Lab Exercise 11- API Security Threats (OWASP API Top 10) Using Spring Boot

---

1. Objective

This lab teaches students how to:

1. Build a **vulnerable REST API** in Spring Boot.
2. Perform **attacks** (BOLA, Broken Authentication, Mass Assignment, Excessive Data Exposure, etc.).
3. Build a **secure version** of the same API using Spring Security.
4. Test everything using **Postman**.
5. Document everything in **Swagger**.

2. Project Structure

```
src/
 └── main/
     ├── java/
     │   └── com/example/securityapi/
     │       ├── model/User.java
     │       ├── service/UserService.java
     │       ├── controller/VulnerableUserController.java
     │       ├── controller/SecureUserController.java
     │       ├── config/SecurityConfig.java
     │       ├── config/OpenAPIConfig.java
     │       └── SecurityapiApplication.java
     └── resources/
         └── application.properties
pom.xml
```

3. Step-by-Step Lab Instructions

ID	Threat	Description
API1	Broken Object Level Authorization (BOLA)	Attackers access others' data using IDs
API2	Broken Authentication	Weak login/authentication mechanisms
API3	Broken Object Property Level Authorization	Over-posting/excessive data exposure
API4	Unrestricted Resource Consumption	No rate-limits → DoS risks
API5	Broken Function Level Authorization	Unauthorized admin actions
API6	Unrestricted Access to Sensitive Business Flows	Critical workflows lack security
API7	Server-Side Request Forgery	API fetches attacker-controlled URLs
API8	Security Misconfiguration	Missing headers, debug enabled
API9	Improper Inventory Management	Unknown/unmaintained API versions
API10	Unsafe Consumption of APIs	Blindly trusting external APIs

---

STEP 1 — Create a New Spring Boot Project

Use Spring Initializr or Maven to create:

- Spring Web
- Spring Security
- Springdoc OpenAPI

STEP 2 — Add All Required Dependencies (pom.xml)

Paste this in pom.xml file

```
<dependency>

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

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

</dependency>
```

STEP 3 — Create the Model (User.java)

```
package com.example.securityapi.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

public class User {

   private Long id;
   private String name;
   private String email;
   private String role;

   @JsonIgnore
   private String password;

   public User() {
   }

   public User(Long id, String name, String email, String role, String password) {
      this.id = id;
      this.name = name;
      this.email = email;
      this.role = role;
      this.password = password;
   }

   // ---------- GETTERS ----------

   public Long getId() {
      return id;
   }
```

```
public String getName() {
    return name;
}

public String getEmail() {
    return email;
}

public String getRole() {
    return role;
}

@JsonIgnore
public String getPassword() {
    return password;
}

// ---------- SETTERS ----------

public void setId(Long id) {
    this.id = id;
}

public void setName(String name) {
    this.name = name;
}

public void setEmail(String email) {
    this.email = email;
}
```

```
    public void setRole(String role) {
        this.role = role;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

STEP 4 — Create the Service (UserService.java)

```
package com.example.securityapi.service;

import com.example.securityapi.model.User;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.atomic.AtomicLong;

@Service
public class UserService {

    private final Map<Long, User> store = new HashMap<>();
    private final AtomicLong counter = new AtomicLong(3);

    public UserService() {
        store.put(1L, new User(1L, "Admin", "admin@test.com", "admin", "admin123"));
        store.put(2L, new User(2L, "User", "user@test.com", "user", "user123"));
    }
    public User find(Long id) { return store.get(id); }
    public Collection<User> findAll() { return store.values(); }
    public User save(User u) {
        if (u.getId() == null) u.setId(counter.getAndIncrement());
        store.put(u.getId(), u);
        return u;
    }

    public boolean delete(Long id) {
        return store.remove(id) != null;
    }
}
```

STEP 5 — Create the VULNERABLE API (OWASP Top 10)

File: VulnerableUserController.java

```
package com.example.securityapi.controller;

import com.example.securityapi.model.User;
import com.example.securityapi.service.UserService;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

@RestController
@RequestMapping("/api/v1") // vulnerable
public class VulnerableUserController {

    private final UserService svc;

    public VulnerableUserController(UserService svc) { this.svc = svc; }

    // API1 – BOLA
```

```
@GetMapping("/users/{id}")
public User getUser(@PathVariable Long id) {
   return svc.find(id);
}

// API3 – Mass Assignment
@PostMapping("/users")
public User createUser(@RequestBody User user) {
   return svc.save(user);
}

// API2 – Broken Authentication
@PostMapping("/login")
public String login(@RequestBody User user) {
   User u = svc.find(user.getId());
   if (u != null && u.getPassword().equals(user.getPassword()))
      return "Login Successful!";
   return "Invalid Login!";
}

// API5 – Broken Function-Level Authorization
@DeleteMapping("/admin/delete/{id}")
public String deleteUser(@PathVariable Long id) {
   svc.delete(id);
   return "User deleted!";
}

// API8 – Excessive Data Exposure
@GetMapping("/debug/info")
public Collection<User> debug() {
   return svc.findAll();
}
```

```
}
```

STEP 6 — Create the SECURE API

File: SecureUserController.java

```
package com.example.securityapi.controller;

import com.example.securityapi.model.User;
import com.example.securityapi.service.UserService;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/secure")
public class SecureUserController {

    private final UserService svc;

    public SecureUserController(UserService svc) { this.svc = svc; }

    @GetMapping("/users/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<User> getUser(@PathVariable Long id) {
        User u = svc.find(id);
        return u == null ? ResponseEntity.notFound().build() : ResponseEntity.ok(u);
    }

    @PostMapping("/users")
    public ResponseEntity<User> createUser(@RequestBody User user) {
        user.setRole("user");
        user.setId(null);
        return ResponseEntity.ok(svc.save(user));
    }

    @DeleteMapping("/admin/delete/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public String delete(@PathVariable Long id) {
        svc.delete(id);
        return "User deleted";
    }
}
```

STEP 7 — Add Security Configuration

File: SecurityConfig.java

```
package com.example.securityapi.config;

import org.springframework.context.annotation.*;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSe
curity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

  @Bean
  public UserDetailsService users() {

    UserDetails admin = User.withUsername("admin")
        .password("{noop}admin123")
        .roles("ADMIN")
        .build();

    UserDetails user = User.withUsername("user")
        .password("{noop}user123")
        .roles("USER")
        .build();

    return new InMemoryUserDetailsManager(admin, user);
  }

  @Bean
  public SecurityFilterChain chain(HttpSecurity http) throws Exception {

    http.csrf(csrf -> csrf.disable())
      .authorizeHttpRequests(auth -> auth
        .requestMatchers("/api/v1/**").permitAll() // vulnerable
        .requestMatchers("/api/secure/**").authenticated()
        .requestMatchers("/swagger-ui/**", "/v3/api-docs/**").permitAll()
      )
      .httpBasic();

    return http.build();
  }
}
```

STEP 8 — Add Swagger Configuration

File: OpenAPIConfig.java

```
package com.example.securityapi.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Bean;
import io.swagger.v3.oas.models.*;
import io.swagger.v3.oas.models.info.*;

@Configuration
public class OpenAPIConfig {

    @Bean
    public OpenAPI custom() {
        return new OpenAPI()
                .info(new Info()
                        .title("Security API Lab")
                        .version("1.0")
                        .description("OWASP API Top 10 Demonstration"));
    }
}
```

STEP 9 — application.properties

```
server.port=8080
```

STEP 10 — Run the Application

```
mvn spring-boot:run
```

STEP 11 — SWAGGER URL

```
http://localhost:8080/swagger-ui/index.html
```

STEP 12 — POSTMAN API LIST (Complete)

A. Vulnerable APIs (NO AUTH REQUIRED)

1. Get User (BOLA)

GET

```
http://localhost:8080/api/v1/users/1
```

2. Create User (Mass Assignment)

POST

```
http://localhost:8080/api/v1/users
```

Body:

```
{
  "id": 99,
  "name": "Hacker",
  "role": "admin",
  "email": "hack@test.com",
  "password": "root"
}
```

3. Delete User Without Auth (Function-Level Auth)

DELETE

```
http://localhost:8080/api/v1/admin/delete/1
```

4. Debug – Full Data Leak

GET

```
http://localhost:8080/api/v1/debug/info
```

B. SECURE APIs (Basic Auth Required)

Use:

```
Username: admin

Password: admin123
```

---

1. Get User (ADMIN only)

GET

```
http://localhost:8080/api/secure/users/1
```

---

2. Create User (Role always forced to "user")

POST

```
http://localhost:8080/api/secure/users
```

Body:

```
{
  "name": "New User",
  "email": "new@test.com",
  "password": "pass123"
}
```

---

3. Delete User (ADMIN only)

DELETE

```
http://localhost:8080/api/secure/admin/delete/2
```