

# OAuth2 Overview

OIDC primer - a course on OpenID Connect

Credits: Roland Hedberg, Ioannis Kakavas

# Introduction to OAuth 2.0

OAuth 2.0 is an IETF standard for authorization. It supersedes OAuth 1.0 with which it is not backward compatible.

## OAuth 2.0 Core

- OAuth 2.0 Framework - RFC 6749
- Bearer Token Usage - RFC 6750
- Threat Model and Security Considerations - RFC 6819



## OAuth 2.0 Extensions

- OAuth 2.0 Device Flow (draft)
- OAuth 2.0 Token Introspection - RFC 7662, to determine the active state and meta-information of a token
- PKCE - Proof Key for Code Exchange, better security for native apps
- Native Apps - Recommendations for using OAuth 2.0 with native apps
- JSON Web Token - RFC 7519
- OAuth Assertions Framework - RFC 7521
- SAML2 Bearer Assertion - RFC 7522, for integrating with existing identity systems
- JWT Bearer Assertion - RFC 7523, for integrating with existing identity systems

# Introduction to OAuth 2.0

*The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. [RFC 6749]*

OAuth2 defines a mean to represent the authorization granted to the third-party, the access token, and a set of flows and mechanisms to:

- obtain the authorization, that is the access token
- convey the authorization to a third-party application
- use the authorization on a protected resource

All on top of the HTTP protocol

# OAuth 2.0 Actors

- **Resource owner** (RO): the granting access entity, usually the user and his User Agent
- **Resource Server** (RS): the server hosting the resource to be accessed (eg an API)
- **Client**: the application to which the grant is entitled (a web app, a desktop app, a mobile app, a javascript-on-top-of-user-agent app...)
- **Authorization Server** (AS): registers clients, authenticates users, and issues access tokens.

# OAuth 2.0 bits and pieces

**Access token:** a string representing an authorization issued to the Client (for which is usually opaque) - OAuth 2.0 does not mandate the format nor the content of the access token

**Refresh token:** credentials used to obtain access tokens when the current access token becomes invalid or expires.

**Scopes:** set of rights delegated to the client on the Resource Server - expressed as a list of space-delimited, case-sensitive strings.

## **Protocol Endpoints:**

- Authorization endpoint (Authorization Server)
- Token endpoint (Authorization Server)
- Redirection Endpoint (Client) [SHOULD require the use of TLS by RFC 6749]

# OAuth 2.0 Flows

## Authorization Code Grant

It is the main flow to obtain an access token, and mainly targeted to web applications.

- client authentication
- employ an intermediate authorization phase represented by an authorization code
- The access token is exchanged without the involvement of the Resource Owner User Agent

## Implicit Grant

A simplified authorization code flow optimized for clients implemented in a browser.

- No client authentication
- No intermediary code to obtain the access token

## Resource Owner Password Credentials Grant

It is a flow for highly trusted Clients:

- the Resource Owner credentials are used directly by the Client to obtain an authorization

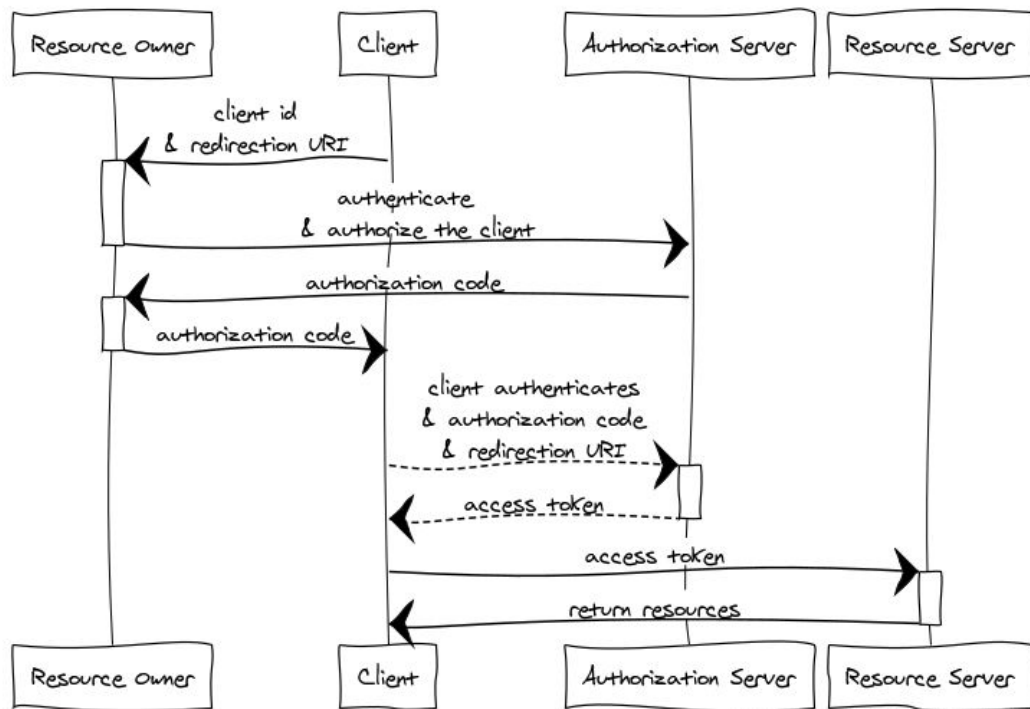
## Client Credentials Grant

It is a flow for third party Clients with very limited access to resources:

- It is based on Client credentials only

# Authorization Code Grant

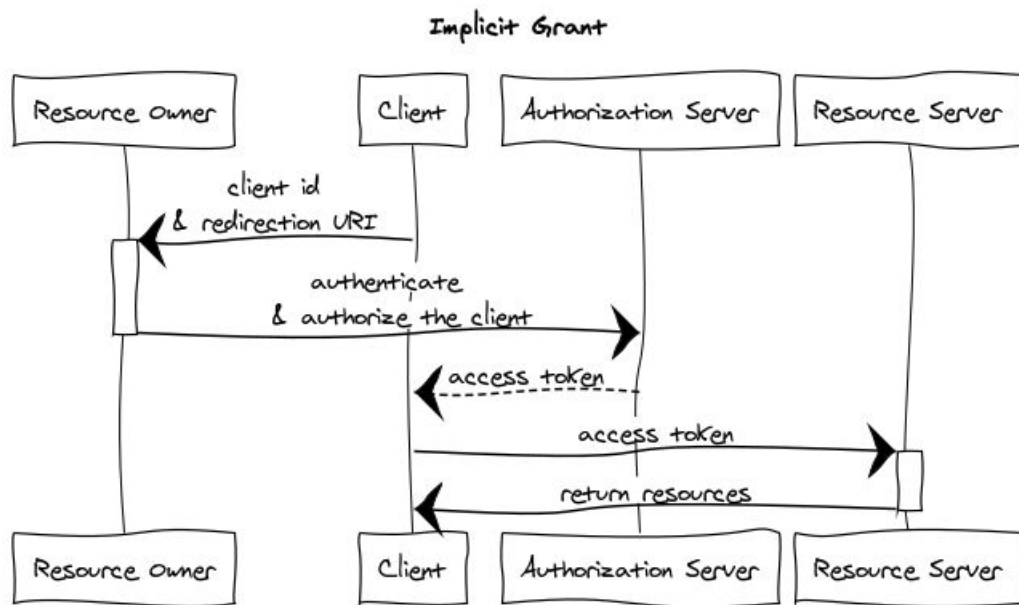
Authorization Code Grant



- The Client redirects the RO to the AS sending client\_id, scopes, and redirection URI
- The AS authenticates the RO and obtains user authorization for the Client
- The AS redirect the RO to the Client with an authorization code
- The Client authenticates on the AS and sends the authorization code along with the redirection URI (for verification)
- The AS sends an access token to the Client
- The Client sends the access token to the RS to access the resources

# Implicit Grant

*Note that in the implicit grant the Client usually runs on top of the Resource Owner User Agent*



- The Client redirects the RO to the AS sending client\_id, scopes, and redirection URI
- The AS authenticates the RO and obtains user authorization for the Client
- ~~The AS redirect the RO to the Client with an authorization code~~
- ~~The Client authenticates on the AS and sends the authorization code along with the redirection URI (for verification)~~
- The AS sends an access token to the Client
- The Client sends the access token to the RS to access the resources



# Authorization Code Grant - Demo

# Register a new OAuth application

---

## Application name

My OAuth2 APP

Something users will recognize and trust

## Homepage URL

https://myoauth2app.com

The full URL to your application homepage

## Application description

My OAuth2 APP

This is displayed to all potential users of your application

## Authorization callback URL

https://myoauth2app.com/cb

Your application's callback URL. Read our [OAuth documentation](#) for more information.

---

**Register application**

**Cancel**

## My OAuth2 APP



**[REDACTED]** owns this application.

**Transfer ownership**

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

**List this application in the Marketplace**

## 0 users

Client ID

**[REDACTED]**7

Client Secret

**[REDACTED]**c

**Revoke all user tokens**

**Reset client secret**

# Authorization Request

## Authorization Server

Authorization Endpoint: <https://github.com/login/oauth/authorize>

## Parameters

Parameter	Current Value
response_type	code
client_id	92c7c261f4c1d0a283a6
redirect_uri	<a href="http://oauth2client.authnzi.org:9000/cb">http://oauth2client.authnzi.org:9000/cb</a>
state	84ad5fb0aeb7ff7b0780c85608c03fa2c93a2658d1c9c466ed7b9909b461b52e
scope	user

## Full HTTP Authorization Request

Request authorization and get and authorization code



Sign into **GitHub**  
to continue to **OAuth2 Example  
Client**

**Username or email address**

**Password**

[Forgot password?](#)

**Sign in**

# Authorization Response

```
{'HTTP_ACCEPT': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
'HTTP_ACCEPT_ENCODING': 'gzip, deflate',
'HTTP_ACCEPT_LANGUAGE': 'it,en-US;q=0.7,en;q=0.3',
'HTTP_CONNECTION': 'keep-alive',
'HTTP_COOKIE': 'session=eyJzdGF0ZSI6eyIgYiI6I6k9EUmhaRFZtWWpCaFpXSTNabVZzWWpBM09EQmpPRFUYyTUroaklETmlZVEpqT1ROaElqWTFPRlF4WXpsak5EWTJaVlEzWWpr
'HTTP_HOST': 'oauth2client.authnzi.org:9000',
'HTTP_REFERER': 'https://github.com',
'HTTP_USER_AGENT': 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:47.0) Gecko/20100101 Firefox/47.0',
'PATH_INFO': '/cb',
'QUERY_STRING': 'code=9f42b9be9f37a3d2ade6&state=84ad5fb0aeb7ff7b0780c85608c03fa2c93a2658d1c9c466ed7b9909b461b52e',
'REMOTE_ADDR': '131.114.2.154',
'REMOTE_PORT': 46366,
'REQUEST_METHOD': 'GET',
'SCRIPT_NAME': '',
'SERVER_NAME': '0.0.0.0',
'SERVER_PORT': '9000',
'SERVER_PROTOCOL': 'HTTP/1.1',
'SERVER_SOFTWARE': 'Werkzeug/0.12.2',
'werkzeug.request': '<Request 'http://oauth2client.authnzi.org:9000/cb?code=9f42b9be9f37a3d2ade6&state=84ad5fb0aeb7ff7b0780c85608c03fa2c93a26
'werkzeug.server.shutdown': <function shutdown_server at 0x7f5e07dd8b90>,
'wsgi.errors': <open file '<stderr>', mode 'w' at 0x7f5e0b3e71e0>,
'wsgi.input': <open file '<socket>', mode 'rb' at 0x7f5e08885f60>,
'wsgi.multiprocess': False,
'wsgi.multithread': False,
'wsgi.run_once': False,
'wsgi.url_scheme': 'http',
'wsgi.version': (1, 0)}
```

## Authorization Response unpacked

### Parameters

Parameter	Current Value
code	9f42b9be9f37a3d2ade6
state	84ad5fb0aeb7ff7b0780c85608c03fa2c93a2658d1c9c466ed7b9909b461b52e

# Access Token Request

## Authorization Server

Token endpoint: [https://github.com/login/oauth/access\\_token](https://github.com/login/oauth/access_token)

### Parameters

Parameter	Current Value
grant_type	authorization_code
code	9f42b9be9f37a3d2ade6
state	84ad5fb0aeb7ff7b0780c85608c03fa2c93a2658d1c9c466ed7b9909b461b52e
client_id	92c7c261f4c1d0a283a6
client_secret	dea3756a34aaf54850f569e0f82e6e040d915be0
redirect_uri	<a href="http://oauth2client.authnzi.org:9000/cb">http://oauth2client.authnzi.org:9000/cb</a>

### Full Token request

```
POST https://github.com/login/oauth/access_token
```

```
grant_type=authorization_code
code=9f42b9be9f37a3d2ade6
state=84ad5fb0aeb7ff7b0780c85608c03fa2c93a2658d1c9c466ed7b9909b461b52e
client_id=92c7c261f4c1d0a283a6
client_secret=dea3756a34aaf54850f569e0f82e6e040d915be0
redirect_uri=http://oauth2client.authnzi.org:9000/cb
```

[Request an Access Token](#)

# Access Token Response

We got our access token, let's have a look...

## The raw response

We requested a json response passing the following header content: `accept: application/json`

This is what we got:

```
{"access_token":"bfdc7a3a3d9226d5b57e55a6934f643c615b0e27","token_type":"bearer","scope":"user"}
```

## Access Token Response unpacked

Parameter	Current Value
access_token	bfdc7a3a3d9226d5b57e55a6934f643c615b0e27
token_type	bearer
scope	user

## Use the Token

Let's use the received token to access some user info calling the github API endpoint passing the access\_token through HTTP headers.

```
GET /user HTTP/1.1
Host: api.github.com
Authorization: Bearer bfdc7a3a3d9226d5b57e55a6934f643c615b0e27
```

Access protected resources



```
{
  "login": "oauth2-exampleclient",
  "id": 29506477,
  "avatar_url": "https://avatars1.githubusercontent.com/u/29506477?v=3",
  "gravatar_id": "",
  "url": "https://api.github.com/users/oauth2-exampleclient",
  "html_url": "https://github.com/oauth2-exampleclient",
  "followers_url": "https://api.github.com/users/oauth2-exampleclient/followers",
  "following_url": "https://api.github.com/users/oauth2-exampleclient/following{/other_user}",
  "gists_url": "https://api.github.com/users/oauth2-exampleclient/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/oauth2-exampleclient/starred{/owner}{/repo}",
  "subscriptions_url": "https://api.github.com/users/oauth2-exampleclient/subscriptions",
  "organizations_url": "https://api.github.com/users/oauth2-exampleclient/orgs",
  "repos_url": "https://api.github.com/users/oauth2-exampleclient/repos",
  "events_url": "https://api.github.com/users/oauth2-exampleclient/events{/privacy}",
  "received_events_url": "https://api.github.com/users/oauth2-exampleclient/received_events",
  "type": "User",
  "site_admin": false,
  "name": null,
  "company": null,
  "blog": "",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": null,
  "public_repos": 0,
  "public_gists": 0,
  "followers": 0,
  "following": 0,
  "created_at": "2017-06-17T15:35:35Z",
  "updated_at": "2017-06-19T09:30:40Z",
  "private_gists": 0,
  "total_private_repos": 0,
  "owned_private_repos": 0,
  "disk_usage": 0,
  "collaborators": 0,
  "two_factor_authentication": false,
  "plan": {
    "name": "free",
    "space": 976562499,
    "collaborators": 0,
    "private_repos": 0
  }
}
```

# Summary

In order to use the OAuth2 Authorization code flow:

- register a Client (indicating the Redirection endpoint), obtain a `client_id` & `client_secret`
- issue an authorization request to the Authorization Endpoint by redirecting the user browser
- parse the Authorization Server request to the Client Redirection Endpoint and extract the authorization code
- issue an access token request to the Token Endpoint sending the authorization code
- parse the response to extract the access token
- use the access token on the Resource Server

Q&A

Thanks for your attention!