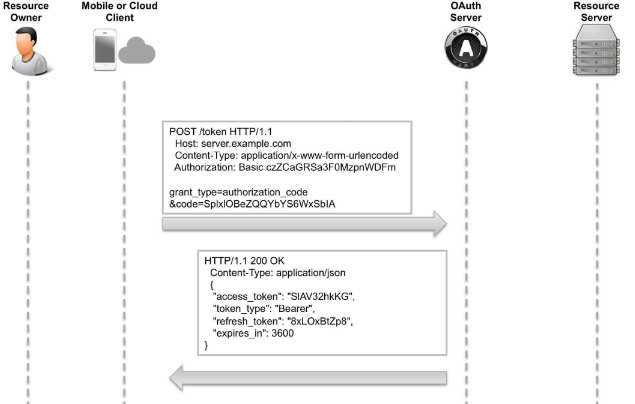


### **Sequential Flow:**

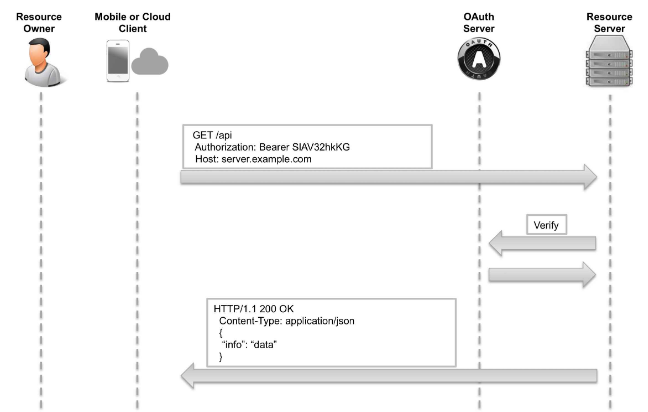
1. **Get the authorization code**: the client requests an authorization code from the authorization endpoint of the Oauth server.



1. **Get the token:** The client requests a token from the token endpoint of the OAuth server/ provider API using the authorization code.

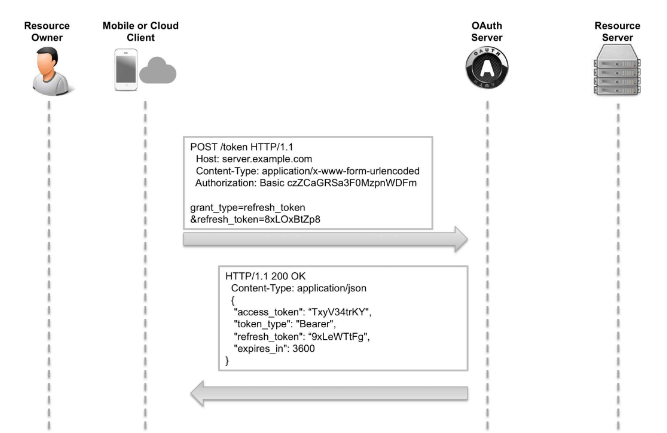


1. **Access the protected resource:** The client access the resource from the resource provider using the access token.



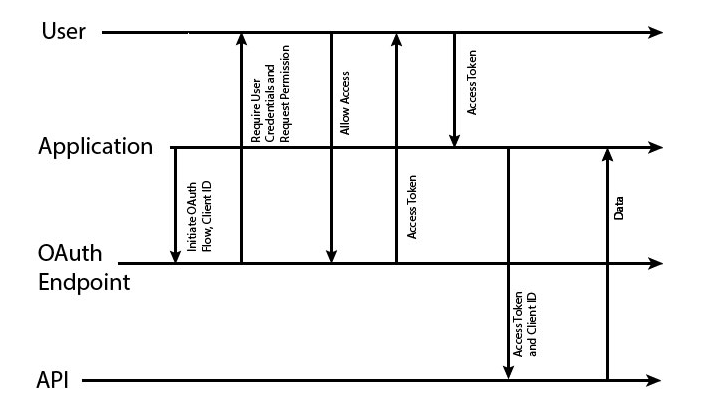
**Note:** **Refresh Token Endpoint**

* In case the access token has expired and the client has a valid refresh token, a new access token can be requested from the token endpoint using HTTP POST. The client provides the refresh token instead of authorization code. Using this flow, the resource owner does not need to authenticate again.
* The client needs to authenticate using the base64 encoded ClientID: ClientSecret in the Authorization HTTP header field.
* The two query parameters to the token endpoint are: *grant\_type*, which is set to the value of refresh\_token and *refresh\_token*, which contains the actual refresh token.
* The successful response contains a JSON object with a new access token, a refresh token and the new validity period of the access token.



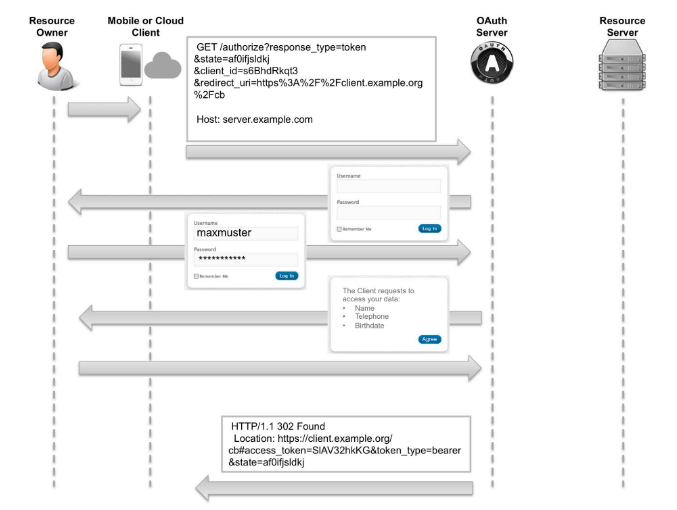
### **6.2 Implicit Flow**

The implicit flow is a simplification of the authorization flow. No refresh token are delivered in this flow. This flow is used for clients that do not have the possibility to securely store the refresh token and Client ID and Client Secret.

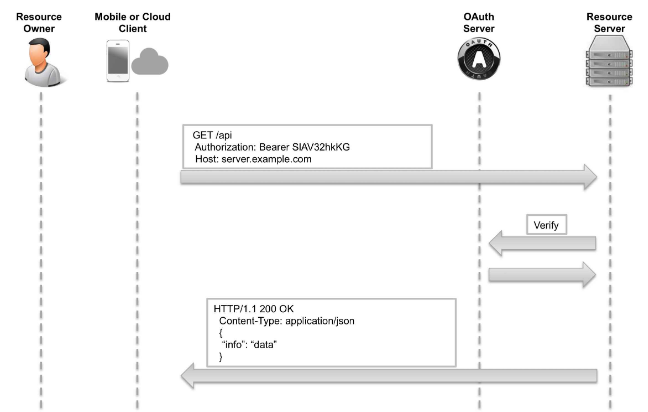


### **Sequential Flow:**

1. **Get Token:** The client requests a token from the authorization endpoint of the OAuth provider.

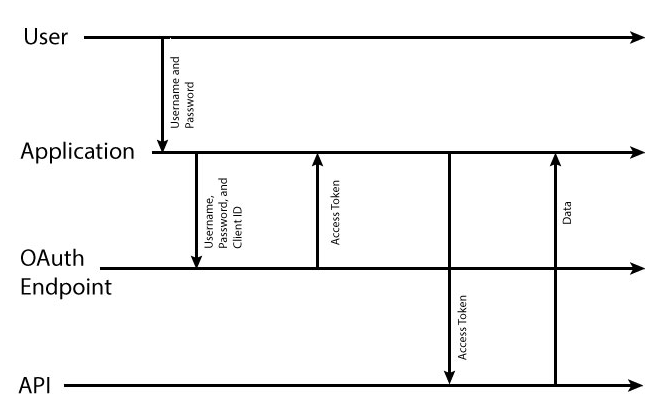


1. **Access protected resources**: The client accesses the resources from the resources provider using the access toke. No token endpoint is involved in this flow. The resource access step in this implicit flow is similar to the resource access in the authorization code flow.



### **6.3 Password Flow/Resource Owner Password Credentials**

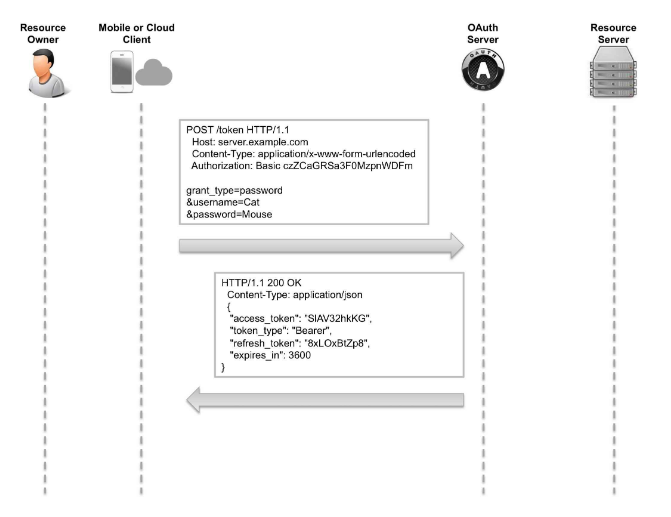
* The pre-condition for this flow is that the resource owner is willing to share his/her credentials with the client, so the client can use the username and password of the resource owner to access resource from resource server.
* In the password flow scheme, the user provides the application with a user name and password that can be used to access the user's data.
* Following this, the client will directly contact the server to request an access token. In this case, trust must exist between user and application because the user's password is revealed to the application.



### **Sequential Flow:**

The resource owner password credentials flow contains the following steps.

1. **Get a token**: the client requests a token from the token endpoint of the OAuth provider

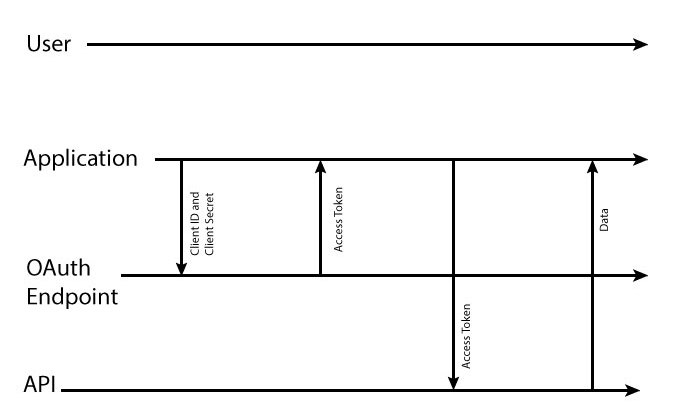


1. **Access the protected resource**:the client accesses the resource from the resource provider using the access token.

**Note:** No authorization endpoint is involved in this flow as the resource owner provides the credentials directly to the client.

### **6.4 Application Flow/Client Credentials Flow**

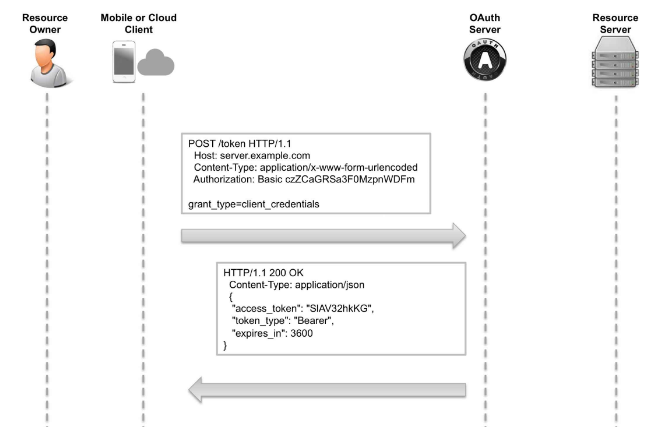
* In this flow the client is identical to the resource owner; a scenario without a resource owner.
* Since no resource owner is present, client credentials are sufficient and the resource owner is not required to authenticate. The Client is identical to the resource owner.
* In the application flow scheme, the user is not required to provide authorization at any stage. Instead, the application uses its client secret to obtain an access token.
* The client need to offer secure storage for ClientID, ClientSecret and access token.



### **Sequential Flow**

The client credentials flow has the following steps

1. **Get Token:** the client sends a request to get access token to the Authorization provider server.



1. **Access the protected resource:** the client accesses the resource from the resource provider using the access token. The resource access step in this flow is similar to the resource access in the authorization code flow.

**Note:** The authorization endpoint is not required in this flow.