**OpenID Connect (OIDC)** is a modern, widely used authentication protocol built on top of OAuth 2.0. It allows clients (such as web or mobile applications) to verify a user's identity and gain access to basic profile information in a secure and standardized way. Here's everything you need to know about OIDC:

**Overview of OpenID Connect (OIDC)**

**What is OIDC?**

* **Authentication Protocol**: OIDC focuses on authenticating users (proving "who they are") rather than authorization (deciding "what they can do").
* **Built on OAuth 2.0**: Adds identity capabilities to OAuth 2.0 by introducing an **ID Token** and standardized scopes.
* **JSON Web Tokens (JWT)**: Relies on the JWT format for secure transmission of identity information.

**Key Components of OIDC**

1. **ID Token**:
   * A JWT that contains claims about the user, such as their identity, authentication time, and more.
   * Signed by the Identity Provider (IdP) to ensure integrity.
   * Example Claims:

{

"iss": "https://idp.example.com",

"sub": "1234567890",

"aud": "client-id-123",

"exp": 1699999999,

"iat": 1699990000,

"name": "John Doe",

"email": "john.doe@example.com"

}

1. **Scopes**:
   * Define what information the client app can request.
   * Common Scopes:
     + openid: Required for OIDC; signifies the use of OIDC.
     + profile: Access to basic profile information.
     + email: Access to the user's email address.
     + offline\_access: Enables issuing of refresh tokens.
2. **Endpoints**:
   * **Authorization Endpoint**: Initiates user authentication and obtains authorization.
   * **Token Endpoint**: Exchanges authorization codes or refresh tokens for an ID Token and Access Token.
   * **UserInfo Endpoint**: Retrieves additional user information.
3. **Identity Provider (IdP)**:
   * The server that authenticates the user and issues tokens (e.g., Google, Okta, Auth0).
4. **Client (Relying Party)**:
   * The application that uses OIDC to authenticate users.

**OIDC Flow Types**

OIDC supports several flows depending on the type of client:

1. **Authorization Code Flow** (Recommended for server-side apps):
   * Secure, as tokens are exchanged on the server.
   * Steps:
     1. User is redirected to the IdP.
     2. IdP authenticates the user and sends an authorization code back to the app.
     3. The app exchanges the code for tokens (ID Token, Access Token) via the Token Endpoint.
2. **Implicit Flow** (Deprecated for most use cases):
   * Tokens are returned directly in the redirect URL.
   * Less secure; discouraged in favor of Authorization Code Flow with PKCE.
3. **Hybrid Flow**:
   * Combines features of the Authorization Code and Implicit flows.
4. **Authorization Code Flow with PKCE** (For public clients like mobile apps):
   * Adds a code challenge and verifier to prevent interception of authorization codes.

**Tokens in OIDC**

1. **ID Token**:
   * Used to authenticate the user.
   * Contains claims about the user and authentication event.
   * Example claims: iss (issuer), sub (subject), aud (audience), iat (issued at), exp (expiration).
2. **Access Token**:
   * Grants access to protected resources (e.g., APIs).
   * Used in conjunction with OAuth 2.0.
3. **Refresh Token**:
   * Used to obtain new access tokens without user reauthentication.

**OIDC Security Features**

1. **JWT Signature Verification**:
   * The ID Token is signed by the IdP using a private key.
   * The client verifies it using the IdP’s public key.
2. **Nonce**:
   * Prevents replay attacks by including a unique identifier in the authentication request and token.
3. **State Parameter**:
   * Protects against Cross-Site Request Forgery (CSRF).
4. **PKCE**:
   * Adds an extra layer of security for public clients by mitigating interception of authorization codes.

**OIDC vs. OAuth 2.0**

| **Aspect** | **OIDC** | **OAuth 2.0** |
| --- | --- | --- |
| **Purpose** | Authentication (who the user is) | Authorization (what the user can access) |
| **Token** | ID Token | Access Token |
| **User Info** | Provides user profile info | Doesn't handle user identity |
| **Use Case** | Login to apps, Single Sign-On (SSO) | API access control |

**OIDC Example Use Cases**

1. **Single Sign-On (SSO)**:
   * Use OIDC to allow users to log in to multiple apps with a single set of credentials.
2. **Social Login**:
   * Authenticate users using social identity providers like Google, Facebook, or Microsoft.
3. **Federated Identity**:
   * Integrate multiple identity systems for seamless user authentication.
4. **API Access**:
   * Combine OIDC for authentication with OAuth 2.0 for authorization.

**Popular OIDC Providers**

1. **Google Identity Platform**
2. **Microsoft Azure AD**
3. **Okta**
4. **Auth0**
5. **Keycloak**

**OIDC Implementation Steps**

1. **Register Your App**:
   * Register with the chosen IdP to get a client ID and secret.
2. **Configure Your App**:
   * Use libraries like OpenID Connect libraries for your framework (e.g., oidc-client.js for JavaScript, Flask-OIDC for Python).
3. **Implement the Authentication Flow**:
   * Redirect users to the Authorization Endpoint.
   * Handle token exchanges at the Token Endpoint.
   * Verify the ID Token.
4. **Handle Security**:
   * Validate JWT signatures and expiration times.
   * Use HTTPS and PKCE for secure token exchanges.

**OIDC Libraries and Tools**

* **JavaScript**: oidc-client, openid-client
* **Python**: Flask-OIDC, python-jose
* **Java**: Spring Security OIDC
* **Node.js**: passport-openidconnect
* **.NET**: IdentityServer4

**Best Practices for OIDC**

1. Use HTTPS to secure all communication.
2. Validate ID Token signatures and expiration claims.
3. Use Authorization Code Flow with PKCE for public clients.
4. Implement logout functionality to invalidate sessions.

OIDC is a powerful protocol for modern authentication needs. Whether you’re building Single Sign-On (SSO) systems or integrating social logins, OIDC provides a robust and secure framework.