Woldia University
School of Computing

Department of Software Engineering
Course Title: **Web Service,**
Course Code: **SEng5127**

Chapter Five: Securing Web Services

By : Demeke G.
AY:2018 E.C

Outline

☛Introduction to Secure Web Services

☛HTTP Basic Authentication

☛Secure Message Transmission with SSL/TLS

☛Authenticating and Authorizing Clients

Learning Outcome

✓Configure secure web services using industry-standard security practices.

✓Implement authentication and authorization mechanisms to protect APIs.

✓Secure data transmission using SSL/TLS (HTTPS).

✓Apply HTTP Basic Authentication for simple access control.

✓Control access to services and methods using role-based security.

✓Implement token-based security using OAuth2 and JSON Web Tokens (JWT).

Securing Web Services

- ☛Web services play a critical role in enabling communication between different applications, platforms, and devices.
- ☛These services often handle sensitive information, such as personal data, financial transactions, and confidential organizational records.
- ☛Ensuring the security of web services is essential to protect data from unauthorized access, tampering, and cyber threats.
- ☛Secure web services focus on implementing mechanisms that guarantee confidentiality, integrity, availability, authentication, and authorization.
- ☛ Without proper security, web services are vulnerable to attacks such as data breaches, man-in-the-middle attacks, identity theft, and service misuse

HTTP Basic Authentication

- ☛A simple technique used to protect web services by requiring a username and password before access is granted
- ☛Never be used without HTTPS, because Base64 is only encoding, not encryption.
- ☛ Without SSL/TLS, attackers can easily intercept and decode credentials.
- ☛SSL (Secure Sockets Layer) and TLS (Transport Layer Security) are cryptographic protocols that provide **secure communication** over a network.
- ☛SSL/TLS protects web services by:
  - Encrypting data to prevent eavesdropping
  - Ensuring message integrity
  - Authenticating the server using digital certificates
- ☛When SSL/TLS is enabled, communication happens over **HTTPS instead of HTTP**.
- ☛Organizations usually obtain SSL/TLS certificates from a Certificate Authority (CA)

Generate or Obtain an SSL Certificate

- Use a self-signed certificate for development or obtain one from a Certificate Authority (CA) for production.
- Generate a Self-Signed Certificate using keytool

  -
    ```
    keytool -genkeypair -alias myapp -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore myapp.p12 -validity 3650
    ```
- Place keystore in project (.p12 or .jks)
- Configure **application.properties**

```
server.port=8443
server.ssl.key-store=classpath:myapp.p12
server.ssl.key-store-password=changeit
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=myapp
```

- **Let's Encrypt:** Provides automated, trusted SSL certificates for free, valid 90 days, widely used.
- Others like DigiCert, GlobalSign, GoDaddy, Entrust, RapidSSL, Thawte are paid CA

Authenticating and Authorizing Clients

- ☛ Authentication and authorization are two different but related security concepts.
- ☛ Authentication is the process of verifying the identity of a user or client.
- ☛ Authorization determines what an authenticated user is allowed to do.
  - Examples : Read data , Update records, Delete resources
- ☛ Both processes work together to ensure that only legitimate users access web services and that they perform only permitted actions.

Controlling Access to Services and Methods

- Controlling access means restricting which users can call specific services or methods based on their roles or permissions.

- This is typically implemented using Role-Based Access Control (RBAC),Permission-based authorization ,Security annotations

- Example in a REST API:/admin/* endpoints are accessible only to users with the ADMIN role. /user/* endpoints are accessible to normal users.

- Enable Method-Level Security

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
@Configuration
@EnableMethodSecurity
public class MethodSecurityConfig {  }
```

Example 1: Securing Controller Methods (Role-Based Access)

```
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api")
public class UserController {
    @GetMapping("/public")
    public String publicApi() {   return "This endpoint is public";    }
    @PreAuthorize("hasRole('USER')")
    @GetMapping("/user")
    public String userApi() {   return "This endpoint is for USER role";    }
    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/admin")
    public String adminApi() {   return "This endpoint is for ADMIN role";    }
}
```

- Any user can access /public
- Only users with ROLE_USER can access /user
- Only users with ROLE_ADMIN can access /admin

Example 2: Securing Service Layer Methods

- Can also protect business logic directly in service classes
- This ensures security even if someone tries to bypass the controller.

```
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.stereotype.Service;
@Service
public class AccountService {
    @PreAuthorize("hasRole('ADMIN')")
    public void deleteAccount(Long id) { // only ADMIN can delete accounts
}
    @PreAuthorize("hasAnyRole('USER','ADMIN')")
    public String viewAccount(Long id) {        return "Account details";    }
}
```

Example 3: Securing URLs in Security Configuration

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -> csrf.disable())
        .authorizeHttpRequests(auth -> auth
            .requestMatchers("/api/public/**").permitAll()
            .requestMatchers("/api/admin/**").hasRole("ADMIN")
            .requestMatchers("/api/user/**").hasAnyRole("USER", "ADMIN")
            .anyRequest().authenticated()
        )
        .httpBasic();
    return http.build();
}
```

Providing Authentication Information (OAuth 2.0, JWT)

- These technologies provide modern, robust ways to manage identity and access tokens.
- OAuth 2.0 is an industry-standard protocol for authorization.
- OAuth 2.0 focuses on delegated access, allowing a third-party application to access limited resources on a user's behalf without sharing the user’s credentials
- JSON Web Tokens (JWT) is a compact, URL-safe means of representing claims between two parties.
- JWTs are often used within an OAuth 2.0 flow to encode authentication and authorization information
- A JWT is a string composed of three parts separated by dots: a header, a payload (containing claims like user ID, expiration time, roles), and a signature used to verify that the token has not been tampered with.
- The signature ensures integrity and authenticity, but the payload itself is typically only encoded (Base64) and not encrypted

Summary

- Web service security is essential to protect data and systems from unauthorized access and cyber threats.
- Secure communication is achieved using SSL/TLS (HTTPS) to encrypt data in transit.
- Authentication verifies the identity of users or systems, while authorization controls access to resources.
- Common security methods include HTTP Basic Authentication, role-based access control, and token-based security.
- Modern applications use OAuth2 and JWT to provide secure, scalable, and stateless authentication.
- Proper configuration of security in frameworks like Spring Boot helps build reliable and secure APIs.

End of Chapter