**Module 6:Assignment - Microservices Security**

**Introduction**

Security is a critical aspect of microservices architecture. This document outlines the steps to implement OAuth2 security using Spring Cloud OAuth2 (2.2.5.RELEASE) and Spring OAuth2 Resource Server.

**Objectives**

1. Implement an OAuth2 Authorization Server.
2. Configure the OAuth2 server to issue JWTs.
3. Set up the API Gateway as a Resource Server.
4. Enable JWT validation in the API Gateway.

**Step 1: Implementing the OAuth2 Authorization Server**

**Dependencies (pom.xml)**

Add the following dependencies to your pom.xml:

<dependency>

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

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

</dependency>

**Configure Authorization Server**

Create a configuration class:

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfig {

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

http

.csrf(csrf -> csrf.disable()) // Disable CSRF

.authorizeHttpRequests(auth -> auth

.anyRequest().permitAll() // Allow ALL requests without authentication

);

**return** http.build();

}

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Bean

**public** JwtAuthenticationConverter jwtAuthenticationConverter() {

JwtAuthenticationConverter jwtAuthenticationConverter = **new** JwtAuthenticationConverter();

// Remove any role-based conversion for now

**return** jwtAuthenticationConverter;

}

}

**Step 2: Configuring JWT Issuance**

**Configure Token Enhancer**

@Component

**public** **class** JwtUtil {

**private** **static** **final** String ***SECRET\_KEY*** = "9f8e8d7a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9a0b1c2d3e4f5a6b7c8d9e"; // Ensure this key is the same across services

**public** String extractUsername(String token) {

**return** extractClaim(token, Claims::getSubject);

}

**public** Date extractExpiration(String token) {

**return** extractClaim(token, Claims::getExpiration);

}

**public** <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

**final** Claims claims = Jwts.~~parser~~()

.~~setSigningKey~~(***SECRET\_KEY***)

.parseClaimsJws(token)

.getBody();

**return** claimsResolver.apply(claims);

}

**public** Boolean validateToken(String token, String username) {

**final** String extractedUsername = extractUsername(token);

**return** (extractedUsername.equals(username) && !isTokenExpired(token));

}

**private** Boolean isTokenExpired(String token) {

**return** extractExpiration(token).before(**new** Date());

}

**public** String generateToken(String username) {

**return** Jwts.*builder*()

.setSubject(username)

.setIssuedAt(**new** Date(System.*currentTimeMillis*()))

.setExpiration(**new** Date(System.*currentTimeMillis*() + 1000 \* 60 \* 60)) // 1 hour expiry

.~~signWith~~(SignatureAlgorithm.***HS256***, ***SECRET\_KEY***)

.compact();

}

}

**Step 3: Setting up API Gateway as a Resource Server**

**Dependencies**

<dependency>

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

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

</dependency>

**Configure Security in API Gateway**

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfig {

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

http

.csrf(csrf -> csrf.disable()) // Disable CSRF

.authorizeHttpRequests(auth -> auth

.anyRequest().permitAll() // Allow ALL requests without authentication

);

**return** http.build();

}

**Step 4: Validating JWT in API Gateway**

Configure application.yml:

spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri: http://localhost:8090

**Execution & Expected Output**





