**OAUTH/OIDC**

**Oauth stands for Open Authorization Protocal. Its works as a Access Token, access tokens are form of JWT(Json web token).**

**OR**

**OAuth stands for Open Authorization. It is open standard. that apps use to provide client applications with access.**

**Or**

**Oauth is an open standard authorization protocol that enables third party apps to obtain limited access to an http services**

**In the OAuth 2.O has some important Roles**

**1.Resource Owner** :Who is the user and owner of the data

The actual end user, responsible for authentication to provide consent to share their resources with the requesting client.

**2. Client :** 1. client is the app That is accessed by the user.

2. This app requires the Authorization.

The client application that is requesting an access token on behalf of the end user

**3 Authorization server :** it is going to authorize the set client and issue the code or a token.

The Okta that authenticates the user and/or client, issues access tokens and tracks the access tokens throughout their lifetime.

**4. Resource server:** it is host the data that the client wants to access.

The target application or API that provides the requested resources. This actor will validate an access token to provide authorization for the action.

**Oath Terms**

1. **Redirect URI**

**The url the authorization server will redirect the resource owner back to after granting permission to the client**

1. **Response type**

**The type of information the client expects to receive the most common resourse type is CODE**

1. **Scope**

**These are the permission the client wants such as access to data or to perform actions**

1. **Consert**

**The authorization server takes the scopes the client is requesting and verifies with the resource owner whether or not they want to give the client permission**

1. **Client secret**

**This is the secret password**

1. **Authentication code**

**A short lived temporary code that client gives the Authorization server in exchanging the access token**

1. **Access token**

**The key the client will use to communicate with resource server.**

**There are three main actions an application developer needs to handle to implement**

OAuth 2.0:

1. Get an access token

2. Use an access token

3. Refresh an access token (optional)

**Tokens**

* **access token: The token issued by the authorization server (Okta) in exchange for the grant.**
* **refresh token: An optional token that is exchanged for a new access token if the access token has expired.**
* **ID token: The token issued by the OpenID provider that contains information about the end user in the form of claims.**
* **claim: The claim is a piece of information about the end user.**

**Redirect URl**

**Redirect uri is the place where the client lands after getting authorized.**

**OpenIDConnect**

**OpenID Connect (OIDC) is an authentication standard built on top of OAuth 2.0. It defines an ID token type to pair with OAuth 2.0 access and refresh tokens.**

**Okta : When we created application or app it gave Client ID and client secreat.**

**Metadata:**

**Okta – security- api- authorization server-defaults- metadata rul- check details**

**1.isssuer**

**2.Autherizarion endpoints : where the client will come to get authorize**

**3.token end point:** **Accessed bt the client to exchange the code for the token**

**4. user endpoint :** **get more information about the user**

**Grant Types**

**OAuth 2.0 provides five standard grant types that can be used to customize the authentication and authorization process depending on the application requirements.**

1. **Authorization Code flow**

**Used for most web & mobile apps scenarios that want to call REST web services**

1. **Implicit flow**
2. **Client Credential**
3. **Resource owner password flow**
4. **Refresh Tokens**
5. **Authorization Code flow. Response type : Code**

**The Authorization Code flow is the method for controlling access to web applications capable of securely storing secrets.**

**Or**

**The Authorization Code Grant Type is probably the most common of the OAuth 2.0 grant types. It is**used by both web apps and native apps to get an access token after a user authorizes an app.

**Flow**

**In this flow we have a client the user and the authorization server with authorization endpoint and token endpoint.**

1. **In this flow the client is going to request an authorization code from the authorization endpoints of the authorization server.**

**(here it will be use GET request and those are going to be the parameters passed in the query string Client ID, redirect uri, response type, scope, state)**

1. **When receives requests from the client The authorization endpoint is going to present Authorization server login screen to the user and Once complete The user authenticates with Authorization server. The Authorization endpoint is going to issued a Authorization code to the client.**
2. **The client will make another request. Generally a POST to the Token Endpoints in Authorization server. The Token endpoints to exchange the code for the token.**

**( here POST request will passed in the body of the request as it post client id, client secret, redirect uri, code, grant type)**

1. **The Token endpoints are going to issue an access token to the client.**
2. **Client Sends access token to resource server and access to app**

**If include OPENID in the scope . it will become OIDC flow and it will get an ID TOKEN and ACCESS TOKEN**

* Request authorization from user and retrieve authorization code

https://localhost:9031/as/authorization.oauth2?

client\_id=ac\_client

redirect\_uri=sample%3A%2F%2Foauth2%2Fcode%2Fcb

response\_type=code

scope=edit

state:12345

Once completed the Get request **Code** will generate.

Swap the authorization code for an access token

For this to occur, the client makes a HTTP POST request to the token endpoint on the AS. This request will use the following parameters sent in the body of the request:

1. grant type

2. Code

3. redirect uri

After success of Post request get Access Token

**Post in result**

1. **Headers**

**{**

**"kid": "NQt9JXW4GZYyJN9PkKPrI7NmwrIDyq-27Gn1HqpUY-w",**

**"alg": "RS256"**

**}**

1. **Payload**

**{**

**"ver": 1,**

**"jti": "AT.sQDcWF4PomBx\_wmCZk\_FmutumfxKYPP20eF6jTkiB7o",**

**"iss": "https://trial-8163325.okta.com/oauth2/default",**

**"aud": "api://default",**

**"iat": 1712375964,**

**"exp": 1712379564,**

**"cid": "0oacz9cdco3LUz8Y3697",**

**"uid": "00ucfhcx8drYUjse4697",**

**"scp": [**

**"openid"**

**],**

**"auth\_time": 1712375964,**

**"sub": "devakamballa@test.com",**

**"firstname": "deva",**

**"Email": "devakamballa@test.com",**

**"lastname": "kamballa"**

**}**

**3.Verify singnature**

**Signature**

**OR**

**Flow**

* **A client application makes an authorization code request to an authorization server**
* **Authorization server sends auth code to client though user agent**
* **Client sends the auth code to auth server and request for access token.**
* **Auth server will validate auth code and gives access token**
* **Sends access token to resource server and access to app**

**2.Implicit flow Grant Response\_type : Token (single page applications)**

Scenario where client is not able to safely hide the client secret (e.g. clientside JavaScript application). Uses the user agent to transport the OAuth2 tokens

**The Implicit flow is intended for applications where the confidentiality of the client secret can't be guaranteed. In this flow, the client doesn't make a request to the /token endpoint. Instead, it receives the access token directly from the /authorize endpoint. The client must be able to interact with the resource owner's user agent and to receive incoming requests (through redirection) from the authorization server.**

**In this flow we have a client the user and the authorization server with authorization endpoint.**

1. **In this flow the client is going to request an ACCESS TOKEN from the authorization endpoints of the authorization server.**

**(here it will be use GET request and those are going to be the parameters passed in the query string Client ID, redirect uri, response type, scope, state,nonse)**

1. **When receives requests from the client The authorization endpoint is going to present Authorization server login screen to the user and The user authenticates with Authorization server and Authorization endpoint is going to issued a Access Token to the client.**
2. **Client Sends access token to resource server and access to app**
3. **Here client will not go to the token endpoints to exchange the code for the token because there is no code .**
4. **Client wants to access token and client get the access token from Aep itself**

**If include OPENID in the scope . it will become OIDC flow and it will get an ID TOKEN and ACCESS TOKEN**

**OR**

**Request authorization from user and retrieve access token**

Using the below parameters as an example, the application will redirect the user to the following URL: https://localhost:9031/as/authorization.oauth2? client\_id=im\_client

response\_type=token

scope=edit

redirect\_uri=sample%3A%2F%2Foauth2%2Fimplicit%2Fcb

**Once success Get request Access token will generate**

**Get in result**

1. **Headers**

**{**

**"kid": "NQt9JXW4GZYyJN9PkKPrI7NmwrIDyq-27Gn1HqpUY-w",**

**"alg": "RS256"**

**}**

1. **Payload**

**{**

**"ver": 1,**

**"jti": "AT.sQDcWF4PomBx\_wmCZk\_FmutumfxKYPP20eF6jTkiB7o",**

**"iss": "https://trial-8163325.okta.com/oauth2/default",**

**"aud": "api://default",**

**"iat": 1712375964,**

**"exp": 1712379564,**

**"cid": "0oacz9cdco3LUz8Y3697",**

**"uid": "00ucfhcx8drYUjse4697",**

**"scp": [**

**"openid"**

**],**

**"auth\_time": 1712375964,**

**"sub": "devakamballa@test.com",**

**"firstname": "deva",**

**"Email": "devakamballa@test.com",**

**"lastname": "kamballa"**

**}**

**3.Verify singnature**

**Signature**

1. **If you give Response\_type : id\_token, scope: opened**

**Get ID\_tokens**

1. **If you gave response type ; id-token token**

**Get ID-token and access token**

1. **id-token code = id token and access token**
2. **code token = access token**
3. **Client Credential grant flow (API services)**

* **Server to server communication**
* **Machine to machine communication**

**Use the Client Credentials flow for server-side ("confidential") client applications with no end user. That means for machine-to-machine communication. In this scenario, your application needs to store its client ID and secret securely and to exchange them with Okta for an access token.**

**Flow**

* **Client provide client credential in Authorization server**
* **Authorization server validates the client credentials and generates a Access token**
* **Client provides the token in http headers when making a request to the resource server.**
* **Resource server validates the token and process the the request is valid**
* Client application (app) makes an authorization request to your Okta authorization server using its client credentials.

(Client\_id,Client\_password.grant\_type,scope)

* Okta responds with an access token if the request credentials are accurate.

Request access token

The client makes a request (HTTP POST) to the token endpoint with the client credentials presented as HTTP Basic authentication:

1.grant\_type

2.scope

**The client credentials can also be provided using the client\_id and client\_secret parameters in the contents of the POST.**

**If the Post request is success get Access token**

1. **Resource Owner Password Credentials (ROPC)**

**Flow**

* **Where the resource owner has a trust relationship with client this flow will use**
* **It can be used only when other flows are not available**
* **Resource owner provides the client with username and password**
* **Client request as access token by using client credential.**

1. The user authenticates with your client application, providing their user credentials.
2. Your app sends these credentials to the Okta authorization server with its client ID and secret in the request header.
3. The authorization server responds with an access token if the credentials are accurate.

**Request authorization from user and retrieve access token**

At this stage, the client displays a login form to the user and collects the credentials (e.g. username/password) and defined scope if required from the resource owner (user) and makes a HTTP POST to the token endpoint. For the example below, the following credentials were received by the client and are used to request an access token:

Username

Password

Using the below parameters as an example, the application will redirect the user to the following URL: https://localhost:9031/as/authorization.oauth2? client\_id=im\_client

Grant\_type

Client\_authentication

scope=edit

**grant\_type=password**

**username=joe**

**password=2Federate**

**scope=edit**

**Once success Post request Access token will generate**

**Post in result**

1. **Headers**

**{**

**"kid": "NQt9JXW4GZYyJN9PkKPrI7NmwrIDyq-27Gn1HqpUY-w",**

**"alg": "RS256"**

**}**

1. **Payload**

**{**

**"ver": 1,**

**"jti": "AT.sQDcWF4PomBx\_wmCZk\_FmutumfxKYPP20eF6jTkiB7o",**

**"iss": "https://trial-8163325.okta.com/oauth2/default",**

**"aud": "api://default",**

**"iat": 1712375964,**

**"exp": 1712379564,**

**"cid": "0oacz9cdco3LUz8Y3697",**

**"uid": "00ucfhcx8drYUjse4697",**

**"scp": [**

**"openid"**

**],**

**"auth\_time": 1712375964,**

**"sub": "devakamballa@test.com",**

**"firstname": "deva",**

**"Email": "devakamballa@test.com",**

**"lastname": "kamballa"**

**}**

**3.Verify singnature**

**Signature**

**Diffrents between Oauth and opened**

|  |  |
| --- | --- |
| **Oauth** | **OIDC** |
| **It is open standerd authorization protocal** | **It is used for authentication on top of the oauth** |
| **It passes access tokens** | **It passes access tokens and id tokents** |
| **Scope =edit** | **Scope =openid** |

**ID Token and access token**

|  |  |
| --- | --- |
| **Id token** | **Access token** |
| **Asume the user is authenticated** | **call on API** |
| **Get user profile data** | **Check if client is allowed something** |
|  |  |

**If decoded**

|  |  |
| --- | --- |
| **Id token** | **Access token** |
| **Header{**  **Alg**  **Jwt}** | **Header { alg**  **Jwt**  **Kids}** |
| **Payload {**  **Issuer**  **Audience**  **Sub**  **Exp**  **Attributes**  **Name**  **Fn**  **ln** | **Payload**  **Issuer**  **Sub**  **Exp**  **Sope(create terms update terms** |
| **signature** | **signature** |

**KID- Key id which holds a key identifier**

**5.Refresh token**

If a refresh token was requested along with the access token, then the refresh token can be used to request a new access token without having to ask the user to re-authenticate. If the refresh token is still valid, then a new access token and refresh token will be returned to the client. If the refresh token has been invalidated for any reason, then the client must require the user to reauthenticate to retrieve a new access token. The reasons for refresh tokens becoming invalid are:

• Refresh token has expired;

• Refresh token has been administratively revoked (separation / security reasons);

• User has explicitly revoked the refresh token

Post request

grant\_type=refresh\_token

&refresh\_token=123…789

HTTP Response

"access\_token":"aaa…ccc",

"token\_type":"Bearer",

"expires\_in":14400,

"refresh\_token":"456…321" }

"

**Secure Web Authentication (SWA)**

Secure Web Authentication (SWA) is a technology used by Okta that provides Single Sign-On (SSO) functionality to external web applications that don't support federated protocols like SAML, Web Services Federation (WS-Fed), or OpenID Connect (OIDC)