**인증과 권한 관리**

**인증(Authentication)**

• 인증은 사용자나 시스템의 신원을 검증하는 것을 말합니다.

**• 인증 방법**

**• 기본 인증 (Basic Authentication):** 사용자가 ID와 비밀번호를 입력하여 인증하는 방식입니다. HTTP 헤더를 통해 전송됩니다.

**• 토큰 기반 인증 (Token-based Authentication):** 사용자 인증 후 서버가 토큰을 발급하며, 이후 요청에서 토큰을 사용하는 방식입니다. 예로는 JWT (JSON Web Token)이 있습니다.

**• OAuth:** 제3자 애플리케이션이 사용자의 자원에 접근할 수 있도록 허용하는 방식입니다. 권한 부여 코드, 암시적, 자격 증명, 클라이언트 자격 증명 등의 플로우가 있습니다.

**• SSO (Single Sign-On):** 한 번의 로그인으로 여러 애플리케이션에 접근할 수 있는 방식입니다

**기본 인증 (Basic Authentication)**

• 사용자가 HTTP 헤더를 통해 사용자의 ID와 비밀번호를 전송하여 인증하는 방식입니다.

• 사용자 이름과 비밀번호가 Base64로 인코딩되어 전송되며, 암호화는 적용되지 않기 때문에 보안에 취약할 수 있습니다.

• 사용자 ID : 비밀번호 -> Base64 Encoding

• 하지만 HTTPS 에서는 데이터가 SSL/TLS 를 통해 암호하되어 인증 정보가 노출될 위험이 크게 줄어듭니다.

• 매우 간단하고 구현이 쉬운 방식입니다.

**세션 기반 인증**

• 사용자가 로그인하면 서버가 세션을 생성하고, 클라이언트는 쿠키를 통해 세션 ID를 유지하는 방식입니다.

**쿠키와 세션의 개념**

**• 쿠키:** 클라이언트 측에 저장되는 작은 데이터 파일로, 사용자의 상태 정보를 유지합니다. 쿠키에는 세션 ID와 같은 인증 정보가 포함될 수 있습니다.

**• 세션:** 서버 측에 저장되는 사용자 정보로, 사용자의 상태를 관리합니다. 세션은 고유한 세션 ID로 식별됩니다.

**세션 관리 메커니즘**

• 사용자가 로그인하면 서버가 고유한 세션 ID를 생성하고, 이를 클라이언트에 쿠키로 전송합니다.

• 클라이언트는 이후 모든 요청에서 쿠키를 통해 세션 ID를 서버에 전송하여 인증을 유지합니다.

• 서버는 세션 ID를 통해 사용자의 상태를 식별하고 관리합니다.

• 세션은 서버 측에 저장되므로, 서버 자원을 많이 사용합니다.

**세션 기반 인증 시, 서버에서 인증을 확인하는 방법**

1. **세션 생성**

• 서버는 고유한 세션 ID를 생성합니다.

• 세션 ID는 사용자 정보(아이디, 권한 등)를 포함한 세션 데이터와 함께 서버의 메모리, 데이터베이스 또는 다른 저장소에 저장됩니다.

1. **세션 ID를 클라이언트에 전달**

• 서버는 세션 ID를 클라이언트에 쿠키로 전달합니다.

• 클라이언트는 이 세션 ID를 브라우저 쿠키에 저장합니다.

1. **서버에서 세션 ID 검증**

• 서버는 클라이언트 요청을 받을 때, 요청에 포함된 세션 ID를 확인합니다.

• 세션 ID를 사용하여 서버에 저장된 세션 데이터와 비교합니다.

1. **인증 확인 및 사용자 정보 로드**

• 세션 ID가 유효하지 않다면 인증 실패를 반환합니다.

**세션 기반 인증의 장단점**

**• 장점**

**• 보안**: 세션 기반 인증은 클라이언트에게 세션 ID만 전송하므로 보안이 강화됩니다. 실제 데이터를 주고받지 않기 때문에 중간자 공격에 강합니다.

**• 서버 부담 감소**: 세션 ID만을 클라이언트에 전송하므로 서버에 저장된 세션 데이터를 통해 인증을 유지할 수 있어 서버 부하가 감소합니다.

**• 탈취 방지:** 세션 ID는 브라우저의 쿠키에 저장되며, 이는 XSS(Cross-Site Scripting) 공격으로부터 보호됩니다.

**• 단점**

**• 상태 유지:** 세션 기반 인증은 서버에 상태를 유지해야 하므로, 확장성이 떨어질 수 있습니다. 사용자가 많아지면 서버에 부하가 걸릴 수 있습니다.

• **확장성:** 서버의 스케일링이 어려울 수 있으며, 서버 장애 시 세션 정보의 손실 문제가 발생할 수 있습니다.

**토큰 기반 인증 (Token-based Authentication)**

• 사용자가 인증되면 서버가 토큰을 발급하고, 클라이언트는 이후 요청에서 토큰을 사용하는 방식입니다.

• 토큰은 클라이언트 측에 저장되며, 각 요청 시 서버에 전송됩니다.

• 서버는 세션 정보를 저장하지 않으며, 클러이언트가 전송한 토큰을 검증하여 사용자를 식별합니다.

• 토큰을 발급하는 예로는 JWT (JSON Web Token) 등이 있습니다.

**토큰 기반 인증의 장단점**

**• 장점**

**• 상태 비유지(Stateless):** 토큰 기반 인증은 클라이언트 측에서 토큰을 저장하고 유지하기 때문에 서버에 상태를 저장할 필요가 없어 확장성이 좋습니다.

**• 사용자 경험:** 토큰은 클라이언트 측에서 저장되기 때문에 매 요청마다 인증 정보를 서버에 보내지 않아도 되어 빠른 인증 속도를 제공합니다.

**• 분리된 인증 서비스:** 토큰 기반 인증은 OAuth와 같은 프로토콜을 사용하여 인증 서버와 리소스 서버를 분리할 수 있습니다.

**• 단점**

**• 보안 문제:** 토큰이 클라이언트 측에 저장되므로, 안전하게 저장되지 않을 경우 탈취될 위험이 있습니다. 이를 보호하기 위해 HTTPS를 사용해야 합니다.

**• 복잡성:** 토큰 기반 인증은 추가적인 설정과 관리가 필요할 수 있으며, 초기 구현에 있어서는 세션 기반 인증보다 더 복잡할 수 있습니다.

**토큰 기반 인증의 종류**

1. **JWT (JSON Web Token)**

• JWT는 클레임(claim)을 포함하는 JSON 객체로 표현되는 암호화된 토큰입니다.

• JWT는 헤더(Header), 페이로드(Payload), 서명(Signature) 으로 이루어져 있으며, 각 요소는 . 으로 구분하여 작성합니다.

• 사용자 인증 및 정보 교환에 사용됩니다. 토큰 안에는 사용자 ID, 권한, 만료 시간 등의 정보가 포함됩니다.

• 서버에서 발급된 토큰은 클라이언트에 저장되어 각 요청마다 함께 전송되며, 서버는 토큰의 유효성을 검증하여 사용자를 인증합니다

1. **페이로드(Payload): 공개 클레임 (Public Claims)**

• 공개 클레임은 JWT 표준에 정의된 것은 아니지만, 일반적으로 사용되는 클레임들입니다.

• 등록된 클레임은 JWT 표준에 명시적으로 정의되어 있지만, 공개 클레임은 표준이 아니며 특정 시스템이나 애플리케이션에서 필요한 정보를 나타내기 위해 사용됩니다.

1. **서명(Signature)**

• JWT의 서명은 헤더와 페이로드를 결합한 후, 선택된 알고리즘을 사용하여 생성됩니다.

• 서명은 토큰이 변조되지 않았음을 검증하는 데 사용됩니다.

• 예를 들어, HMAC SHA-256 알고리즘을 사용하여 헤더와 페이로드를 서명한 후, 이 서명은 JWT의 세 번째 부분으로 추가됩니다.

1. **OAuth (Open Authorization)**

• 웹 및 모바일 응용 프로그램을 위한 표준 인증 프로토콜입니다.

• 사용자가 리소스 소유자의 인증 정보를 제공하지 않고도 다른 웹 사이트 또는 앱에 접근 권한을 부여할 수 있는 방법을 제공합니다.

• **인증(Authorization)**

• OAuth는 인증 정보를 제공하지 않고도 사용자가 다른 서비스 또는 애플리케이션에 대한 접근 권한을 부여할 수 있는 인증 메커니즘을 제공합니다.

• 이는 사용자의 비밀번호와 같은 민감한 정보를 노출하지 않고도, 외부 서비스에 대한 접근 권한을 관리할 수 있도록 합니다.

• **인가(Authorization**)

• OAuth는 사용자가 특정 리소스에 대한 액세스 권한을 부여하거나 거부할 수 있는 기능을 제공합니다.

• 이를 통해 애플리케이션은 사용자가 선택한 권한 범위 내에서만 데이터에 접근할 수 있습니다.

**OAuth (Open Authorization) 작동 방식**

• OAuth는 주로 클라이언트 애플리케이션이 인증 서버를 통해 리소스 서버에 접근할 수 있도록 하는 인증 프로토콜입니다.

**1. 리소스 소유자(Resource Owner):** 자원에 접근하려는 사용자를 의미합니다. 예를 들어, 소셜 미디어 사이트에서 사용자는 자신의 데이터에 대한 접근 권한을 제어할 수 있습니다.

**2. 클라이언트(Client):** 사용자의 자원에 접근하려는 애플리케이션 또는 서비스를 의미합니다. 예를 들어, 소셜 미디어 앱이나 웹 사이트가 클라이언트가 될 수 있습니다.

**3. 인증 서버(Authentication Server):** 사용자를 인증하고 권한 부여 코드(Authorization Code)를 발급하는 서버입니다. 사용자는 인증 서버를 통해 클라이언트에게 접근 권한을 부여하게 됩니다.

**4. 리소스 서버(Resource Server):** 클라이언트가 접근하려는 사용자의 자원을 호스팅하는 서버입니다. 예를 들어, 사용자의 사진이나 프로필 정보를 제공하는 서버가 리소스 서버가 될 수 있습니다.

**4. SSO (Single Sign-On)**

• SSO(Single Sign-On) 는 하나의 인증을 통해 여러 시스템 또는 응용 프로그램에 접근할 수 있는 인증 메커니즘을 의미합니다.

• 사용자는 단일 인증 과정을 거치면서 다양한 서비스에 접근할 수 있으며, 각 서비스마다 별도로 로그인할 필요가 없습니다.

**주요 특징**

**• 단일 인증:** 사용자는 첫 번째 애플리케이션에 로그인한 후 다른 모든 연결된 애플리케이션에 대해 추가적인 인증 없이 접근할 수 있습니다. 이는 사용자의 생산성을 높이고 비밀번호를 기억해야 하는 부담을 줄여줍니다.

**• 중앙화된 인증 관리:** SSO는 보통 중앙화된 인증 서비스나 ID 관리 시스템을 통해 작동합니다. 사용자의 인증 정보는 한 곳에서 관리되고, 다양한 서비스에 전파됩니다.

**• 사용자 경험 향상:** 사용자는 여러 서비스 간의 시스템 전환 없이, 통합된 사용자 경험을 제공받을 수 있습니다. 이는 시간 절약 및 혼란을 줄여줍니다.

**SSO 구현 방식**

**• SAML (Security Assertion Markup Language):** 기업 환경에서 널리 사용되는 프로토콜로, XML 기반의 인증 및 권한 부여 데이터를 전달합니다.

**• OAuth 및 OpenID Connect:** 웹 및 모바일 애플리케이션에서 SSO를 구현하는 데 널리 사용되는 프로토콜이며, 클라이언트와 서버 간의 인증 및 인가를 담당합니다.

**• LDAP (Lightweight Directory Access Protocol):** 기업 내부 인프라에서 사용자 인증 및 디렉토리 서비스를 관리하는 데 사용되며, SSO의 일환으로 구현될 수 있습니다.

**토큰 기반 인증 시, 서버에서 인증을 확인하는 방법**

1. **토큰 생성**

• 서버에서 사용자에 대한 정보를 포함한 토큰을 생성합니다.

• 이 토큰은 보통 사용자 ID, 권한, 토큰 만료 시간 등의 정보를 포함합니다.

• JWT의 경우, 토큰은 세 부분(헤더, 페이로드, 서명)으로 구성됩니다.

1. **토큰을 클라이언트에 전달**

• 생성된 토큰을 클라이언트에게 반환합니다.

• 클라이언트는 이 토큰을 저장하고, 이후 요청 시 포함하여 서버로 보냅니다.

1. **서버에서 토큰 검증**

• 서버는 클라이언트의 요청을 받을 때마다 토큰을 확인합니다.

• 토큰 형식 확인 -> 서명 검증 -> 토큰 만료 시간 확인 -> 사용자 권한 정보 확인

1. **인증 확인 및 사용자 정보 로드**

• 세션 ID가 유효하지 않다면 인증 실패를 반환합니다

**권한 관리(Authorization)**

**권한 관리 (Authorization)**

• 권한 관리는 사용자가 특정 자원이나 기능에 접근할 수 있는 권한을 부여하고 관리하는 프로세스입니다.

• 이는 정보 시스템에서 데이터 보호와 사용자 권한을 효과적으로 관리하기 위해 중요한 요소입니다.

**권한 관리의 주요 개념**

**• 권한 (Permission):** 특정 자원이나 기능에 대한 접근을 허용하는 권한을 나타냅니다.

**• 역할 (Role):** 사용자 그룹에 할당되는 권한의 집합으로, 비슷한 업무를 수행하는 사용자들을 그룹화하여 권한을 관리합니다.

**• 정책 (Policy):** 권한을 관리하기 위해 설정된 규칙이나 지침으로, 어떤 사용자가 어떤 자원에 접근할 수 있는지를 결정합니다.

**역할 기반 접근 제어 (RBAC)**

• 역할 기반 접근 제어(Role-Based Access Control, RBAC)는 사용자가 수행하는 역할에 기반하여 접근 권한을 관리하는 접근 제어 모델입니다.

• 이 모델은 사용자의 업무 기능에 따라 그룹화된 역할을 할당하고, 이 역할에 따라 특정 자원이나 기능에 접근할 수 있는 권한을 부여합니다.

**RBAC의 작동 방식**

• **역할 할당(Role Assignment):** 관리자는 각 사용자에게 적절한 역할을 할당합니다. 이를 통해 사용자는 자신의 역할에 따라 접근 권한을 획득합니다.

• **권한 부여(Permission Grant):** 각 역할에는 특정 자원이나 기능에 대한 권한이 할당됩니다. 역할을 가진 사용자는 해당 권한을 사용하여 시스템의 자원에 접근할 수 있습니다.

**• 역할 관리(Role Management): 시스템의** 변화에 따라 역할을 추가, 수정 또는 삭제하여 접근 제어를 유지하고 관리합니다.

**역할 기반 접근 제어 (RBAC) 장점**

**• 관리 용이성:** 역할 기반 접근 제어는 역할을 기준으로 권한을 관리하므로, 사용자별로 직접적으로 권한을 설정할 필요가 없어 관리가 용이합니다.

**• 보안 강화:** 사용자의 역할에 따라 접근할 수 있는 자원이 명확히 정의되어 있어 보안을 강화할 수 있습니다.

**• 비즈니스 요구 사항 충족:** 비즈니스의 변화에 따라 역할과 권한을 조정하여 요구 사항에 맞추기 쉽습니다.

**• 적용 예시**

**• 기업 내부 시스템:** 직무 기능에 따라 역할을 할당하여 데이터 접근을 제어합니다.

**• 금융 서비스:** 고객 서비스 담당자, 관리자, 재무 담당자 등 다양한 역할에 따라 접근 권한을 부여하여 보안을 강화합니다.

**• 교육 기관:** 학생, 교수, 관리자 등 각각의 역할에 따라 시스템 자원에 접근할 수 있는 범위를 제한합니다.

**속성 기반 접근 제어 (ABAC)**

**•** 속성 기반 접근 제어(Attribute-Based Access Control, ABAC)는 사용자의 속성, 대상의 속성, 환경 조건 등 다양한 속성을 기반으로 접근 권한을 결정하는 접근 제어 모델입니다.

• ABAC는 사용자의 특성과 상황을 더욱 동적으로 반영하여 접근 제어를 수행합니다.

**ABAC의 작동 방식**

• **속성 수집(Attribute Collection):** ABAC는 다양한 속성을 수집하여 평가에 사용합니다. 이는 사용자의 프로필, 세션 정보, 요청된 자원의 속성 등을 포함할 수 있습니다.

**• 정책 평가(Policy Evaluation):** 정책에 정의된 규칙과 속성을 비교하여 사용자가 요청한 자원에 대한 접근을 허용할지 여부를 결정합니다. 예를 들어, 특정 역할과 지리적 위치가 정책에 부합하는지 확인할 수 있습니다.

**• 동적 접근 제어(Dynamic Access Control):** ABAC는 사용자의 상황에 따라 동적으로 접근 권한을 결정할 수 있습니다. 예를 들어, 특정 시간대에만 특정 데이터에 접근할 수 있도록 제한할 수 있습니다.

**속성 기반 접근 제어 (ABAC) 장점**

**• 동적 접근 제어:** 사용자의 상황에 따라 접근 권한을 동적으로 조정할 수 있어 유연성이 높습니다.

**• 정확성과 정밀성:** 다양한 속성을 고려하여 정확하고 정밀한 접근 제어를 제공합니다.

**• 복잡한 환경 처리:** 다양한 조직 내 속성과 정책을 효과적으로 관리할 수 있습니다.

**• 적용 예시**

**• 클라우드 환경:** 다양한 사용자와 자원 간의 접근을 유연하게 관리하여 보안을 강화합니다.

**• 금융 서비스:** 다양한 고객 그룹에 대한 접근 권한을 정밀하게 제어하여 개인정보 보호를 강화합니다.

**인증 보안관리**

**브루트 포스 공격 (Brute Force Attack)**

• 모든 가능한 조합을 시도하여 인증을 뚫으려는 공격.

**• 대응 방안**

**• 강력한 비밀번호 정책:** 길이와 복잡성 요구 사항을 설정하여 암호의 예측 불가능성을 증가시킵니다.

**• 계정 잠금 기능:** 일정 시도 횟수 이상 실패한 계정을 일시적으로 잠금 처리하거나, CAPTCHA와 같은 추가적인 인증 요구를 적용합니다.

**• IP 차단:** 일정 IP 범위에서 너무 많은 요청이 발생할 경우 해당 IP를 일시적으로 차단합니다.

**세션 하이재킹 (Session Hijacking)**

• 사용자의 세션 ID를 탈취하여 해당 세션으로 접근하려는 공격.

**• 대응 방안**

**• HTTPS 사용:** 데이터를 암호화하여 중간자 공격을 방지하고, 세션 데이터의 안전성을 보장합니다.

**• 세션 유효 시간 관리:** 사용자의 비활동 시간 동안 세션을 자동으로 만료시키거나, 사용자가 로그아웃할 때까지 세션을 유지하는 방법을 채택합니다.

**• 세션 관리:** 보안 쿠키를 사용하거나, 사용자의 장치 및 IP 주소와 같은 추가 정보를 고려하여 세션을 관리하고 보호합니다.

**암호화 및 해싱**

**암호화 (Encryption)**

• 암호화는 데이터를 암호화하여 외부에서 이해할 수 없는 형태로 변환하는 과정을 말합니다.

• 주로 데이터의 기밀성을 보호하기 위해 사용됩니다.

• 대표적인 암호화 기법으로는 대칭키 암호화와 비대칭키 암호화가 있습니다.

• **대칭키 암호화**: 암호화와 복호화에 동일한 키를 사용하는 방식입니다. 데이터 전송 시에 키를 공유해야 하지만, 빠르고 효율적입니다.

• 예시: AES (Advanced Encryption Standard)

• **비대칭키 암호화:** 암호화와 복호화에 서로 다른 키를 사용하는 방식으로, 공개키와 개인키를 함께 사용합니다. 보안성이 뛰어나지만 연산 비용이 높을 수 있습니다.

• 예시: RSA (Rivest-Shamir-Adleman)

**해싱 (Hashing)**

• 해싱은 임의의 길이를 가진 데이터를 고정된 길이의 데이터로 매핑하는 알고리즘입니다.

• 해시 함수는 동일한 입력에 대해 항상 동일한 출력을 생성하며, 원본 데이터의 무결성을 검증하거나 비밀번호와 같은 데이터를 안전하게 저장하는 데 사용됩니다.

• 예시: SHA-256 (Secure Hash Algorithm 256-bit)

**• 비밀번호 해싱**

• 사용자 비밀번호와 같은 중요 정보는 단방향 해시 함수를 사용하여 저장합니다.

• 이는 데이터베이스에 실제 비밀번호를 저장하지 않고, 로그인 시 입력된 비밀번호를 해시하여 저장된 해시 값과 비교합니다.

• 해시 함수에 솔팅(Salting) 기법을 추가하여 보안성을 더욱 강화할 수 있습니다.

**• 비밀번호 해싱의 중요성**

• **보안 강화:** 해싱을 통해 비밀번호를 안전하게 보호하여 데이터 유출 시에도 비밀번호 노출을 최소화합니다.

• 레인보우 테이블 방지: 솔팅 기법을 사용하여 레인보우 테이블 공격을 방지합니다.

**• 규정 준수:** 개인정보 보호 법규에 따른 보안 요구사항을 준수하여 사용자의 개인정보를 안전하게 관리합니다.