CSRF,
OAuth 2.0
OpenID Connect and
Session management.

CSRF (Cross-Site Request Forgery)

Definition:

An attack that forces authenticated users to submit a request to a web application against which they're currently authenticated.

Example: CSRF Attack

Scenario: User logs into their bank account.

Attack: Attacker tricks the user into clicking a malicious link.

Outcome: The link executes an unwanted action (e.g., transferring money) using the user's authenticated session.

Prevention:

Anti-CSRF Tokens: Include a unique token in each form submission that the server validates.

Same Site Cookies: Set cookies with the Same Site attribute to prevent them from being sent along with cross-site requests.

OAuth 2.0

Definition:

An authorization framework that enables applications to obtain limited access to user accounts on an HTTP service.

Example: "Login with Google" Feature

Step 1: User clicks "Login with Google" on a third-party site.

Step 2: Google asks the user to grant permissions to the third-party site.

Step 3: If approved, the third-party site receives an access token to access the user's Google data.

Key Concepts:

Roles: Resource Owner, Client, Authorization Server, Resource Server.

Grant Types: Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials.

Tokens: Access tokens and refresh tokens.

OpenID Connect

Definition: An identity layer on top of the OAuth 2.0 protocol, allowing clients to verify the identity of the end-user.

Features:

ID Tokens: JSON Web Tokens (JWT) that contain user identity information.

User Info Endpoint: Provides additional claims about the user.

Scopes: Standard set of scopes for requesting specific information.

Discovery and Dynamic Registration: Mechanisms for clients to discover and register with identity providers.

Session Management

Definition:

The process of securely handling user sessions in web applications.

Example: Secure Session Handling

Generate Unique Session ID: Upon user login, generate a unique session ID.

Store Session Data Server-Side: Keep session data on the server to prevent tampering.

Set Session Timeout: Define a timeout period after which the session expires.

Invalidate Session on Logout: Ensure the session is invalidated when the user logs out.

Best Practices:

Secure, HTTP-Only Cookies: Use cookies that are not accessible via JavaScript.

Proper Logout Mechanisms: Ensure users can log out securely.

Random and Long Session IDs: Use sufficiently random and long session IDs to prevent guessing attacks.