Securing RESTful Web Services using Spring Security and JWT

1. Introduction

This assignment demonstrates the implementation of secure RESTful Web Services using Spring Boot, Spring Security, and JSON Web Tokens (JWT). The goal is to protect APIs with both HTTP Basic Authentication and JWT-based stateless authorization. The assignment walks through securing endpoints, creating authentication mechanisms, and implementing filters to validate JWTs.

2. Technology Stack

- Java 8+
- Spring Boot 2.x
- Spring Security
- jjwt (JWT library)
- Maven
- Postman / curl for testing

3. Implementation Steps

Step 1: Add Spring Security Dependency

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

Step 2: Create SecurityConfig Class

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
      .withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")
      .and()
      .withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");
```

```
    }

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")
            .anyRequest().authenticated()
            .and().httpBasic()
            .and().addFilter(new JwtAuthorizationFilter(authenticationManager()));
    }
}
```

Step 3: Create AuthenticationController

```
@RestController
public class AuthenticationController {

    @GetMapping("/authenticate")
    public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {
        String user = getUser(authHeader);
        String token = generateJwt(user);

        Map<String, String> map = new HashMap<>();
        map.put("token", token);
        return map;
    }
```

```
    private String getUser(String authHeader) {
        String encodedCredentials = authHeader.replace("Basic ", "");
        byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);
        String decoded = new String(decodedBytes);
        return decoded.split(":")[0];
    }

    private String generateJwt(String user) {
        JwtBuilder builder = Jwts.builder();
        builder.setSubject(user);
        builder.setIssuedAt(new Date());
        builder.setExpiration(new Date(System.currentTimeMillis() + 1200000)); // 20 minutes
        builder.signWith(SignatureAlgorithm.HS256, "secretkey");
        return builder.compact();
    }
}
```

Step 4: Create JwtAuthorizationFilter

```
public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

    public JwtAuthorizationFilter(AuthenticationManager authManager) {
        super(authManager);
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain chain)
            throws IOException, ServletException {

        String header = request.getHeader("Authorization");
        if (header == null || !header.startsWith("Bearer ")) {
            chain.doFilter(request, response);
```

```
            return;
        }

        UsernamePasswordAuthenticationToken authentication = getAuthentication(request);
        SecurityContextHolder.getContext().setAuthentication(authentication);
        chain.doFilter(request, response);
    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
        String token = request.getHeader("Authorization");
        if (token != null) {
            try {
                Claims claims = Jwts.parser()
                        .setSigningKey("secretkey")
                        .parseClaimsJws(token.replace("Bearer ", ""))
                        .getBody();

                String user = claims.getSubject();
                if (user != null) {
                    return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());
                }
            } catch (JwtException e) {
                return null;
            }
        }
        return null;
    }
}
```

4. Testing Scenarios

Scenario	Endpoint	Method	Authorization	Expected Result
Get Token	/authenticate	GET	Basic user:pwd	200 OK + JWT Token
Access Protected Endpoint with Token	/countries	GET	Bearer JWT	200 OK
No Token Provided	/countries	GET	None	401 Unauthorized
Wrong Credentials	/authenticate	GET	Basic user:wrongpwd	401 Unauthorized
Access with Wrong Role (admin instead of user)	/countries	GET	Bearer JWT (admin)	403 Forbidden

5. Security Best Practices

- Use HTTPS to secure data in transit.
- Keep secret keys in environment variables, not hardcoded.
- Set short expiry for access tokens.
- Use refresh tokens for long sessions.
- Log access to sensitive endpoints.

6. Conclusion

In this assignment, we implemented RESTful web services secured with Spring Security and JWT. We demonstrated role-based authentication using HTTP Basic Auth and stateless authorization using JWT. We also introduced a custom Spring Filter to intercept and validate JWTs for each secured request.

This approach ensures secure, scalable, and modern web service security suitable for microservices and enterprise APIs.