Spring Security로 MFA(Multi-Factor Authentication) 간단하게 구현하기

MFA란?

- MFA(Multi-Factor Authentication)는 사용자 인증 시 2단계 이상의 정보를 요구하는 방식이에요.
- 예를 들면
  1. 아이디 & 비밀번호 입력
  2. SMS나 이메일로 온 OTP 입력
  3. 지문 입력

이처럼 한 번 더 본인임을 증명해야 하는 거죠.

MFA 구현 목표

이번 시간엔 Spring Security로 다음 과정을 구현해볼 거예요.

1. 비밀번호 인증 → 아이디, 비번으로 인증
2. OTP 인증 → 일회용 코드로 추가 인증
3. JWT 발급 → 인증 완료 시 토큰 반환
4. JWT 필터링 → 이후 요청은 토큰으로 인증 처리

OTP란?

- 일회용 암호(OTP)는 단일 로그인 세션 또는 트랜잭션에 유효한 임시 암호입니다.
- 일반적으로 온라인 계정, 특히 금융 또는 민감한 거래의 경우 추가 보안 계층으로 사용됩니다.
- OTP는 SMS, 이메일 또는 인증 앱과 같은 다양한 수단을 통해 전달될 수 있습니다.

JWT 등장 배경

- HTTP는 stateless 프로토콜입니다.
- 하지만 웹이 복잡해지면서 로그인기능, 장바구니 기능 등의 사용자가 누구인지 기억해야 구현할 수 있게 되요. HTTP는 stateless이기 때문에 구현에 한계가 있죠.

- 그래서 Cookie가 등장합니다.
- 쿠키는 웹 서버가 사용자의 브라우저(클라이언트)에 저장하는 작은 데이터 조각이에요.

JWT 등장 배경

- 하지만 쿠키는 브라우저에 저장되기 때문에 쉽게 노출될 수 있어요.

  Ex) XSS(교차 사이트 스크립팅) 공격
- 그래서 서버에 사용자 정보를 저장하는 세션이 등장합니다.

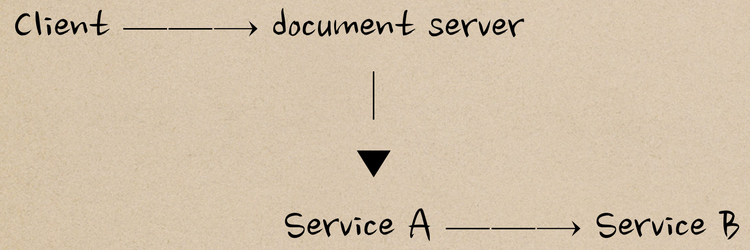
But...

- 웹이 발전하고 서버가 많아짐에 따라 사용자 정보도 여러서버에 공유하기 힘들어졌습니다.
- 마이크로 서비스 환경에서 인증하기도 힘들어졌습니다.
- Rest-Api의 서버가 상태를 기억하지 않는 stateless가 원칙에 위배됩니다.

- 그래서 등장한 것이 JWT입니다.

JWT란?

- JWT(JSON Web Token)는 서버가 인증에 성공한 유저에게 토큰 형태로 인증 정보를 Document서버에 발급하고, 이후 요청은 이 토큰만으로 인증을 유지할 수 있게 해주는 기술이에요.



- Document서버가 가지고 있는 토큰만으로 인증이 완료되기 때문에,
- 서버는 stateless하고,
- 서버를 확장해도 문제가 없게 되었습니다. (MSA 등등)

- JWT는 서버간 통신에 많이 사용됩니다.

JWT구조

- JWT는 각각의 구성요소가 (.)로 구분됩니다.

xxxxxxx.yyyyyyyyy.zzzzzzzzzzzzz

(header).(payload).(signature)

```
{ // Header
  "typ": JWT
  "alg": "HS256"
}
{ // PayLoad
  "username": "chill chill"
}
```

Header: 알고리즘, 타입 정보 (ex: HS256, JWT)

Payload: 유저 정보 (ex: username, roles 등)

Signature: 위의 정보를 기반으로 서버가 서명한 값

- xxxxxxx.yyyyyyyyy: 이 부분은 헤더와 페이로드는 base64를 인코딩한 JSON입니다.
- Signature는 JWT가 위조되지 않았는지 확인하는 수단입니다. 즉, "이 토큰은 진짜 서버가 발급했는 거야!" 라고 증명하는 거죠.

JWT구조

- JWT 시그니처 서명 방식

HS256	비밀키 하나로 서명&검증
RS256, ES256	개인키/ 공개키 쌍으로 서명 & 검증

- HS256방식은 secretKey 하나만 가지고 서명도하고 검증도 합니다.
- 그래서 모든 검증 서버가 같은 키를 가지고 있어야 합니다.

- RS256는 서버가 개인키로 서명한 JWT를 발급합니다.
- 그래서 다른 서버는 공개키로 검증만 할 수 있습니다.
    1. 키를 공유할 필요가 없습니다.
    2. 서버가 여러 개여도, 공개키만 배포하면 됩니다.
    3. 클라이언트에서도 토큰의 유효성을 검증할 수 있습니다. (OAuth/OpenID에서 활용)

MFA 구현

- 시스템 구성: 인증 서버 + 비즈니스 서버
  - 인증 서버 : 사용자의 ID/PW 또는 OTP 인증을 담당
  - 비즈니스 서버: 실제 비즈니스 로직이 실행되는 서버. JWT를 통해 인증된 사용자만 접근 허용

왜 인증서버와 비즈니스 서버를 구분해야할까요?

보안 책임을 분리하기 위해 (Security Isolation) 사용합니다.

인증 서버는 민감한 정보(비밀번호, OTP, 토큰 등)를 처리하는 핵심 위치입니다.

- 인증 관련 로직만 집중 관리
- 보안 취약점을 줄이고 감시 범위를 축소
- 침입 발생 시 인증 정보 유출 범위 최소화

그 이외에도 확장성과 유지보수에 유리하기 때문에 인증 서버를 분리합니다.

구현 코드 링크: https://github.com/dnjstjt1297/spring-mfa-practice

MFA 인증 서버 구현

1. 사용자 생성 및 저장하기

클라이언트 요청

```
curl -X POST -H "Content-Type: application/json" \
    -d '{"username":"chill", "password":"12345"}' \
    http://localhost:8080/user/add -i
```

```
Configuration  dnjstjt1297
ublic class SecurityConfig {

    @Bean  dnjstjt1297
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean  dnjstjt1297
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
                .csrf(AbstractHttpConfigurer::disable)
                .formLogin(AbstractHttpConfigurer::disable)
                .httpBasic(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                        .requestMatchers("/user/add", "/user/auth", "/otp/check").permitAll(
                        .anyRequest().authenticated());
        return http.build();
    }
```

- 요청이 /user/add를 들어오면 인증 필터를 통과하고, 컨트롤러 ⟶ 서비스 ⟶ 리포지토리로 이동
- 필터 부분에서는 Rest를 사용하므로 formlogin과 httpBasic은 사용하지 않았습니다.

```
ublic void addUser(User user) {  1개 사용 위치  dnjstjt1297
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    userRepository.save(user);
}
```

- 유저의 패스워드는 Bcrypt방식으로 인코딩해서 저장했습니다.

MFA 인증 서버 구현

2. OTP 생성 및 저장하기

클라이언트 요청

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"username":"chill", "password":"12345"}' \
     http://localhost:8080/user/auth -i
```

```
public void auth(User user){  1개 사용 위치  dnjstjt1297
    Optional<User> o = userRepository.findUserByUsername(user.getUsername());

    if(o.isPresent()){
        User u = o.get();
        if(passwordEncoder.matches(user.getPassword(), u.getPassword())){ // 비밀번호 일치 확인
            reNewOtp(u); // OTP 새로 발급
        }
    } else {
        throw new BadCredentialsException("Bad credentials"); // 사용자 없거나 비밀번호 틀림
    }
}

private void reNewOtp(User u) {  1개 사용 위치  dnjstjt1297
    String code = CodeUtil.generateCode(); // 랜덤 OTP 생성

    Optional<Otp> userOtp = otpRepository.findOtpByUsername(u.getUsername()); // 기존 OTP 조회

    if(userOtp.isPresent()){
        Otp otp = userOtp.get();
        otp.setCode(code); // 기존 OTP 코드 갱신
    } else {
        Otp otp = new Otp();
        otp.setUsername(u.getUsername());
        otp.setCode(code); // 새 OTP 코드 생성
        otpRepository.save(otp);
    }
```

- 요청이 /user/auth로 들어오면 인증 필터를 통과하고, 컨트롤러 → 서비스 → 리포지토리로 이동
- auth()는 OTP를 생성하는 메서드입니다.

MFA 인증 서버 구현

3. OTP 인증

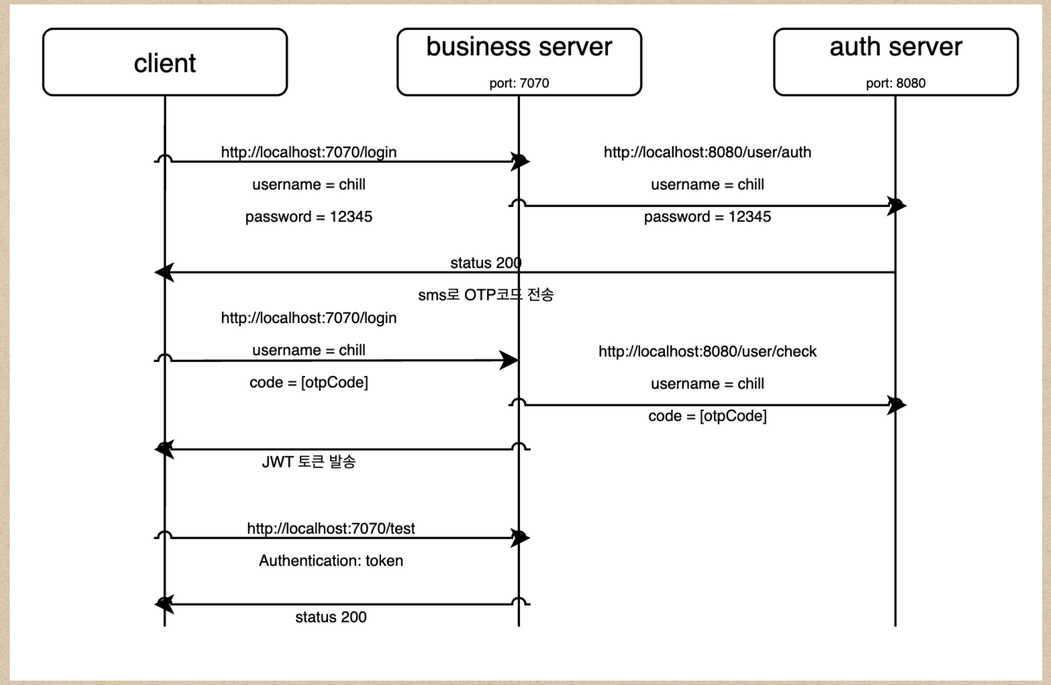
클라이언트 요청

```
curl -X POST -H "Content-Type: application/json" \
     -d '{"username":"chill", "code":"[otpcode]"}' \
     http://localhost:8080/otp/check -i
```

```
public boolean check(Otp otpToValidate){  1개 사용 위치  dnjstjt1297
    Optional<Otp> userOpt = otpRepository.findOtpByUsername(otpToValidate.getUsername());

    if(userOpt.isPresent()){
        Otp otp = userOpt.get();

        if(otpToValidate.getCode().equals(otp.getCode())){ // OTP 일치 여부 검사
            return true;
        }
    }

    return false;
}
```

- 요청이 /otp/check로 들어오면 인증 필터를 통과하고, 컨트롤러 → 서비스 → 리포지토리로 이동
- check()는 클라이언트에게 받은 OTP코드가 유효한지 판단하는 메서드입니다.

로그인 흐름


client
business server
port: 7070
auth server
port: 8080
http://localhost:7070/login
username = chill
password = 12345
http://localhost:8080/user/auth
username = chill
password = 12345
status 200
sms로 OTP코드 전송
http://localhost:7070/login
username = chill
code = [otpCode]
http://localhost:8080/user/check
username = chill
code = [otpCode]
JWT 토큰 발송
http://localhost:7070/test
Authentication: token
status 200

- 비즈니스 서버 구현은 다음 시간에 발표하겠습니다..

질문

- 저의 인증서버는 문제점이 있습니다.
- 1. 악성사용자가 OTP코드를 계속해서 생성해 서버에 보내면 언젠가는 DB에 저장되있는 OTP코드와 일치하겠죠.

  이를 어떻게 막을 수 있을까요?

- OPT인증의 경우 트랜잭션 격리수준을 어느정도 하는게 좋을까요?

QNA