OAuth2.O/OIDC

01).What is Oauth2.O? How it is works? Advantages of Oauth?

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

**Its works as a Access Token Mechanism, access tokens are form of JWT(Json web token).**

**Advantages**

**Increased Security**

#### Improved User Experience

#### Reduced Development Time And Costs

* **Increased Security:** Users can safely authorize third-party applications without needing to enter sensitive details like usernames and passwords.
* **Improved User Experience:** Authorizing apps with OAuth is simple and straightforward, eliminating the need for lengthy forms.
* **Enhanced Data Protection:** By limiting third-party apps’ access rights on an account level, data remains more secure than ever before. In short, implementing OAuth authentication in your application ensures a smoother experience for users while providing greater peace of mind for both them and you as the application provider.

02).What is OIDC? How it is works? Advantages of OIDC?

**OpenID Connect (OIDC) is an authentication standard built on top of OAuth 2.0. It passes Access Tokens and Id Tokens**

OIDC stands for OpenID Connect protocol, which is an identity authentication protocol used to enable two unrelated applications to share user profile information without compromising user credentials.

03) What is Access Tokens and what does it contains?

**The access token is a piece of code used for authenticating the client application to access specific resources on the resource owner’s.**

**The token issued by the authorization server (Okta) in exchange for the grant.**

|  |
| --- |
| **Access token** |
| **Header { alg**  **Jwt**  **Kids}** |
| **Payload**  **Issuer**  **Sub**  **Exp**  **Sope(create terms update terms** |
| **signature** |

an access token contains authorization information about the actions the application is allowed to perform at the API

04) What is Id Tokens and what does it contains ?

**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.**

**Id Token Contains**

|  |
| --- |
| **Id token** |
| **Header{**  **Alg**  **Jwt}** |
| **Payload {**  **Issuer**  **Audience**  **Subject**  **Expire**  **Attributes**  **Name**  **Fn**  **ln** |
| **signature** |

ID Token are the JSON Web Token (JWT) format. A JWT consists of a JSON header, a JSON payload, and a signature that can be verified. The payload contains a number of fields, called "claims", that describe the token and what it can access.

01) What are the grant types of Oauth and explain?

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

1. **Authorization Code flow**
2. **Implicit flow**
3. **Client Credential**
4. **Resource owner password flow**

**Authorization Code flow** .

**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 server is going to issue 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. **If code verification is success then Authorization server will issue an access token to the client.**
2. **Client 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**
5. **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**

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.

06) What does it contains Oauth Metadata? Where to check it in okta?

**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**

06) Difference B/w Saml and Oauth?

SAML (Security Assertion Markup Language) and OAuth (Open Authorization) are both protocols used for authentication and authorization, but they serve different purposes and have different architectures:

1. **Purpose:**
   * **SAML:** SAML is primarily used for single sign-on (SSO), allowing users to log in once and access multiple applications without needing to log in again. It facilitates the exchange of authentication and authorization data between identity providers (IdP) and service providers (SP).
   * **OAuth:** OAuth is primarily used for delegated authorization, allowing a user to grant a third-party application limited access to their resources without sharing their credentials. It's commonly used for API authorization and access delegation in scenarios like social login (e.g., "Login with Facebook") and accessing third-party services.
2. **Flow and Authorization:**
   * **SAML:** In SAML, the authentication process involves the exchange of XML-based security tokens between the identity provider (IdP) and the service provider (SP). The user authenticates with the IdP, which generates a SAML assertion containing authentication and authorization data. The SP trusts the IdP and validates the assertion to grant access.
   * **OAuth:** In OAuth, there are different flows (e.g., Authorization Code, Implicit, Client Credentials) for different scenarios, but the general flow involves the user granting authorization to a third-party application. This authorization is represented by an access token, which the third-party application can use to access the user's resources (with limitations defined by the scope).
3. **Usage:**
   * **SAML:** SAML is commonly used in enterprise environments for SSO, where users need to access multiple applications within an organization's ecosystem without needing to log in separately to each one.
   * **OAuth:** OAuth is commonly used in web and mobile applications to enable features like social login, where users can log in using their existing credentials from a provider like Google or Facebook, as well as to provide access to APIs and services without sharing sensitive information like passwords.
4. **Technology Stack:**
   * **SAML:** SAML is XML-based and relies on cryptographic signatures for security. It typically involves more heavyweight XML processing and is often seen as more complex to implement.
   * **OAuth:** OAuth is HTTP-based and typically uses JSON-formatted tokens. It's more lightweight and is designed to be easier to implement for web and mobile applications.

07) Difference b/w Oauth and OIDC?

OAuth 2.0 and OpenID Connect (OIDC) are both protocols used for authentication and authorization, but they serve different purposes and have different functionalities:

1. **OAuth 2.0:**
   * **Purpose:** OAuth 2.0 is primarily an authorization framework that allows third-party applications to obtain limited access to a user's resources without exposing the user's credentials.
   * **Functionality:** OAuth 2.0 enables scenarios such as social login (e.g., "Login with Facebook"), API authorization, and delegated access to resources.
   * **Tokens:** OAuth 2.0 defines various types of tokens, such as access tokens, refresh tokens, and authorization codes, which are used for authorization and access delegation.
   * **Security:** OAuth 2.0 focuses on providing a framework for secure authorization flows, but it does not define standards for authentication or user identity verification.
2. **OpenID Connect (OIDC):**
   * **Purpose:** OIDC is an authentication layer built on top of OAuth 2.0, providing identity verification and user authentication services.
   * **Functionality:** OIDC extends OAuth 2.0 to provide a standardized way for clients to verify the identity of the end-user based on authentication performed by an authorization server, as well as to obtain basic user profile information.
   * **Tokens:** OIDC introduces a new type of token called the ID token, which contains information about the authenticated user and the authentication event.
   * **Security:** OIDC adds additional security features to OAuth 2.0, such as ID token validation, token binding, and session management, to enhance the security of authentication processes.

08) Difference b/w Access Tokens and ID Tokens?

Access tokens and ID tokens are both integral parts of OAuth 2.0 and OpenID Connect (OIDC) protocols, used in modern authentication and authorization systems. However, they serve different purposes and contain different types of information:

1. **Access Tokens:**
   * **Purpose:** Access tokens are used to authorize access to protected resources, such as APIs, on behalf of a user.
   * **Content:** Access tokens typically contain information about the permissions granted to the client application and are used to access protected resources.
   * **Expiration:** They have a limited lifespan and need to be refreshed or renewed periodically to maintain access.
   * **Usage:** Access tokens are sent with API requests in the Authorization header (typically as a bearer token) to authenticate the client and authorize access to the requested resource.
   * **Example Scenario:** After a user logs in and grants permissions to a third-party application, the authorization server issues an access token to that application. The application then includes this token in API requests to access the user's data from the resource server.
2. **ID Tokens:**
   * **Purpose:** ID tokens are used to provide information about the authenticated user.
   * **Content:** ID tokens typically contain information about the user (such as user ID, name, email) and information about the authentication event (such as authentication time, issuer).
   * **Expiration:** Like access tokens, they also have a limited lifespan but are typically shorter-lived since they contain sensitive user information.
   * **Usage:** ID tokens are primarily used by the client application to obtain information about the authenticated user and make authorization decisions within the application.
   * **Example Scenario:** After a user logs in using OIDC, the authorization server issues an ID token containing information about the user. The client application can then use this token to personalize the user's experience, display their name, or make authorization decisions based on their role or other attributes.

09) what arethe prerequisites of Oauth?

Redirect Uri

Grant type

Claims (user attributes)