Security

Spring Security (세션 기반 관리)

Spring Security란

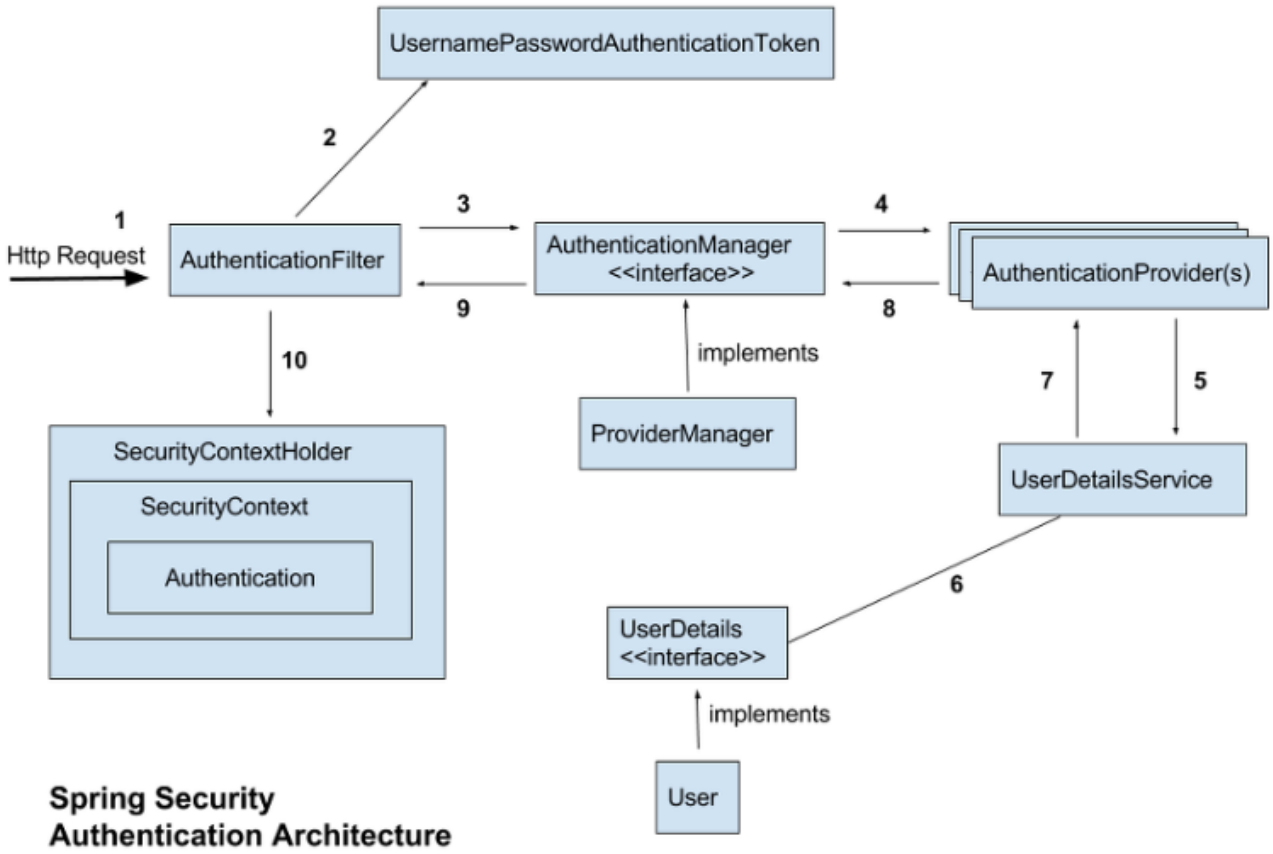
Spring Security는 스프링 기반의 어플리케이션의 보안(인증과 권한)을 담당하는 프레임워크이고 개발자가 자체적으로 세션을 체크하고 redirect할 수고러움을 덜어준다.

Spring Security가 애플리케이션 보안을 구성하는 2가지 영역

- 인증 (Authentication)
  애플리케이션의 작업을 수행할 수 있는 주체(사용자)라고 주장할 수 있는 것을 말함
- 권한 (Authorization)
  인증된 주체가 애플리케이션의 동작을 수행할 수 있도록 허가받았는지를 결정하는 것
    - 권한부여

1. 웹 요청 권한 부여
2. 메소드 호출 및 도메인 인스턴스에 대한 접근 권한 부여

Spring Security 구조



- Client가 request (http request)

- AuthenticationFilter 에서부터 위와같이 user DB까지 타고 들어감
- DB에 있는 유저라면 UserDetails로 꺼내서 유저의 session 생성
- Spring Security의 인메모리 세션저장소인 SecurityContextHolder에 저장
- 유저에게 sessionID와 함께 response를 보냄
- 이후 request에서는 요청쿠이에서 sessionID를 확인해서 검증 후 유효(validate)하면 Authentication 객체를 준다.
  (인증과 관련 권한 부여 완료됨)

메소드 기반 순서 설명

Security 인증 순서는
UserDetailsService 에 **loadUserByUsername(String username)** 로 DB에서 유저정보를 조회 해오면
AuthenticationProvider 에서 **authenticate(Authentication authentication)** 메소드로
UserDetailsService.loadUserByUsername(String username) 메소드로 가지고온 DB유저정보와
authenticate(Authentication authentication) 로 접속한유저 정보(authentication)로 비밀번호를 인증을 처리하게되고
AuthenticationProvider 인증에 성공하게되면 **Authentication객체** 를 돌려주게된다.

Spring Security 설정

의존성 추가

Maven(pom.xml)

```
<!-- Properties -->
<security.version>4.2.2.RELEASE</security.version>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
  <version>${security.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>${security.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>${security.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
```

```
  <version>${security.version}</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <version>${security.version}</version>
</dependency>
```

web.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    classpath:applicationContext.xml
    classpath:applicationContext-security.xml
  </param-value>
</context-param>

<!-- Spring Security-->
<listener>
  <listener-
class>org.springframework.security.web.session.HttpSessionEventPublisher</listener
-class>
</listener>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-name>org.springframework.web.filter.DelegationgFilterProxy</filter-
class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-patter>/*</url-pattern>
</filter-mapping>
```

- HttpSessionEventPublisher : 한 유저가 다른 브라우저로 동시 로그인하는 것을 막음
- DelegatingFilterProxy : 모든 요청은 해당 프록시필터를 거친다. Spring Security는 해당 프록시필터 내의 필터들의 doFilter()를 통해 인증, 인가를 수행
    - `WebSecurityConfigurerAdapter`를 상속받은 클래스에 `@EnableWebSecurity`어노테이션을 명시하고

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{
  ...
}
```

- springSecurityFilterChain을 등록하기 위해서 `AbstractSecurityWebApplicationInitializer`를 상속받은 `WebApplicationInitializer`를 만드는 것으로 대체가능

```
public class SecurityWebApplicationInitializer
          extends AbstractSecurityWebApplicationInitializer{
  ...
}
```

- Spring MVC를 이용해서 애플리케이션을 구성하기 때문에 ApplicationInitializer에 `WebSecurityConfigurerAdapter를 상속` 받은 클래스를 `getRootConfigClasses()` 메소드에 추가함

```
public class ApplicationInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer{
  @Override
  protected Class<?> getRootConfigClasses(){
    return new Class[] { WebSecurityConfig.class };
  }
  ...
}
```

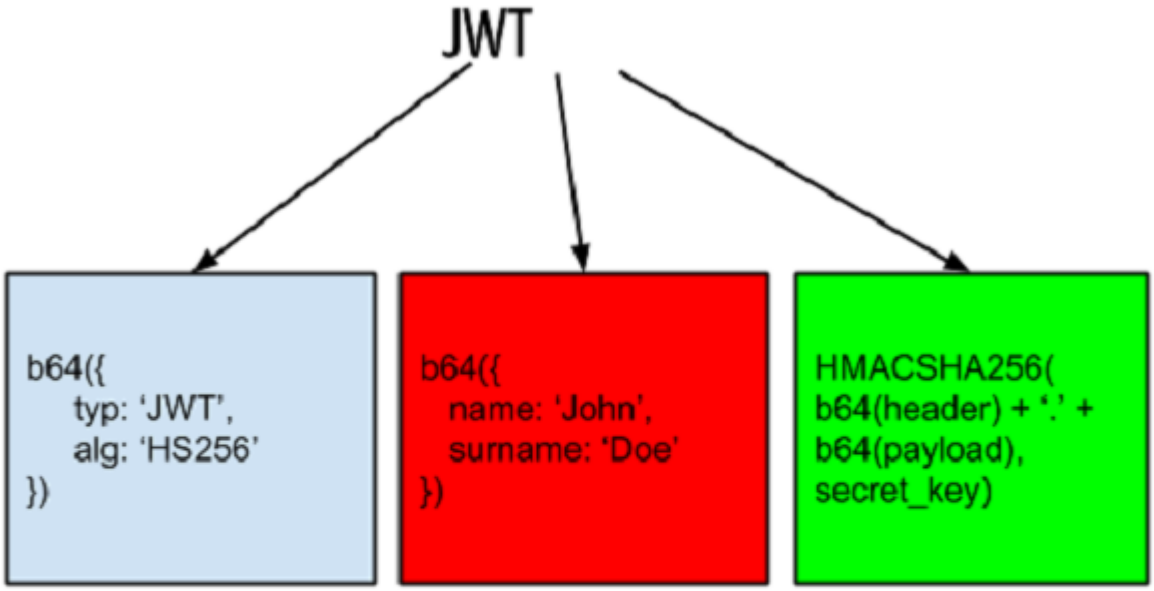
JWT (Json Web Token)

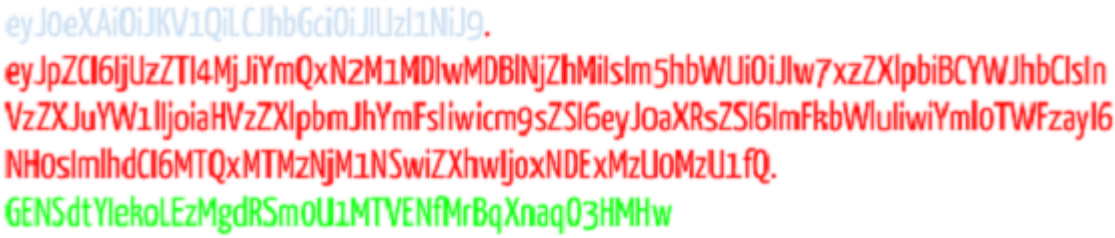
JWT란?

- JSON을 이용한 토큰 기반 인증 방식이고 토큰과 연관된 정보를 서버 쪽에서 찾을 필요없이 토큰 자체가 인증을 위한 정보를 가지고 있다.
- 이 때 사용되는 JSON 데이터는 URL-Safe 하도록 URL에 포함할 수 있는 문자만으로 만든다.
- JWT는 HWAC 알고리즘을 사용하여 비밀키 또는 RSA를 이용한 (Public Key/Private Key)쌍 으로 서명 가능
- 별도의 인증 저장소(세션 저장소)가 필요없다.

JWT의 구조

header.payload.signature 형태로 구성





- header
  토큰 타입과 해시 알고리즘을 담고 있다.
- payload
  사용자 아이디 또는 이메일 등 사용자를 식별하기 위한 정보를 담고 있다.
  역할 또는 권한을 포함할 수 있는 등, 담을 수 있는 내용에는 제한이 없다. 다만, 많은 정보를 담을수록 토큰 크기는 커지므로, 꼭 필요한 정보만 담아야 한다.
- signature
  해시 키를 담고 있다.

JWT를 이용하는 경우

- 회원 인증
  Server가 Client에게 Request받을 때마다 , 해당 토큰이 유효하고 인증됐는지 검증을 하고, 유저가 요청한 작업에 권한이 있는지 확인하여 작업을 처리

- 정보 교류
  JWT는 두 개체 사이에서 안정성있게 정보를 교환하기에 좋은 방법
  - 정보가 서명 되어있기 때문에 정보를 보낸 이가 바뀌진 않았는지, 또 정보가 도중에 조작되지는 않았는지 검증 가능

JWT를 통한 Client-Server간 인증과정

1. Client에서 로그인을 하면
2. Server에서 로그인 확인 후 secret key를 통해 Access token을 발급하여 Client에 JWT를 반환해준다.

3. Server에서 Client가 준 JWT의 서명을 체크하고 JWT에서 사용자 정보를 확인한 뒤
4. 올바를 경우, Client의 Request에 맞는 Response를 반환하고, 맞지 않을 경우 그에 따른 처리를 한다.

OAuth 2.0 이론

OAuth란?

제 3의 앱이 자원의 소유자인 서비스 이용자를 대신하여 서비스를 요청할 수 있도록 자원 접근 권한을 위임하는 방법

어떤 사이트를 이용할 때, 페이스북이나 구글 등으로 회원가입이 가능한 경우가 OAuth를 적용시킨 것

- 페이스북이나 구글 아이디로 제 3의 서비스에 로그인해서 페이스북, 구글에 등록되어 있는 정보나 기능에 접근할 수 있는 권한을 제어하기 위한 표준 프로토콜

OAuth 2.0 의 구성

- Resource Owner(자원 소유자 : DB를 장악하고 있는 OAuth를 사용하는 사람)
- Authorization Server(OAuth 인증 서버)
- Resource Server(REST API Server)
- Client(Resource를 사용하는 직접 사용자)

1.0a와 달라진 점

- 기능의 단순화, 기능과 규모의 확장성 등을 지원하기 위해 만들어짐
- 기존 1.0a는 디지털 서명 기반이었지만 OAuth 2.0 의 암호화는 https에 맡김으로써 복잡한 디지털 서명에 관한 로직을 요구하지 않기 때문에 구현 자체가 개발자입장에서 쉬워짐
- 1.0a는 인증방식이 한가지였지만 2는 다양한 인증방식을 지원한다
- API 서버에서 인증 서버를 분리할 수 있도록 해 놓았다.

OAuth 2.0 의 인증종류

- Authorization Code Grant
    - 서버사이드 코드로 인증하는 방식
    - 권한서버가 클라이언트와 리소스서버간의 중재역할
    - Access Token을 바로 클라이언트로 전달하지 않아 잠재적 유출을 방지
    - 로그인 시에 페이지 URL에 `response_type=code` 라고 넘긴다.
- Implicit Grant
    - token과 scope에 대한 스펙 등은 다르지만 OAuth 1.0a와 가장 비슷한 인증방식
    - Public Client인 브라우저 기반의 어플리케이션이나 모바일 어플리케이션에서 이 방식을 사용하면 된다.
    - OAuth 2.0 에서 가장 많이 사용되는 방식
    - 권한코드 없이 바로 발급되서 보안에 취약
    - 주로 Read only인 서비스에 사용
    - 로그인 시에 페이지 URL에 `response_type=token` 라고 넘긴다.
- Resource Owner Password Credentials Grant

    - Client에 아이디/패스워드를 저장해 놓고 아이디/패스워드로 직접 access token을 받아오는 방식
    - Client를 믿을 수 없을 때에는 사용하기에 위험하기 때문에 API 서비스의 공식 어플리케이션이나 믿을 수 있는 Client에 한해서만 사용하는 것을 추천
    - 로그인 시에 `API에 POST로 grant_type=password` 라고 넘긴다.

- Client Credentials Grant

    - 어플리케이션이 Confidential Client일 때 id와 secret을 가지고 인증하는 방식
    - 로그인 시에 `API에 POST로 grant_type=client_credentials` 라고 넘긴다.

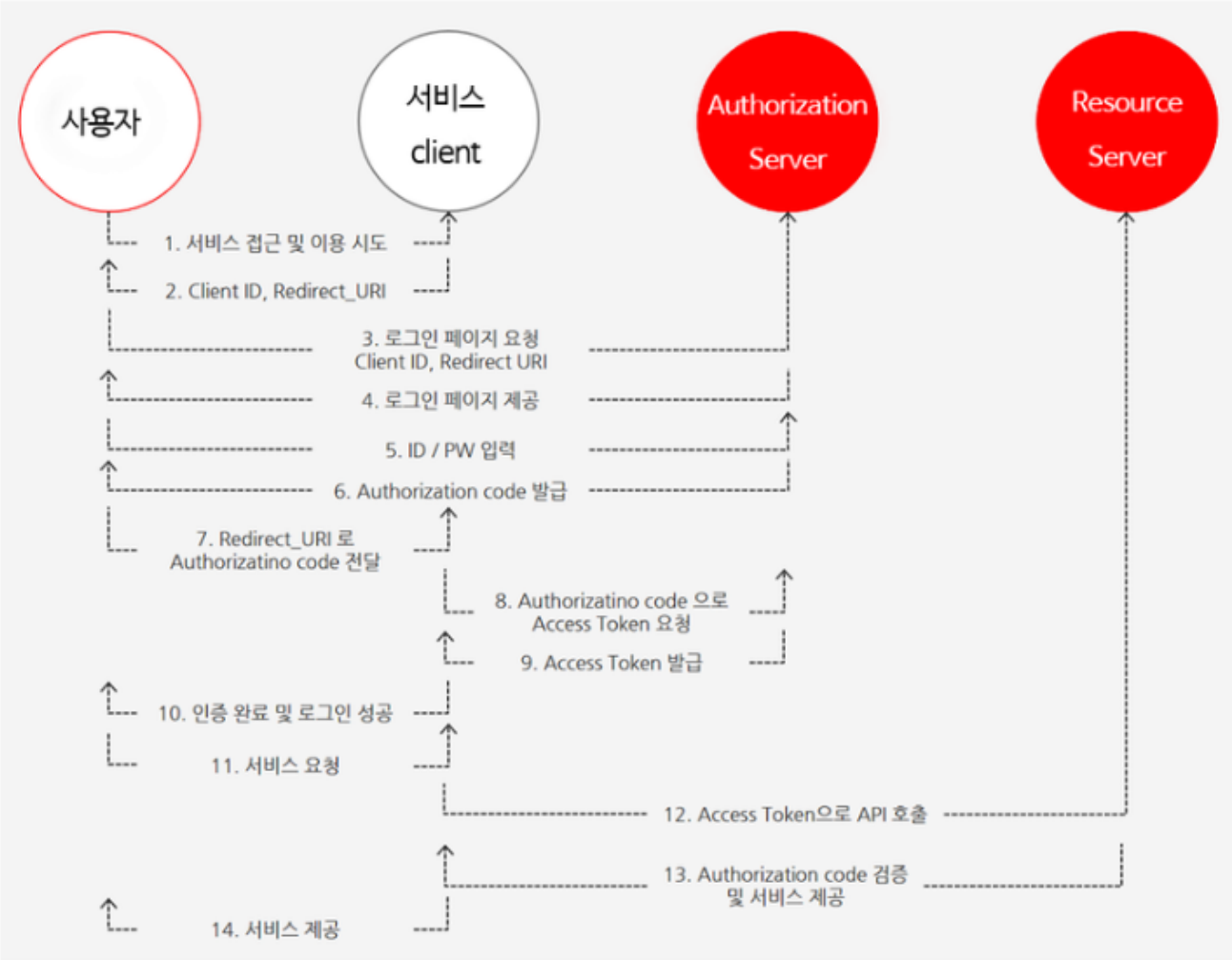
Token

Access Token
위에서 말한 4가지 궈한 요청 방식 모두, 요청 절차를 정상적으로 마치면 클라이언트에게 Access Token이 발급된다. 이 토큰은 보호된 리소스에 접근할 때 권한 확인용으로 사용된다. 문자열 형태이며 클라이언트에 발급된 권한을 대변하게 된다. 계정 아이디와 비밀번호 등 계정 인증에 필요한 형태들을 이 토큰 하나로 표현함으로써, 리소스 서버는 여러 가지 인증 방식에 각각 대응 하지 않아도 권한을 확인할 수 있게 된다.

Refresh Token
한번 발급받은 Access Token은 사용할 수 있는 시간이 제한되어 있다. 사용하고 있던 Access Token이 유효기간 종료 등으로 만료되면, 새로운 액세스 토큰을 얻어야 하는데 그때 이 Refresh Token이 활용된다. 권한 서버가 Access Token을 발급해주는 시점에 Refresh Token도 함께 발급하여 클라이언트에게 알려주기 때문에, 전용 발급 절차 없이 Refresh Token을 미리 가지고 있을 수 있다. 토큰의 형태는 문자열 형태이다. 단, 권한 서버에서만 활용되며 리소스 서버에는 전송되지 않는다.

Token의 갱신 과정
클라이언트가 권한 증서를 가지고 권한서버에 Access Token 을 요청하면, 권한 서버는 Access Token과 Refresh Token 을 함께 클라이언트에 알려줍니다. 그 후 클라이언트는 Access Token을 사용하여 리소스 서버에 각종 필요한 리소스들을 요청하는 과정을 반복합니다. 그러다가 일정한 시간이 흐른 후 액세스 토큰이 만료되면, 리소스 서버는 이후 요청들에 대해 정상 결과 대신 오류를 응답하게 됩니다. 오류 등으로 액세스 토큰이 만료됨을 알아챈 클라이언트는, 전에 받아 두었던 Refresh Token을 권한 서버에 보내어 새로운 액세스 토큰을 요청합니다. 갱신 요청을 받은 권한 서버는 Refresh Token 의 유효성을 검증한 후, 문제가 없다면 새로운 액세스 토큰을 발급해줍니다. 이 과정에서 옵션에 따라 Refresh Token 도 새롭게 발급 될 수 있습니다.

OAuth 2.0 처리 순서



JWT와 OAuth의 차이

JWT와 OAuth는 둘 다 토큰 기반 인증방식을 사용하나, OAuth는 Authorization Server에게 발급받은 토큰을 이용하여 Resource Server에게 사용자 리소스(유저 정보)를 받아 사용하는 반면에 JWT는 토큰 자체에 유저 정보를 담아서 HTTP 헤더로 전달한다.