Vulnerability Report

COMP47910 Secure Software Engineering

Luis Marron (23202882)

A Lab Journal in part fulfilment of the degree of

MSc. in Advanced Software Engineering

Supervisor: Dr. Liliana Pasquale

UCD School of Computer Science
University College Dublin

July 26, 2025

Table of Contents

Chapter 1: **A05:2021 Security Misconfiguration**

1.1 CWE-693: Protection Mechanism Failure

Description: The BookShop application does not implement Content Security Policy (CSP) headers, leaving it vulnerable to Cross-Site Scripting (XSS) and data injection attacks. CSP is a critical security layer that helps detect and mitigate various types of attacks including data theft, site defacement, and malware distribution by declaring approved sources of content that browsers should be allowed to load.

CWE Explanation: CWE-693 occurs when a protection mechanism (in this case, CSP headers) is not implemented or is improperly configured, allowing attackers to bypass security controls and execute malicious content.

Severity: High

Risk Level:

- **Probability**: High - CSP headers are completely absent, making XSS attacks trivial to execute
- **Impact**: High - Can lead to data theft, session hijacking, and malware distribution
- **Overall Risk**: High - Critical security control missing with high exploitation potential

Location:

- Frontend: `http://localhost:3000`
- Backend: `http://localhost:8080`

Source Code Location:

- Frontend: `frontend/bookshop-frontend/public/index.html` (lines 6-12) - Missing CSP meta tag in HTML head section
- Backend: No security configuration class exists in `src/main/java/com/example/bookshop/config/` directory
- Application Properties: `src/main/resources/application.properties` - No CSP-related configuration

Exploitation:

- An attacker can inject malicious JavaScript code through user input fields (e.g., book titles, descriptions) that will execute in the context of the application

- The absence of CSP headers allows any script source to be executed, including inline scripts and external domains
- Example Attack Vector (Intercepted in ZAP):

  ```
  POST /admin/book HTTP/1.1
  Host: localhost:8080
  Content-Type: application/json
  Cookie: JSESSIONID=xyz789

  {
      "title": "<script>alert('XSS')</script>",
      "author": "Author",
      "price": 29.99,
      "copies": 10
  }
  ```

- The malicious script would execute when an admin views the book list, potentially stealing session cookies or performing unauthorized actions
- ZAP's Active Scan and manual testing confirmed the absence of CSP headers in HTTP responses

Impact:

- Enables Cross-Site Scripting (XSS) attacks, allowing execution of arbitrary JavaScript code
- Facilitates data theft through session hijacking and cookie stealing
- Enables site defacement and distribution of malware to users
- Compromises user privacy and application integrity
- May lead to regulatory violations (GDPR, CCPA) due to data exposure

WebGoat Practice:

- Replicate in WebGoat's **Cross-Site Scripting (XSS)** lessons
- Practice XSS attacks in **Reflected XSS** and **Stored XSS** modules
- Use ZAP to intercept requests and inject malicious scripts
- Verify the absence of CSP headers using browser developer tools or ZAP's Response tab
- Practice bypassing CSP restrictions (when implemented) in WebGoat's **Client Side ¿ Cross-Site Scripting** lessons

Remediation:

1. **Backend Fix (Spring Boot)**:
    - Create a security configuration class:

```
// src/main/java/com/example/bookshop/config/SecurityConfig.java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exc
        http.headers(headers -> headers
            .contentSecurityPolicy(csp -> csp
                .policyDirectives("default-src ’self’; " +
                    "script-src ’self’ ’unsafe-inline’; " +
                    "style-src ’self’ ’unsafe-inline’; " +
                    "img-src ’self’ data:; " +
                    "connect-src ’self’ http://localhost:3000; " +
                    "frame-ancestors ’none’;")
            )
        );
        return http.build();
    }
}
```

- Add Spring Security dependency to `pom.xml`:

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

2. **Frontend Fix (React)**:

- Add CSP meta tag to `frontend/bookshop-frontend/public/index.html`:

```
<meta http-equiv="Content-Security-Policy"
      content="default-src ’self’;
               script-src ’self’ ’unsafe-inline’;
               style-src ’self’ ’unsafe-inline’;
               connect-src ’self’ http://localhost:8080;" />
```

3. **Testing the Fix**:

- Use ZAP to verify CSP headers are present in HTTP responses
- Test XSS payloads are blocked by CSP policies
- Use browser developer tools to check CSP header implementation

1.2 CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Description: The BookShop application does not implement anti-clickjacking headers, leaving it vulnerable to clickjacking attacks. The response does not protect against 'ClickJacking' attacks and should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options header to prevent malicious sites from embedding the application in iframes.

CWE Explanation: CWE-1021 occurs when the application fails to properly restrict how its UI layers or frames can be rendered, allowing attackers to overlay malicious content over legitimate application interfaces to trick users into performing unintended actions.

Severity: Medium

Risk Level:

- **Probability**: Medium - Clickjacking attacks require user interaction but are relatively easy to execute
- **Impact**: Medium - Can lead to unauthorized actions, data theft, and user deception
- **Overall Risk**: Medium - Missing security header with moderate exploitation complexity

Location:

- Frontend: `http://localhost:3000`
- Backend: `http://localhost:8080`

Source Code Location:

- Frontend: `frontend/bookshop-frontend/public/index.html` (lines 6-12) - Missing X-Frame-Options meta tag
- Backend: No security configuration class exists in `src/main/java/com/example/bookshop/config/` directory
- Application Properties: `src/main/resources/application.properties` - No frame-ancestors configuration

Exploitation:

- An attacker can create a malicious website that embeds the BookShop application in a transparent iframe
- The malicious site overlays invisible buttons or forms over the legitimate application interface
- Example Attack Vector (Intercepted in ZAP):

```
<!DOCTYPE html>
<html>
<head><title>Malicious Site</title></head>
<body>
    <div style="position: relative; opacity: 0.1;">
        <iframe src="http://localhost:3000"
                style="width: 100%; height: 100vh; border: none;">
        </iframe>
    </div>
    <button style="position: absolute; top: 50%; left: 50%;
```

```
                    z-index: 1000; opacity: 0.9;">
                Click here to win!
            </button>
        </body>
        </html>
```

- Users believe they're clicking on legitimate buttons but actually perform actions on the embedded BookShop application
- ZAP's Active Scan detected the missing anti-clickjacking headers in HTTP responses

Impact:

- Enables clickjacking attacks where users perform unintended actions
- Can lead to unauthorized purchases, account modifications, or data exposure
- Compromises user trust and application integrity
- May result in financial losses or privacy violations

WebGoat Practice:

- Replicate in WebGoat's **Client Side ¿ Cross-Site Scripting** lessons
- Practice iframe-based attacks in **Cross-Site Request Forgery (CSRF)** modules
- Use ZAP to test frame-ancestors and X-Frame-Options headers
- Practice creating malicious iframe overlays in WebGoat's clickjacking exercises
- Test browser security policies using developer tools

Remediation:

1. **Option 1: Content Security Policy (CSP) frame-ancestors**:
   - Add frame-ancestors directive to CSP header in `SecurityConfig.java`:

     ```
     .contentSecurityPolicy(csp -> csp
         .policyDirectives("default-src 'self'; " +
             "script-src 'self' 'unsafe-inline'; " +
             "style-src 'self' 'unsafe-inline'; " +
             "img-src 'self' data:; " +
             "connect-src 'self' http://localhost:3000; " +
             "frame-ancestors 'none';")
     )
     ```

   - For same-origin framing only, use: `frame-ancestors 'self';`
2. **Option 2: X-Frame-Options Header**:
   - Add X-Frame-Options to security configuration:

```
http.headers(headers -> headers
    .frameOptions().deny()  // or .sameOrigin()
)
```

- Use `.deny()` to prevent all framing or `.sameOrigin()` for same-origin only

3. **Frontend Fix (React)**:

   - Add X-Frame-Options meta tag to `frontend/bookshop-frontend/public/index.html`:

     ```
     <meta http-equiv="X-Frame-Options" content="DENY" />
     ```

   - Or for same-origin only: `content="SAMEORIGIN"`

4. **Testing the Fix**:

   - Use ZAP to verify X-Frame-Options or frame-ancestors headers are present
   - Test iframe embedding is properly blocked in browser developer tools
   - Verify malicious iframe overlays are prevented
   - Check that legitimate same-origin framing still works (if using SAMEORIGIN)

Chapter 2: **A07:2021 Identification and Authentication Failures**

2.1 CWE-264: Permissions, Privileges, and Access Controls

Description: The BookShop application has a Cross-Origin Resource Sharing (CORS) misconfiguration that permits cross-domain read requests from arbitrary third-party domains to unauthenticated APIs. While web browser implementations do not permit arbitrary third parties to read responses from authenticated APIs, this misconfiguration could still be exploited by attackers to access sensitive data through unauthenticated endpoints.

CWE Explanation: CWE-264 occurs when the application fails to properly restrict cross-origin requests, allowing unauthorized domains to access resources and potentially sensitive data.

Severity: Medium

Risk Level:

- **Probability**: Medium - CORS is configured but allows specific origins, reducing but not eliminating risk
- **Impact**: Medium - Can lead to unauthorized data access but limited by browser security policies
- **Overall Risk**: Medium - Misconfiguration exists but with some mitigating factors

Location:

- Frontend: `http://localhost:3000`
- Backend: `http://localhost:8080`

Source Code Location:

- `src/main/java/com/example/bookshop/controller/BookControl.java` (line 10) - `@CrossOrigin(o = "http://localhost:3000", allowCredentials = "true")`
- `src/main/java/com/example/bookshop/controller/AdminControl.java` (line 13) - `@CrossOrigin(origins = "http://localhost:3000", allowCredentials = "true")`
- `src/main/java/com/example/bookshop/controller/AuthControl.java` (line 10) - `@CrossOrigin(o = "http://localhost:3000", allowCredentials = "true")`
- `src/main/java/com/example/bookshop/controller/CartControl.java` (line 13) - `@CrossOrigin(o = "http://localhost:3000", allowCredentials = "true")`

Exploitation:

- An attacker can create a malicious website that makes cross-origin requests to the BookShop API endpoints
- The `@CrossOrigin` annotation allows requests from `http://localhost:3000` with credentials
- Example Attack Vector (Intercepted in ZAP):

  ```
  GET /books HTTP/1.1
  Host: localhost:8080
  Origin: http://malicious-site.com
  Access-Control-Allow-Origin: http://localhost:3000
  Access-Control-Allow-Credentials: true
  ```

- The unauthenticated `/books` endpoint returns all book data, which could be accessed by malicious sites
- ZAP's Active Scan detected the CORS misconfiguration and flagged it as a potential security issue

Impact:

- Unauthorized access to sensitive data through unauthenticated API endpoints
- Potential data leakage of book information, prices, and inventory details
- Enables cross-site request forgery (CSRF) attacks when combined with other vulnerabilities
- Violates the Same Origin Policy (SOP) security model

WebGoat Practice:

- Replicate in WebGoat's **Cross-Site Request Forgery (CSRF)** lessons
- Practice CORS bypass techniques in **Client Side ¿ Cross-Site Scripting** modules
- Use ZAP to intercept and modify CORS headers in requests
- Test cross-origin requests using browser developer tools or ZAP's Active Scan
- Practice implementing proper CORS policies in WebGoat's security configuration lessons

Remediation:

1. **Option 1: Remove CORS Entirely (Recommended for Internal Apps)**:
   - Remove all `@CrossOrigin` annotations from controller classes:
     - `src/main/java/com/example/bookshop/controller/BookControl.java` (line 10)
     - `src/main/java/com/example/bookshop/controller/AdminControl.java` (line 13)

    - src/main/java/com/example/bookshop/controller/AuthControl.java (line 10)
    - src/main/java/com/example/bookshop/controller/CartControl.java (line 13)
  - This allows browsers to enforce Same Origin Policy (SOP) more restrictively

2. **Option 2: Global CORS Configuration (For Multi-Domain Apps)**:
   - Create a centralized CORS configuration:

     ```
     // src/main/java/com/example/bookshop/config/CorsConfig.java
     @Configuration
     public class CorsConfig {
         @Bean
         public CorsConfigurationSource corsConfigurationSource() {
             CorsConfiguration configuration = new CorsConfiguration();
             configuration.setAllowedOrigins(Arrays.asList("http://localhost:
             configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PU
             configuration.setAllowedHeaders(Arrays.asList("*"));
             configuration.setAllowCredentials(true);

             UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigu
             source.registerCorsConfiguration("/**", configuration);
             return source;
         }
     }
     ```

   - Remove individual @CrossOrigin annotations from controllers

3. **Option 3: Add Authentication to Sensitive Endpoints**:
   - Implement authentication for the /books endpoint:

     ```
     @GetMapping
     @PreAuthorize("hasRole('USER')")
     public List<Book> listBooks() {
         return bookRepo.findAll();
     }
     ```

   - This ensures sensitive data is not available in unauthenticated manner

4. **Testing the Fix**:
   - Test cross-origin requests are properly blocked or restricted
   - Use browser developer tools to check CORS headers
   - Verify ZAP no longer flags CORS misconfiguration

Chapter 3: **Conclusions**

The BookShop application exhibits a critical **OWASP Top 10 A5: Security Misconfiguration** vulnerability (CWE-693), specifically the absence of Content Security Policy headers. This security misconfiguration enables Cross-Site Scripting attacks, data theft, and potential malware distribution. The vulnerability affects both the frontend React application and the backend Spring Boot API, requiring immediate remediation through proper CSP header implementation. Practice these vulnerabilities in WebGoat to build detection and exploitation skills before testing production applications.

Chapter 4: **Prepared By**

- **Tester**: Luis Marron
- **Date**: January 2025
- **Contact**: luis.marron@ucdconnect.ie

Bibliography

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley*. 395 pages. ISBN-13: 978-0-201-63361-0. Pearson Education, 1994.