Chapter 11 보안 : 인증과 SPA

클라우드 네이티브 스프링 인 액션 스터디

이동준

식별, 인증, 인가 비교

Authorization Code는 인증 코드가 아니라 인가 코드

개념	식별(Identification)	인증(Authentication)	인가(Authorization)
정의	사용자를 시스템에 알리는 과정	**사용자** 확인하는 과정	**자원**에 접근할 권한 확인 과정
예시	사용자 ID, 이메일	비밀번호, OTP, 생체 인식	관리자/일반 사용자 권한 설정
목적	사용자 구별	사용자의 신원 검증	접근 가능한 자원 제한

KeyCloak

1. KeyCloak, OAuth2, OpenID Connect(OIDC) 비교
2. KeyCloak
    i. 오픈소스 아이디 및 접근 관리(Identity and Access Management, IAM) **솔루션**
    ii. 다양한 애플리케이션에 대한 중앙 집중식 인증 및 권한 관리를 가능하게 하는 인증 서버
3. OAuth2
    i. 자원 서버의 접근 권한을 타 애플리케이션에 위임할 수 있는 **인가 프레임워크**
    ii. 주요 행위자 : 인증 서버, 사용자, 클라이언트
4. OpenID Connect (OIDC)
    i. **OAuth2 기반 인증 프로토콜**로, 사용자의 신원을 확인하고 인증을 제공하기 위한 표준
    ii. 주요 행위자 : OIDC 공급자, 엔드 유저, 신뢰 당사자

Keycloak, OAuth2, OIDC 주요 행위자 비교

OIDC는 OAuth2 프레임워크를 기반으로 한 프로토콜

OAuth2	OIDC	역할
인증 서버(Authorization Server)	OIDC 공급자 (OIDC Provider)	토큰을 발급하는 서버(Keycloak). 사용자의 동의에 따라 클라이언트에 액세스 토큰을 발급.
사용자(User / Resource Owner)	엔드 유저(End User)	자원의 소유자. 자신의 신원을 클라이언트에 인증하여 제공하는 주체
클라이언트(Client)	신뢰 당사자(Relying Party)	인증 요청 수행자. ID 토큰과 액세스 토큰을 통해 사용자의 신원 확인 및 자원 접근 수행

주요 토큰 비교

액세스 토큰과 리프레시 토큰은 OAuth 2.0, ID 토큰은 OIDC 로 발급

토큰 종류	설명	목적	발급 대상
엑세스 토큰	리소스 서버에 요청을 보낼 때 권한을 인증하는 토큰	보호된 리소스에 접근할 수 있도록 **인가**	클라이언트 (리소스 서버에 사용)
리프레시 토큰	엑세스 토큰이 만료되었을 때 새로운 엑세스 토큰을 발급받기 위한 토큰	액세스 토큰의 만료에 따른 재인증을 최소화	클라이언트 (인증 서버와 교환하여 발급받음)
ID 토큰	사용자의 신원을 증명하는 토큰 (사용자 정보 포함)	클라이언트가 사용자를 **인증**하고 사용자 정보를 확인	클라이언트 (주로 사용자 로그인에 사용)

OIDC 프로토콜이 지원하는 인증 흐름(그림 11.3)

구분	작업	상세
사용자	클라이언트로 작업 요청	클라이언트로 작업 요청 시 keyCloak으로 302
브라우저	사용자에게 KeyCloak 화면 전환	아이디와 패스워드 로 로그인 성공
KeyCloak	클라이언트로 302	인가 코드를 포함하여 클라이언트로 302
클라이언트	KeyCloak에게 인증 요청	KeyCloak에게 받은 인가 코드를 재요청
KeyCloak	클라이언트로 ID 토큰 제공	인가 코드를 이용해 클라이언트에게 ID 토큰 제공
브라우저	ID 토큰 세션 저장	ID 토큰을 세션에 저장
클라이언트	API 서버로 작업 요청	API 서버에 세션 정보와 함께 작업 요청

- 인가 코드를 이용해 다시 한번 ID 토큰을 재요청함으로써 작업 분리(암시적 허가 제한)

OIDC 프로토콜이 지원하는 로그아웃 흐름(그림 11.6)

구분	작업	상세
사용자	클라이언트로 로그아웃 요청	클라이언트로 세션 삭제 요청
클라이언트	브라우저 세션 삭제	브라우저 세션 삭제
클라이언트	키클록으로 로그아웃 요청	토큰을 무효화
브라우저	클라이언트로 302	로그인 페이지 노출

- 클라이언트(엣지 서버)는 로그아웃 요청 시 로그아웃 후 302가 될 URL을 미리 지정
- OIDC 프로토콜은 로그아웃 한 다음에 리다이렉트할 URL을 지정하는 것이 표준

프론트엔드 SPA를 적용했을 때 인증 흐름(11.5.2 절)

구분	작업	상세
사용자	클라이언트로 작업 요청	302가 아니라 스프링 시큐리티가 **401** 응답
클라이언트	**브라우저**가 KeyCloak 화면 전환	로그인 페이지로 이동하는 URL을 브라우저에 명령
브라우저	Keycloak 로그인 페이지로 이동	사전 정의된 URL(`/oauth2/authorization/keycloak`)
사용자	Keycloak 로그인	Keycloak 로그인 페이지에서 아이디와 비밀번호 입력

- Keycloak에서 로그인 성공한 다음에는 인가 코드, ID 토큰을 이용한 인증은 동일
- AJAX 요청 대신 브라우저의 표준 HTTP 요청을 통해 인증이 시작
- 스프링 시큐리티는 OAuth2/OIDC를 기반으로 인증 흐름을 할 때 사용하는 `/auth2/authorization/{registrationId}` 엔드포인트를 제공

OAuth 2를 구현하는 방법

허가 유형	설명	비고
인가 코드 (Authorization Code)	사용자가 인가 코드를 통해 인증 서버에서 액세스 토큰을 요청하는 방식	보안성이 높은 편
암호(Password)	사용자 자격 증명을 클라이언트에 직접 전달하여 액세스 토큰을 요청하는 방식	보안성이 낮은 편
리프레시 토큰 (Refresh Token)	기존 액세스 토큰이 만료된 후 새로운 액세스 토큰을 요청하는 방식	유효 기간 연장, 서비스 연속성 유지
클라이언트 자격 증명 (Client Credentials)	클라이언트가 직접 인증 서버로부터 액세스 토큰을 요청하는 방식	사용자 개입 없이 서버 간 통신에 유용

암시적 허가(Implicit Grant)

1. 암시적 허가

- OAuth2에서 사용자가 인가 코드 교환 과정 없이 직접 액세스 토큰을 요청하는 인증 방식
- **클라이언트 측 애플리케이션(예: SPA, 단일 페이지 애플리케이션)** 같은 서버가 없는 애플리케이션에 주로 사용

2. 암시적 허가의 특징

- 간편한 인증 과정: 인가 코드 교환 과정이 없고, 액세스 토큰을 바로 요청
  - 클라이언트 측 애플리케이션에 적합: 주로 **단일 페이지 애플리케이션(SPA)** 에 사용
- 보안 위험: 액세스 토큰이 브라우저에 직접 노출되기 때문에 가로채기 가능
  - 토큰이 바로 브라우저에 제공되므로 브라우저 히스토리에 저장될 수 있어 보안에 취약
- 리프레시 토큰을 사용하지 않으며, 대개 액세스 토큰의 유효 기간을 짧게 설정

OAuth2에 존재하는 보안상 허점

허점	설명	대응 방안
토큰 유출	토큰이 노출되면, 이를 탈취한 공격자가 사용자 권한으로 자원에 접근 가능	- 짧은 토큰 유효 기간 - PKCE 사용
인가 코드 가로채기	공격자가 인가 코드를 가로채 클라이언트 대신 액세스 토큰을 발급받는 공격	- PKCE 사용
암시적 허가	암시적 허가는 인가 코드 교환 없이 토큰이 브라우저에 직접 노출되어 보안에 취약	- 인가 코드 + PKCE 사용으로 대체
CSRF(Cross-Site Request Forgery)	악성 사이트가 사용자의 인증 세션을 악용해 사용자의 자격으로 인증 서버에 접근	- CSRF 방지 토큰

PKCE

1. 정의
   PKCE(Proof Key for Code Exchange) 는 OAuth2의 인가 코드(Authorization Code) 흐름에서 보안성을 높이기 위해 추가된 메커니즘

2. PKCE의 작동 원리

- 클라이언트에서 코드 검증값 생성
  - 클라이언트는 임의의 난수 문자열인 **코드 검증값(Code Verifier)** 을 생성
    코드 검증값은 사용자가 인증 서버로 리디렉트되기 전에 클라이언트에서 생성
- 코드 챌린지 생성 및 전송
  - 클라이언트는 코드 검증값을 SHA-256 해싱하여 **코드 챌린지(Code Challenge)** 를 생성하고, 인가 요청 시 코드 챌린지를 인증 서버에 전달
- 인증 서버에서 인가 코드 발급
  - 사용자가 인증을 완료하면, 인증 서버는 인가 코드를 클라이언트에 반환
- 클라이언트의 토큰 요청 및 검증 과정
  - 클라이언트는 받은 인가 코드를 사용해 액세스 토큰을 요청할 때, 초기 요청 시 사용한 코드 검증 값을 함께 전송

PKCE 검증 테스트

```
@Test
void whenAuthorizeRequestIncludesCodeChallenge() {
  when(clientRegistrationRepository.findByRegistrationId("test"))
      .thenReturn(Mono.just(testClientRegistration()));

  webClient
      .mutateWith(SecurityMockServerConfigurers.mockOidcLogin())
      .get()
      .uri("/oauth2/authorization/test")
      .exchange()
      .expectStatus().is3xxRedirection()
      .expectHeader().exists("Location")
      .expectHeader().value("Location", location -> {
        assert location.contains("code_challenge=");
        assert location.contains("code_challenge_method=S256");
      });
}
```

- `/oauth2/authorization/test` 경로로 요청을 보냈을 때, 리디렉션된 Location 헤더에 `code_challenge`와 `code_challenge_method=S256`가 포함되어 있는지 확인

실습

1. 검증에 사용한 주요 명령어

```
docker-compose up -d polar-postgres polar-keycloak polar-ui polar-redis
```

2. catalog-service, edge-service 기동
3. 책에서 시키는 대로 입력
4. 설정을 가져오는지 확인

```
http://localhost:8080/realms/PolarBookshop/.well-known/openid-configuration
```

들어주셔서 감사합니다