

OAuth and OIDC

Introduction

OAuth is an open standard protocol that enables secure authorization of third-party applications to access a user's resources without exposing their credentials. It allows users to grant limited access to resources (such as profiles, data, or services) hosted on one platform to another without sharing their credentials (like usernames and passwords).

OIDC is an authentication layer built on OAuth 2.0, adding an identity layer to OAuth. It provides a standardized way for clients to verify end-users identities based on authentication performed by an authorization server.

OAuth

OAuth (Open Authorization) is a widely adopted framework that allows third-party applications to access resources (like user data) on behalf of a resource owner (typically a user) without exposing the owner's credentials. It's designed for secure authorization and access delegation in the context of APIs, web applications, and services. OAuth specifies a set of protocols and best practices for token-based authorization.

Critical Components of OAuth:

1. Actors (Roles):

- a. **Resource Owner:** The entity that grants access to its protected resources. Typically, this is the end-user.
- b. **Client:** The application requesting access to the protected resources on behalf of the resource owner.
- c. **Resource Server:** The server hosts the protected resources that the client wants to access.
- d. **Authorization Server:** The server authenticates the resource owner and issues access tokens after obtaining authorization.

2. Tokens:

- a. **Access Token:** A credential representing the authorisation the resource owner granted to the client. It allows access to specific resources on the resource server.
- b. **Refresh Token:** An optional token to obtain a new access token when the current one expires without needing to re-authenticate the user.

3. **Authorization Grant Types:** Different grant types define how clients obtain access tokens. Some standard grant types include:
 - a. **Authorization Code:** In web applications, the client receives an authorization code and exchanges it for an access token.
 - b. **Implicit Grant:** Suitable for client-side applications like JavaScript-based apps, where the access token is returned directly.
 - c. **Client Credentials:** Used for server-to-server communication where no user is involved, the client requests an access token directly.
4. **Flows or Protocols:**
 - a. **Authorization Flow:** Involves redirecting the user to the authorization server's authentication and consent page before granting the client an access token.
 - b. **Token Exchange:** Process by which a client exchanges an authorization grant for an access token.

OAuth Workflow Steps:

1. **Client Registration:** The client registers with the authorization server and receives client credentials (client ID and client secret) for authentication.
2. **Authorization Request:** The client requests authorization from the resource owner to access their resources by redirecting them to the authorization server's authentication page.
3. **User Authentication and Consent:** The resource owner authenticates themselves and grants or denies permission to the client's request.
4. **Token Request:** If the authorization is granted, the client exchanges the authorization grant for an access token from the authorization server.
5. **Accessing Protected Resources:** The client uses the access token to access the protected resources on the resource server on behalf of the resource owner.

OAuth Security Considerations:

- Protecting tokens from interception or leakage.
- Ensuring secure storage and transmission of credentials.
- Managing token expiration and refresh mechanisms securely.
- Authorization server security against various attacks.

OAuth provides a standardized and secure framework for authorization, enabling controlled access to resources without exposing sensitive user credentials. Its versatility and adoption make it a cornerstone of modern authorization mechanisms in web and API-driven environments.

OIDC

OIDC (OpenID Connect) is an identity layer built on OAuth 2.0, designed to provide authentication and user identity information in modern web applications. It extends OAuth 2.0's capabilities by adding a standardized way to authenticate users and obtain information about them.

Critical Components of OIDC:

1. **ID Tokens:**
 - a. **ID Token:** A JSON Web Token (JWT) that contains claims about the authentication of an end-user, providing information such as the user's identity and authentication status.
2. **Endpoints and Discovery:**
 - a. **Authorization Endpoint:** The endpoint where clients initiate the authentication process by redirecting users for authentication and consent.
 - b. **Token Endpoint:** Used by clients to obtain access tokens and ID tokens.
 - c. **UserInfo Endpoint:** A standardized endpoint where OIDC providers expose user information after successful authentication.
3. **Authentication Flows:**
 - a. **Authentication Flows:** OIDC defines authentication flows, or methods clients can use to authenticate users and obtain identity information.
 - i. **Authorization Code Flow:** Similar to OAuth's Authorization Code Grant, it involves exchanging an authorization code for an ID token and access token.
 - ii. **Implicit Flow:** Returns ID tokens directly after authentication without using an authorization code.
4. **Claims and Scopes:**
 - a. **Claims:** Information about the authenticated user conveyed in the ID token, such as user ID, email, name, etc.
 - b. **Scopes:** Permissions or sets of information requested by the client during authentication, determining the level of access to user data.
5. **User Authentication and Consent:**
 - a. **User Authentication:** OIDC providers authenticate users through their identity credentials (username/password, social logins, etc.).
 - b. **Consent:** Users grant or deny permission to release their information to the requesting client.

OIDC Workflow:

1. **Client Registration:** The client registers with the OIDC provider and receives client credentials (client ID and client secret) for authentication.
2. **Authentication Request:** The client initiates the authentication process by redirecting the user to the OIDC provider's authentication endpoint.
3. **User Authentication and Consent:** The user authenticates and grants consent for sharing their information with the client application.
4. **ID Token and Access Token Issuance:** After successful authentication and consent, the OIDC provider issues an ID token containing user information and an access token for accessing protected resources.
5. **Accessing User Information:** The client uses the ID token to identify the user and access user-specific information via the UserInfo endpoint.

OIDC Advantages:

- Standardizes user authentication and identity information retrieval.
- Supports Single Sign-On (SSO) across multiple applications.
- Provides a user-centric identity framework.
- Extends OAuth 2.0's capabilities with identity-related features.

OIDC enhances OAuth 2.0 by focusing on user authentication and identity, providing a standardized and secure way for applications to authenticate users and access their identity information. It's widely used in various applications requiring user authentication and information retrieval.

OKTA

Okta is a comprehensive identity management platform that provides secure authentication, authorization, and user management services for businesses and organizations. It offers tools and services that enable businesses to manage user identities, access control, and security policies across various applications and devices.

Key Features of Okta:

1. **Single Sign-On (SSO):**
 - a. **Centralized Authentication:** Allows users to log in once and access multiple applications without re-authentication.
 - b. **Unified User Experience:** Provides a seamless login experience across web, mobile, and on-premises applications.

2. **Multi-Factor Authentication (MFA):**
 - a. **Enhanced Security:** Supports multiple factors for authentication, including SMS, email, biometrics, and hardware tokens, adding an extra layer of security.
3. **User Lifecycle Management:**
 - a. **User Provisioning and Deprovisioning:** Streamlines adding and removing users from various systems and applications.
 - b. **User Directory:** Manages user profiles, attributes, and groups in a centralized directory.
4. **Access Management:**
 - a. **Authorization and Access Policies:** Enforces access controls and policies based on user roles, groups, and attributes.
 - b. **API Access Management:** Provides security and access controls for APIs, including authorization and rate limiting.
5. **Adaptive Authentication:**
 - a. **Contextual Access Policies:** Adjusts authentication requirements based on contextual factors like device, location, and behaviour patterns.
 - b. **Risk-Based Authentication:** Identifies and responds to potential security threats using risk-based authentication policies.
6. **Integration and Customization:**
 - a. **Extensive Integrations:** Offers integrations with various applications, protocols, and identity standards like SAML, OAuth, and OIDC.
 - b. **Customizable Workflows:** Allows customization of authentication flows and branding to match the organization's needs.
7. **Reporting and Compliance:**
 - a. **Audit and Reporting:** Provides detailed logs and reporting for user activities, access, and security events.
 - b. **Compliance Support:** Helps organizations adhere to industry standards and compliance requirements like GDPR, HIPAA, etc.
8. **Developer-Friendly Tools:**
 - a. **APIs and SDKs:** Offers APIs and software development kits (SDKs) for easy integration and development of identity-related features.

Okta Components and Architecture:

- **Okta Identity Cloud:** The centralized platform that provides all the identity and access management services.
- **Okta Integration Network:** Catalog of pre-built integrations with various applications and services.
- **Okta API:** Allows developers to access and customize Okta's functionality through RESTful APIs.

Use Cases for Okta:

- **Enterprise Identity and Access Management:** Provides a unified solution for managing employee identities and access to corporate resources.
- **Customer Identity and Access Management (CIAM):** Helps businesses manage customer identities, providing a seamless and secure experience across applications.
- **Partner and B2B Identity Management:** Enables secure access for partners, suppliers, and external collaborators.

Okta's robust features, scalability, and flexibility make it a popular choice for organizations looking to streamline their identity and access management processes, ensuring security and efficiency across their digital ecosystems.

Sign in with Google

"Sign in with Google" is an authentication mechanism allowing users to log in to third-party applications or services using their Google account credentials, eliminating the need to create and manage separate login credentials for each service. It's a convenient and secure way to authenticate users by leveraging Google's authentication system.

Critical Components of "Sign in with Google":

1. **OAuth 2.0 Integration:**
 - a. **Authentication Protocol:** Uses OAuth 2.0 to authenticate users and authorize access to their Google account data.
 - b. **Authorization Grant Types:** Typically employs authorization code grant or implicit grant flows for obtaining access tokens.
2. **Google OAuth APIs:**
 - a. **Google Sign-In API:** Provides libraries and SDKs for various platforms (web, mobile, etc.) to integrate "Sign in with Google" functionality.
 - b. **User Authentication:** Allows users to log in using their Google account credentials.

3. **Access to User Information:**

- a. **User Profile Data:** Grants access to user information like name, email, profile picture, and other relevant details associated with their Google account.
- b. **Scope Permissions:** Users can control what information they allow the third-party application to access via scope permissions during authentication.

4. **Security and Convenience:**

- a. **Secure Authentication:** Utilizes Google's robust authentication mechanisms, ensuring secure user logins.
- b. **Single Sign-On (SSO):** Enables users to log in once and access multiple applications that support "Sign in with Google" without re-entering credentials.

5. **Implementation Process:**

- a. **Client-Side Integration:** Includes embedding a Google Sign-In button in the application's login page.
- b. **Server-Side Verification:** After successful authentication, the application server exchanges the received authorization code for an access token and validates it with Google's servers.

Benefits of "Sign in with Google":

1. **User Convenience:** Simplifies the login process, as users can use their existing Google credentials.
2. **Security:** Leverages Google's robust security measures for user authentication.
3. **Trust and Familiarity:** Users are familiar with Google's login process and trust its security.
4. **Reduced Friction:** Enhances user experience by minimizing the need for remembering multiple login credentials.

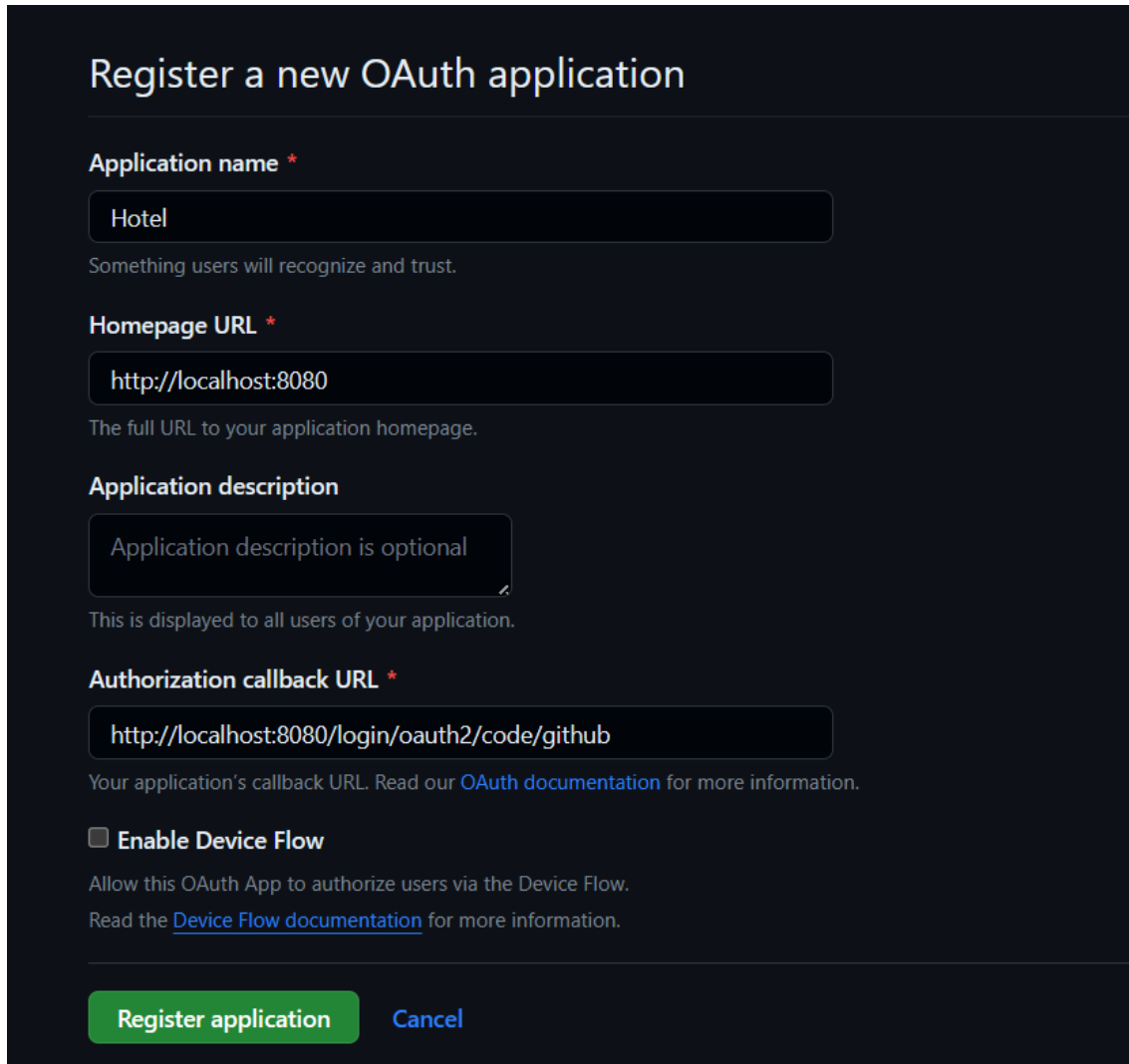
Use Cases:

- **Web Applications:** E-commerce platforms, forums, social networks, etc.
- **Mobile Apps:** Android and iOS applications across various domains.
- **Cross-Platform Authentication:** Enables seamless login experiences across different devices and platforms.

Implementing "Sign in with Google" involves integrating Google's OAuth and authentication APIs into the application's login flow, allowing users to authenticate using their Google accounts, thereby enhancing user experience and security in the authentication process.

Sign in with Github

1. To use GitHub's OAuth 2.0 authentication system, we must first create a new [GitHub application](#):



Register a new OAuth application

Application name *

Hotel

Something users will recognize and trust.

Homepage URL *

http://localhost:8080

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

http://localhost:8080/login/oauth2/code/github

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

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.
Read the [Device Flow documentation](#) for more information.

Register application **Cancel**

2. Now, we'll need to modify our application.yml:

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId: ${GITHUB_CLIENT_ID}
            clientSecret: ${GITHUB_CLIENT_SECRET}
```

3. Now, we need to Configure the security config my modifying filterChain bean.
4. Now, we need to add the "Login with Github" link in our login.html file.

Thats it! We have completed our OAuth2 login with Github.

Conclusion

OAuth and OIDC are integral components of modern authentication and authorization mechanisms. OAuth, a widely adopted framework, focuses on authorization and resource access delegation, allowing secure access to protected resources without sharing user credentials. It facilitates controlled access between clients and resource servers while ensuring secure delegation through access tokens.

OIDC, built upon OAuth, adds an identity layer, explicitly focusing on user authentication and providing standardized identity information. It enhances OAuth's capabilities by offering ID tokens containing user identity data, standardized endpoints for user information retrieval, and authentication flows.

Together, OAuth and OIDC provide a robust framework for secure, standardized, and user-centric authentication and authorization, catering to the complex needs of modern applications across various platforms.

Instructor Codes

- [Hotel Application with Okta](#)
- [Hotel Application with Google](#)

References

1. [OAuth](#)
2. [OIDC](#)
3. [Okta Official Documentation](#)
4. [Sign in with google](#)
5. [Sign in with github](#)