JWT(Json Web Token) 인증 방식은 클라이언트와 서버 간의 안전한 인증을 위한 토큰 기반 인증 방법이다. JWT는 주로 사용자 인증과 권한 부여에서 사용되며 세션을 서버에 저장하지 않고 사용자 정보를 담은 토큰을 클라이언트 측에 저장해 사용하는 방식이다.

JWT의 구조

1. 헤더(Header) : 토큰의 형식(JWT)과 사용된 알고리즘 등에 대한 정보를 담고 있다.
2. 페이로드(Payload) : 실제 사용자 정보를 담는 부분이다. 사용자 ID, 이름, 권한 등 다양한 정보를 포함할 수 있다.
3. 서명(Signature) : 헤더와 페이로드를 연결하고 토큰의 무결성을 보장하기 위한 부분이다. 비밀 키를 사용하여 해시 함수를 적용하여 생성된다.

JWT 형식 예시

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9
```

JWT의 저장 위치

1. 로컬 스토리지 vs 쿠키
    1. 로컬 스토리지 : 브라우저의 로컬 스토리지에 저장되면 서버와의 통신에 자동으로 포함되지 않아 사용자가 명시적으로 토큰을 추가해야 한다. 하지만 로컬 스토리지는 XSS(크로스 사이트 스크립팅) 공격에 취약할 수 있다.
    2. 쿠키 : 쿠키에 저장할 경우 `HttpOnly` 속성과 `Secure` 속성을 설정해 XSS와 CSRF(크로스 사이트 요청 위조) 공격을 방지할 수 있다. 하지만 CSRF에 취약할 수 있어 CSRF 방어 전략을 함께 적용하는 것이 중요하다.

JWT 무효화 전략

1. JWT는 자체적으로 상태를 저장하지 않기 때문에 토큰 발급 후 서버에서 무효화하기 어렵다. 이를 해결하기 위해 몇 가지 방법이 있다
    1. 블랙리스트 : 무효화할 JWT를 서버에 저장하고 매 요청마다 토큰을 블랙리스트와 비교하여 차단한다. 그러나 이 방식은 서버의 상태를 저장하게 되어 JWT의 무상태 성격과 모순된다.
    2. 짧은 만료 시간과 리프레시 토큰 사용 : JWT의 만료 시간을 짧게 설정하고 리프레시 토큰을 이용하여 만료된 JWT를 갱신하는 방식이다. 이 방법은 리프레시 토큰을 서버에 저장하여 보다 유연하게 무효화할 수 있다.

JWT 암호화

기본적으로 JWT는 서명을 통해 무결성(변조 방지)을 보장하지만 Payload에 담긴 정보는 누구나

열람할 수 있다. 기밀 데이터가 포함된 경우 JWT를 JWE(Json Web Encryption) 형식으로 암호화하여 전송하는 것이 필요할 수 있다.

알고리즘 선택

1. HS256 (HMAC with SHA-256) : 비밀 키를 사용한 대칭 암호화 방식으로 성능이 좋고 사용이 간단하지만 키가 유출되면 쉽게 공격받을 수 있다.
2. RS256 (RSA Signature with SHA-256) : 비대칭 암호화 방식으로 공개 키와 비밀 키를 사용한다. 서명 검증은 공개 키로 할 수 있어 보안성이 뛰어나지만 성능이 떨어질 수 있습니다.

OAuth 2.0과의 관계

1. JWT는 주로 OAuth 2.0과 함께 사용된다. OAuth 2.0은 권한 부여를 위한 프로토콜이고 JWT는 이러한 프로토콜에서 인증을 위한 토큰으로 사용된다. OAuth 2.0에서 JWT는 **Access Token**이나 **ID Token**으로 활용된다.
2. OAuth 2.0의 **Implicit Flow**(브라우저 기반 애플리케이션에 사용)나 **Authorization Code Flow**(일반적인 웹 애플리케이션 인증에 사용)에서 JWT를 발급하여 클라이언트가 권한을 사용할 수 있도록 하며 이때 리프레시 토큰과 함께 사용해 보안성을 강화할 수 있다.

JWT와 OAuth 2.0 사용 시 보안 고려사항

1. 리프레시 토큰 보안 : 리프레시 토큰은 더 긴 만료 시간을 갖고 새로운 액세스 토큰을 발급하는 데 사용되기 때문에 안전하게 관리해야 한다. 리프레시 토큰도 탈취되면 동일한 위험이 발생할 수 있다.
2. Scope와 권한 관리 : JWT에 담긴 권한(scope) 정보가 잘못 관리되면 불필요하게 높은 권한을 부여받은 토큰이 발급될 수 있다. 따라서 Scope를 적절히 설정하고 관리해야 한다.

JWT 활용 예시

1. 마이크로서비스 아키텍처 : 각 서비스가 분산되어 있을 때, JWT는 무상태 특성 덕분에 각 서비스 간 인증 정보를 공유하는 데 용이하다.
2. 소셜 로그인 : Google, Facebook 등의 소셜 로그인 서비스에서 JWT를 Access Token으로 활용하여 인증을 통합적으로 처리할 수 있다.

JWT 서드 파티 라이브러리

1. 다양한 언어에서 JWT를 다루기 위한 라이브러리가 있다. jsonwebtoken (Node.js), pyjwt (Python), jwt-go (Go), jjwt (Java JWT) 등이 있다.

JWT 와 OAuth 차이점

항목	JWT	OAuth 2.0
개념	토큰 포맷(인증 정보를 담아 전송)	권한 부여 프로토콜(제3자 권한 부여 관리)

항목	JWT	OAuth 2.0
역할	인증 정보와 권한을 포함한 토큰 전달	클라이언트에게 자원에 대한 권한 부여
토큰 형식	JWT 자체가 토큰 형식으로 사용	OAuth 2.0은 JWT와 같은 토큰을 사용하거나 다른 토큰 형식도 사용 가능
목적	사용자 인증 및 권한 정보 전달	제3자에게 특정 자원에 대한 접근 권한 부여
토큰 저장	클라이언트가 JWT를 로컬 스토리지나 쿠키에 저장	OAuth는 서버에서 액세스 토큰을 발급하고 관리
사용 사례	API 인증, 사용자 로그인	소셜 로그인, 외부 앱의 API 권한 부여
무효화	JWT는 자체적으로 상태를 관리하지 않아서 무효화가 어려움	OAuth 2.0은 리프레시 토큰으로 토큰 갱신 가능
서드 파티	JWT는 서버와 클라이언트 간의 인증에 사용	OAuth 2.0은 제3자 클라이언트에 권한 부여를 중점으로 함
보안	JWT는 암호화되지 않고, 서명을 통해 변조를 방지	OAuth 2.0은 토큰을 관리하며, 토큰 만료 및 갱신을 통해 보안을 강화
권한 관리	JWT는 권한 정보(scope)를 직접 포함할 수 있음	OAuth 2.0은 권한 범위(scope)를 명시적으로 관리

1. JWT는 인증 정보나 사용자 권한을 포함한 토큰 형식으로 주로 사용되며 인증 과정에서 서버와 클라이언트 간에 정보를 주고받는 데 유용
2. OAuth 2.0은 사용자로부터 권한을 받아 제3자 클라이언트가 자원에 접근할 수 있게 해주는 권한 부여 프로토콜이다. OAuth 2.0 자체가 토큰 형식을 정의하지는 않지만 JWT와 같은 토큰을 활용할 수 있다.

JWT 인증 과정

1. 사용자 로그인 : 사용자가 로그인 요청을 보내면 서버는 사용자의 자격 증명을 확인하고 확인이 완료되면 사용자 정보와 만료 시간 등의 정보를 담은 JWT를 생성한다.
2. 토큰 발급 : 서버는 JWT를 클라이언트에게 응답으로 반환한다. 이때 서버는 JWT를 자체적으로 저장하지 않는다.
3. 클라이언트 저장 : 클라이언트는 발급받은 JWT를 로컬 스토리지나 쿠키에 저장한다.
4. 인증 요청 : 클라이언트는 이후 인증이 필요한 요청(예시 : API 요청) 시, JWT를 HTTP 헤더(보통 `Authorization: Bearer <토큰>` 형식)로 서버에 전송한다.
5. 토큰 검증 : 서버는 요청을 받을 때 JWT의 유효성을 검증한다. 검증 과정은 Header와 Payload의 정보가 변경되지 않았는지 서명이 올바른지 만료 시간이 지났는지를 확인하는 과정이다.
6. 요청 처리 : 토큰이 유효하면 서버는 해당 사용자의 정보를 추출하여 요청을 처리하고 유효하지 않으면 인증 오류를 반환한다.

JWT의 장점

1. 상태 비저장 : 서버 측에서 세션을 저장할 필요가 없으므로 서버 확장이 용이하다.
2. 확장성 : API 간의 인증에서 특히 유용하며 분산 시스템에서도 쉽게 사용할 수 있다.
3. 편리한 데이터 전송 : Payload에 사용자 권한 정보나 만료 시간 등을 포함할 수 있어 추가적인 DB 조회 없이도 클라이언트 요청을 처리할 수 있다.

JWT의 단점

1. 크기 : JWT는 자체적으로 사용자 정보를 담고 있기 때문에 일반적인 세션 ID보다 크기가 크다.
2. 보안 : 클라이언트 측에 저장된 토큰이 유출될 경우 유효 기간 동안 악용될 수 있다. 따라서 HTTPS와 같은 안전한 통신 채널을 통해 전송해야 하며 토큰의 만료 시간 설정도 중요하다.
3. 토큰 무효화 어려움 : 서버에서 JWT를 무효화하는 것은 어렵다. 서버는 JWT 자체를 저장하지 않기 때문에 유효 기간 내에서는 무효화할 수 있는 방법이 제한적이다.

보안 고려 사항

1. HTTPS 사용 : JWT는 클라이언트에 저장되기 때문에 토큰을 탈취당하지 않도록 반드시 HTTPS를 통해 통신해야 한다.
2. 토큰 만료 시간 설정 : 짧은 만료 시간을 설정하여 만료 후 새로운 토큰을 발급하는 것이 좋다.
3. 리프레시 토큰 사용 : 만료된 JWT를 갱신하기 위해 별도의 리프레시 토큰을 사용하여 보안을 강화할 수 있다.

이렇게 JWT는 서버 확장성, API 기반 인증에서 매우 유용하지만 보안적인 측면에서 주의해야 할 부분도 많다.