**Exercise 1: Centralized Authentication with OAuth 2.1 / OIDC**

**Task:**  
Implement OAuth2 login (e.g., with Google) in a Spring Boot app.

**Steps:**

1. **Add dependencies**:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-oauth2-client</artifactId>

</dependency>

1. **application.yml**:

spring:

security:

oauth2:

client:

registration:

my-client:

client-id: YOUR\_CLIENT\_ID

client-secret: YOUR\_CLIENT\_SECRET

scope: openid, profile, email

authorization-grant-type: authorization\_code

redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"

provider:

my-provider:

authorization-uri: https://accounts.google.com/o/oauth2/auth

token-uri: https://oauth2.googleapis.com/token

user-info-uri: https://openidconnect.googleapis.com/v1/userinfo

user-name-attribute: sub

1. **SecurityConfig.java**:

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login();

}

}

1. **UserController.java**:

@RestController

public class UserController {

@GetMapping("/user")

public Principal user(Principal principal) {

return principal;

}

}

**Exercise 2: Configuring Authorization and Resource Servers**

**Task:**  
Set up both Authorization Server and Resource Server.

**Steps:**

1. **Add dependencies**:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>

</dependency>

1. **application.yml**:

spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri: https://issuer.example.com

1. **ResourceServerConfig.java**:

@EnableWebSecurity

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2ResourceServer()

.jwt();

}

}

1. **SecureController.java**:

@RestController

public class SecureController {

@GetMapping("/secure")

public String secure() {

return "This is a secure endpoint";

}

}

**Exercise 3: Using JWT for Secure Communication**

**Task:**  
Use JSON Web Tokens (JWT) for authentication and secure communication.

**Steps:**

1. **Add dependencies**:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

1. **application.yml**:

spring:

security:

jwt:

secret: YOUR\_SECRET\_KEY

1. **JwtConfig.java**:

@Configuration

public class JwtConfig {

@Value("${spring.security.jwt.secret}")

private String secret;

public String getSecret() {

return secret;

}

}

1. **JwtTokenProvider.java**:

@Component

public class JwtTokenProvider {

@Autowired

private JwtConfig jwtConfig;

public String createToken(String username) {

Claims claims = Jwts.claims().setSubject(username);

Date now = new Date();

Date validity = new Date(now.getTime() + 3600000); // 1 hour

return Jwts.builder()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(validity)

.signWith(SignatureAlgorithm.HS256, jwtConfig.getSecret())

.compact();

}

public boolean validateToken(String token) {

try {

Jwts.parser().setSigningKey(jwtConfig.getSecret()).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

public Authentication getAuthentication(String token) {

String username = Jwts.parser()

.setSigningKey(jwtConfig.getSecret())

.parseClaimsJws(token)

.getBody()

.getSubject();

return new UsernamePasswordAuthenticationToken(username, "", new ArrayList<>());

}

}

1. **JwtTokenFilter.java**:

public class JwtTokenFilter extends OncePerRequestFilter {

@Autowired

private JwtTokenProvider jwtTokenProvider;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

String token = resolveToken(request);

if (token != null && jwtTokenProvider.validateToken(token)) {

Authentication auth = jwtTokenProvider.getAuthentication(token);

SecurityContextHolder.getContext().setAuthentication(auth);

}

filterChain.doFilter(request, response);

}

private String resolveToken(HttpServletRequest request) {

String bearerToken = request.getHeader("Authorization");

if (bearerToken != null && bearerToken.startsWith("Bearer ")) {

return bearerToken.substring(7);

}

return null;

}

}

1. **SecurityConfig.java**:

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtTokenFilter jwtTokenFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

}