Types of Authentications inside ASP.NET

1. **Form Authentication**
2. **Passport Authentication**
3. **Windows Authentication**
   1. **Anonymous**
   2. **Basic**
   3. **Digest**
   4. **Integrated Windows (NTML & Kerberos)**
4. **Custom authentication Provider**
   1. **Multipass**
   2. **JWT (JSON Web token)**
   3. **SSO Protocol**
      1. **SAML (Security Assertion Markup Language)**
      2. **OAuth**
      3. **OpenID**

<https://www.c-sharpcorner.com/article/authentication-and-authorization-in-Asp-Net/>

Diagram

Description automatically generated

SAML: Authentication and Authorization both

Diagram

Description automatically generated

OAuth: Authorization Only

Graphical user interface

Description automatically generated with medium confidence

OAuth 2.0 (Open Authorization 2.0) is an open standard and widely used protocol that allows secure and delegated access to protected resources on behalf of a user. It is commonly used to enable third-party applications or services to access resources from a user's account on a different website (known as the resource server) without requiring the user to share their login credentials with the third-party application.

OAuth 2.0 provides a more secure and flexible authorization mechanism compared to its predecessor, OAuth 1.0. It is widely adopted by many online platforms and APIs for granting access to user data securely. The protocol typically involves four parties:

1. Resource Owner: The user who owns the data or the resource being protected and wants to grant access to a third-party application.

2. Resource Server: The server that hosts the protected resources and can respond to requests for those resources. It validates access tokens and provides access to resources if the token is valid.

3. Client: The third-party application that requests access to the user's resources on the resource server. This is the application that the user wants to grant access to their data.

4. Authorization Server: The server that issues access tokens to the client after successfully authenticating the user and obtaining their consent. It acts as an intermediary between the client and resource server.

The OAuth 2.0 flow generally involves the following steps:

1. The client requests authorization from the resource owner by redirecting them to the authorization server.

2. The resource owner authenticates themselves and provides consent to the client's request.

3. Upon successful authentication and consent, the authorization server issues an access token to the client.

4. The client presents the access token to the resource server to request access to the protected resources.

5. The resource server verifies the access token, and if it's valid, it provides access to the requested resources.

By using OAuth 2.0, users can control which applications have access to their data without directly sharing their login credentials. Additionally, access tokens can be short-lived and easily revocable, enhancing security and user control over their data. OAuth 2.0 is widely used in various applications, including social media platforms, online services, and API integrations.

OAuth 2.0 provides several types of authorization flows to accommodate different use cases and security requirements. Each flow is designed to address specific scenarios and is used to obtain access tokens, which are then used to access protected resources. The main type of authorization flows in OAuth 2.0 are:

1. Authorization Code Flow:

- This is the most commonly used flow for web applications. It involves two steps:

a. Authorization: The client requests authorization from the resource owner (user) by redirecting them to the authorization server. After the user grants permission, the authorization server issues an authorization code to the client.

b. Token Exchange: The client exchanges the authorization code with the authorization server for an access token, which can then be used to access protected resources.

2. Implicit Flow:

- This flow is designed for browser-based applications (JavaScript or Single Page Applications) that cannot securely store client credentials. It involves a single step:

a. Authorization: The client directly requests an access token from the authorization server, passing the required parameters in the URL fragment. The access token is returned directly to the client without using an authorization code.

3. Resource Owner Password Credentials Flow:

- This flow is intended for trusted applications (such as first-party clients) where the client can securely collect the user's username and password. It involves a direct exchange of credentials:

a. User Credentials: The client directly sends the user's credentials (username and password) to the authorization server to obtain an access token.

4. Client Credentials Flow:

- This flow is used when the client (the application itself) needs to access resources that it owns, not on behalf of any particular user. It involves direct client authentication with the authorization server:

a. Client Authentication: The client directly sends its credentials (client ID and client secret) to the authorization server to obtain an access token.

5. Refresh Token Flow:

- This flow is used to refresh an expired access token without involving the user. It allows the client to obtain a new access token using a refresh token that was previously obtained.

6. Device Authorization Flow (Device Flow):

- This flow is designed for devices with limited input capabilities (such as smart TVs, game consoles) where the user can't interact directly with the device. It involves a two-step process similar to the authorization code flow, but with user interactions on a separate device (e.g., smartphone or computer).

The choice of authorization flow depends on factors like the type of application (web, mobile, desktop, IoT), the level of trust between the client and the authorization server, and the specific requirements of the use case. Each flow has its own advantages and security considerations. It is essential to select the appropriate flow based on the security and usability needs of the application.

Timeline

Description automatically generated