Project Report

CS6417: Software Security

Winter 2024

Simple Book Store : A Secure shopping application

Karthik Beedubail

3734007

Abstract

In this project, a secure web application for a bookstore was developed to explore and apply software security practices. Utilizing the Spiral model[1] of the Software Development Life Cycle (SDLC), the application was designed with a focus on identifying and mitigating potential security threats. Key features implemented include customer account management, book browsing, order placement, and administrative functions. Through a detailed analysis of the application's attack surface and potential attack vectors, such as user inputs, authentication mechanisms, and cross-site scripting, technical controls were implemented to enhance security. The project leveraged Spring Boot[7] for the backend, Next.js React[8] for the frontend, and applied rigorous testing with tools like ZAP[5] and concepts from Web Goat. This report outlines the development process, attack surface analysis, technical controls, and testing methodologies used, providing insights into the effective application of software security measures in a practical project.

Introduction

Selection of the SDLC Model

For this project, the main goal was to develop a web application that is not only functional but also secure, capable of mitigating the risk of any cyber attack. To achieve this, it was imperative to choose a development model that facilitates the integration of software security practices at every stage of development.

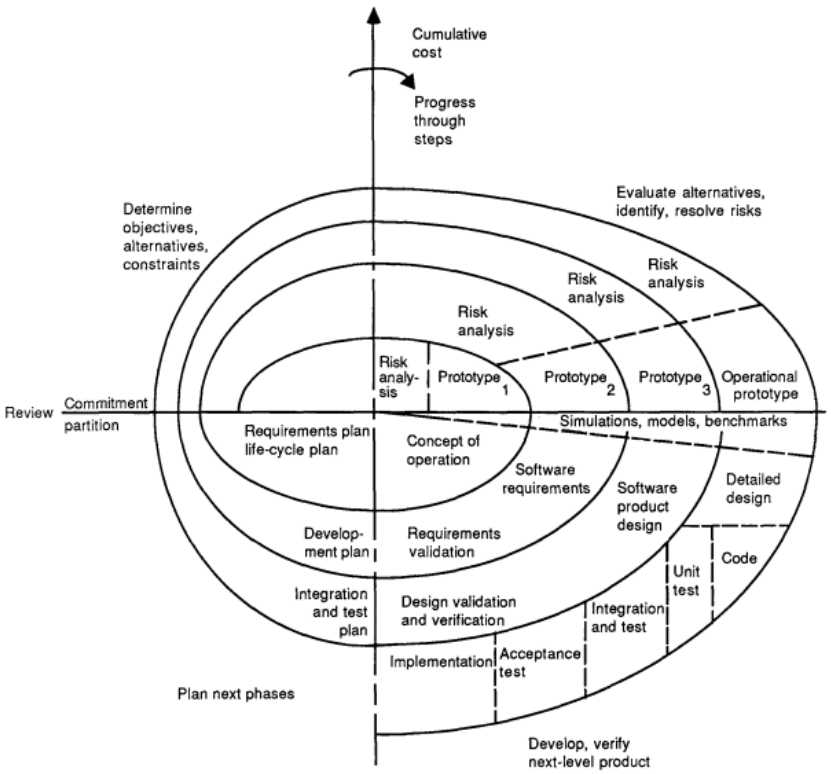

Figure 1: Spiral model of the software process [1]

After careful consideration, the Spiral model was selected for its distinctive approach to risk management and iterative development. The Spiral model [1] is known for its focus on early

identification and mitigation of risks, making it an ideal choice for projects where security is important. This model divides the project lifecycle into four major phases: planning, risk analysis, engineering (or development), and evaluation (or review). This cyclical approach allows for continuous refinement and evolution of the project, with an emphasis on risk management at each turn of the spiral. Figure 1 illustrates the iterative nature and emphasis on risk analysis considered in Spiral Model.

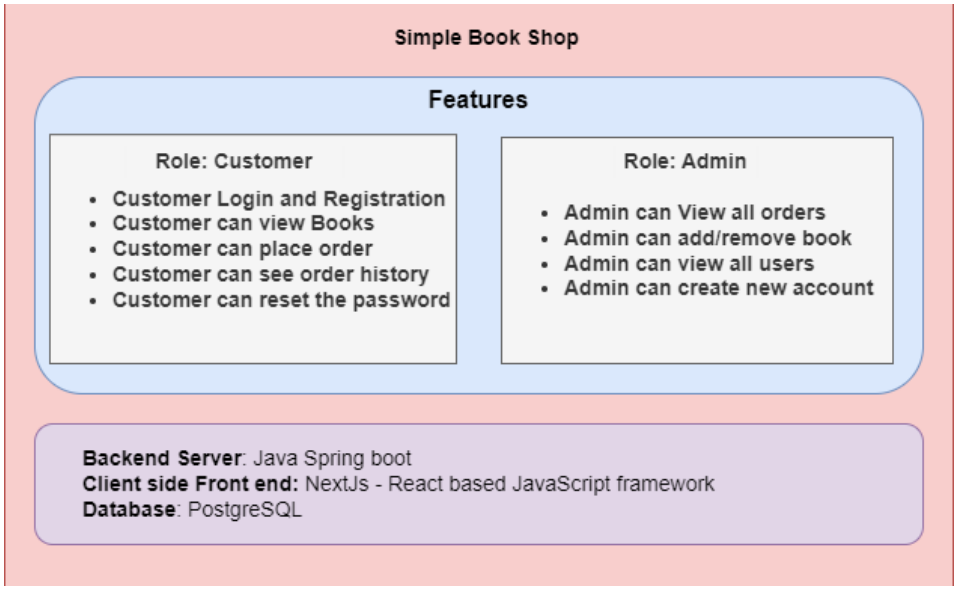

Figure 2. Key Features of our “Simple Book shop” web application.

Following the Spiral model's guidelines, we embarked on the development of the web application's key features, including customer login and registration, book browsing, order placement, and administrative functions. Figure 2 lists out the major functionalities of our “Simple Book shop” web application. The pre-emptive approach of Spiral model is crucial in avoiding potential failures in the later stages of the project, ensuring that security measures are not just an afterthought but are embedded in the core of the application's development.

Attack Tree and Attack Surface

Overview of Attack Surfaces

In the analysis to identify the web application's potential attack surfaces, all the components of our application were explored for vulnerabilities. Common Attack Pattern Enumeration and Classification (CAPEC) list [2] was instrumental in identifying and understanding the potential attack vectors. With that we identified some of the key categories and using that we extensively noted the Attack surface as shown in Table 1.

Category	**Attack surface**
User interface forms and fields	Forms and form fields
	URL Query parameters
Authentication and Access Mechanisms	Login pages
	Unauthorized access
	Access token steal
	Access Token encryption
RESTful APIs	All CRUD API end points
	SSL certificates
Database	Form Fields
	Query Injection in API Request body
	Database server
	Data leak
Error Handling and Logging	Information disclosure through error messages
	Notifications
Cross-Site Scripting (XSS)	User forms
	User-generated content
Cross-Site Request Forgery (CSRF)	Network requests

Table 1: Attack Surface based on category.

Construction of the Attack Tree

This hierarchical diagram allowed us to visualize the primary objectives of an attacker and decompose them into sub-goals, providing a clear framework for understanding how various attacks could be orchestrated and interlinked. Resources such as the Open Web Application Security Project (OWASP)[3] and insights gained from security training platforms like Web Goat[4] played a pivotal role in shaping our approach.

Figure 3 represents the attack tree which is a visual and organized methodology to explore the attack vectors and comprehend the potential exploits. The attack tree encompasses three primary branches: 'Break Authentication,' 'Exploit UI Input Fields,' and 'Exploit Server,' each delving into specific tactics an attacker might employ.

- **Break Authentication**: This branch addresses threats such as 'Brute Force,' 'Steal Password,' and 'Hijack Session,' each with its respective sub-branches, like 'Common Passwords Attack,' 'Credential Stuffing,' 'Shoulder Surfing,' 'Fishing/Scam/Threatening,' and 'Session Sniffing.' These nodes articulate the multilayered nature of threats against authentication mechanisms, emphasizing the necessity for robust authentication and session management protocols.
- **Exploit UI Input Fields**: This branch includes 'Abuse Application Logic,' leading to 'SQL Injection,' and 'Cross-Site Scripting (XSS),' further breaking down into 'Stored XSS' and 'Reflected XSS.' The sub-branches 'Input unexpected values' under 'SQL Injection' highlight the need for strict input validation and sanitation practices.
- **Exploit Server**: Focused on server-side vulnerabilities, this branch details attack types like 'Crash server using bots,' 'Memory Attack,' and 'Exploit Misconfigurations,' branching further into 'Unsecured Database Access' and 'Access open APIs,' with 'Default Credentials' cited as a specific risk. These categories stress the importance of securing server infrastructure and performing continuous configuration management.

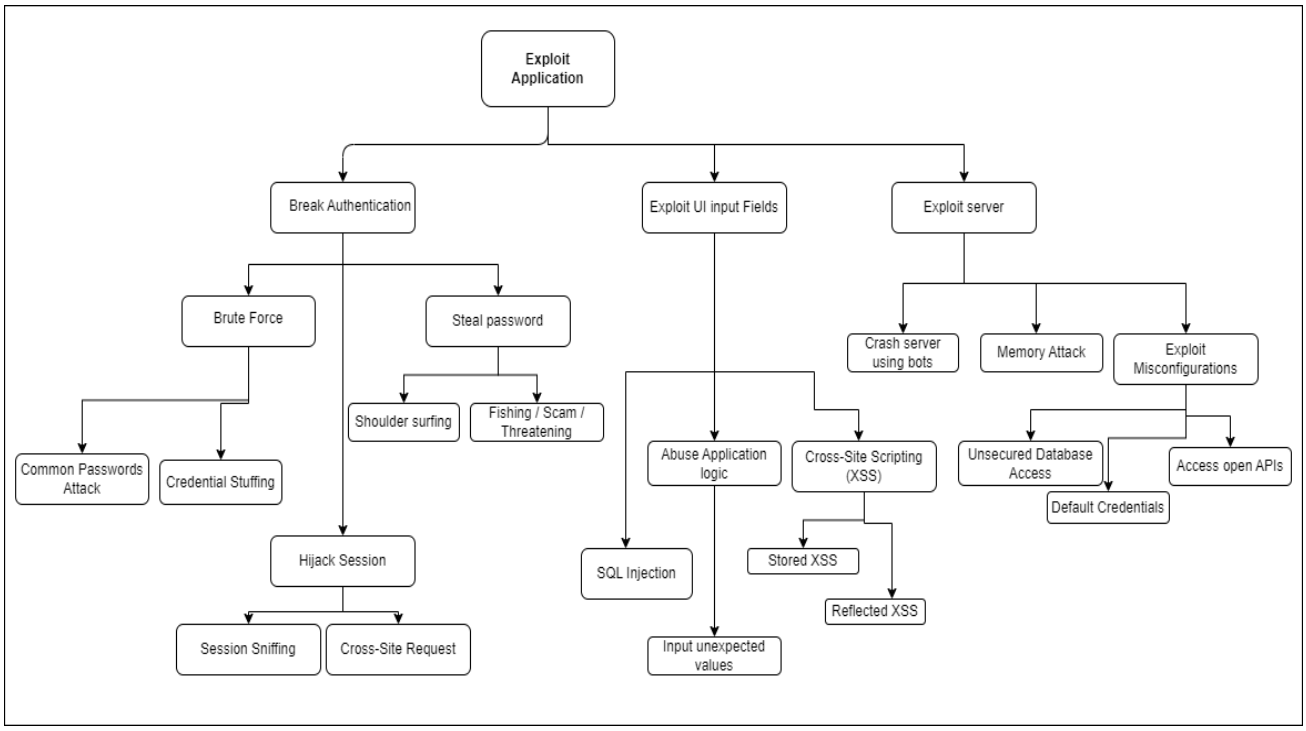

Figure 3. Attack tree for Simple book shop web application.

Technical Controls

The technical architecture of our secure web application is designed with a security-first approach, using the insights from our attack tree and surfaces. This section outlines the security mechanisms and design choices that contribute to the robust security of the application.

Architecture and Design

At the heart of the backend is Spring Boot, chosen for its robust security features and the ease of integrating additional security layers. The frontend is powered by Next.js, selected for its developer

friendly React library support that help in enhancing security and performance of the application. The architecture diagram illustrated in Figure 4 shows seamless integration of these technologies, where each component plays a pivotal role in the overall security posture.

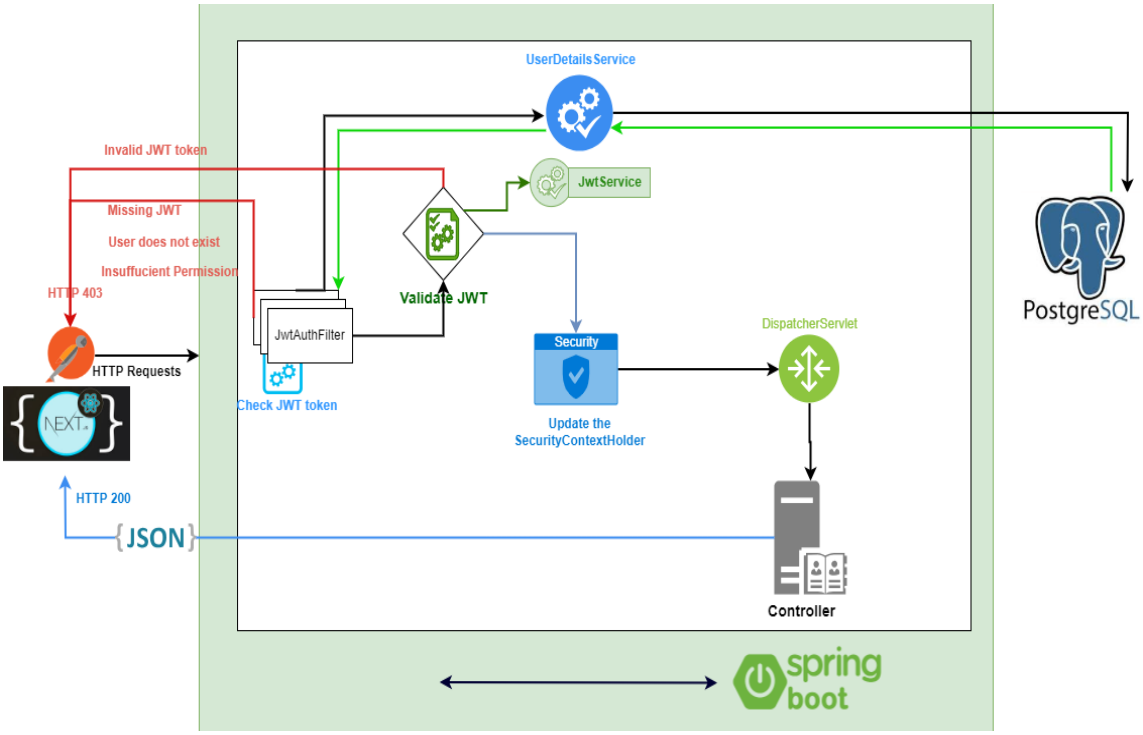

Figure 4. Simple book store Architecture diagram

Authentication and Session Management

For authentication, JWT tokens encrypted with HS256 are used, providing a secure method of verifying user identity. The JwtAuthFilter component is a custom security filter that validates these tokens with each request, ensuring their integrity and authenticity. In case of invalid tokens, the system responds with appropriate HTTP status codes such as 403 Forbidden, reflecting insufficient permissions or non-existent users.

JWT also facilitates proper session management. Upon successful authentication, JWT tokens are issued to users, granting them access to their respective sessions without storing session state on the server, thereby reducing the attack surface for session hijacking.

Input Sanitization and Validation

To prevent XSS attacks, we incorporated the js-xss library in our NextJs front end code. This is a proven tool for sanitizing user inputs on the client side. Both the frontend and the backend perform thorough input validation, ensuring that all data entering the system is checked for potential malicious content, conforming to our defense-in-depth strategy.

Rate Limiting to Mitigate Brute Force Attacks

A rate limiter is implemented to deter brute force attacks, limiting the number of requests a user can make within a certain timeframe. This control is crucial to prevent automated attacks that aim to guess user passwords or overload the system with high volumes of traffic.

Robust Access Control

Access control within the application is strictly enforced through role-based access control (RBAC)[6].

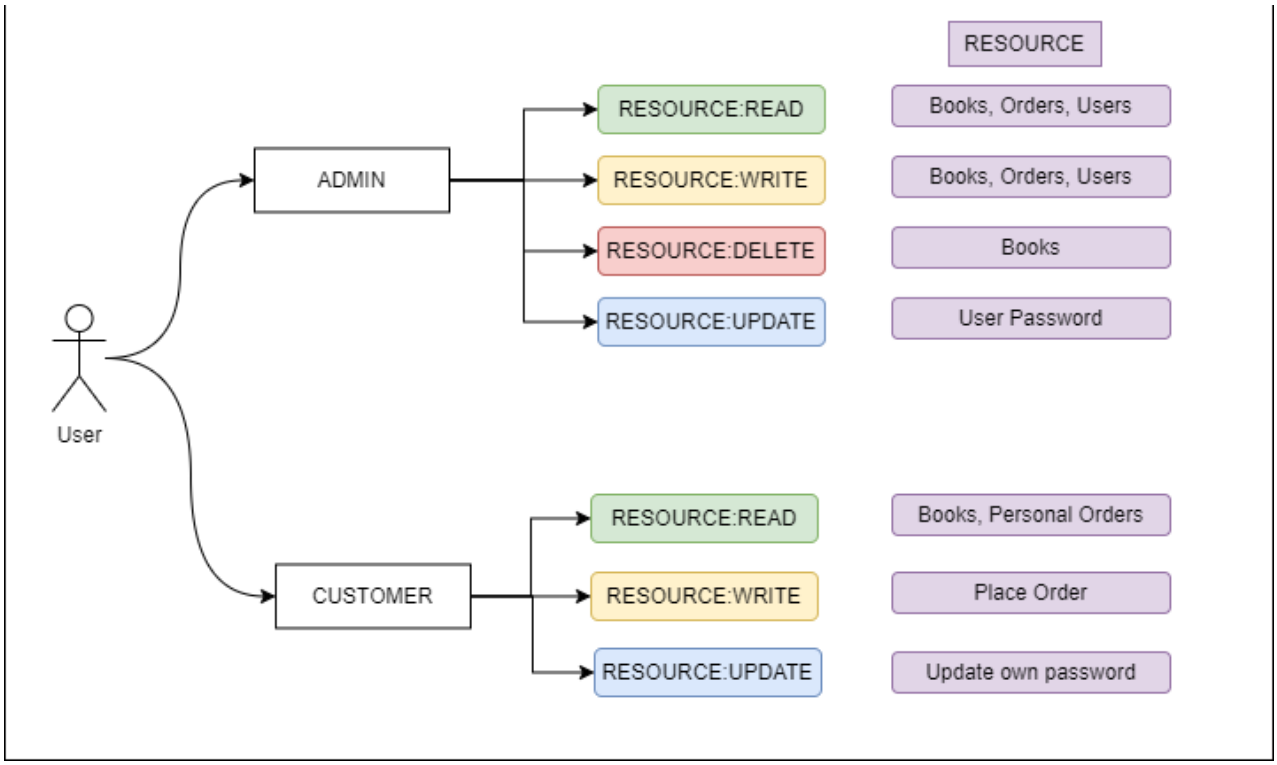

Figure 5: Role-based access control to restrict the unauthorized resource access in our application.

This ensures that customers and administrators have distinctly separate permissions, with each role being granted the least privileges necessary to perform its functions. Figure 5 shows the details on how Role base access control is implemented in our book shop application. This segregation of duties is key in minimizing the risk of unauthorized access to sensitive functionalities and data.

Secure Communication and Data Storage

All communication between the client and server is transmitted over HTTPS, ensuring that data in transit is encrypted and secure from eavesdropping or tampering. PostgreSQL is used for data storage, with configurations hardened to prevent SQL injection and other database-related attacks.

Testing

In the process of validating the security of our web application, we selected the Zed Attack Proxy (ZAP) tool[5], an offering from. ZAP provided us with a suite of capabilities, including both automated and manual security testing features.

The testing approach was carried out as follows:

- **Automated Scanning**: We used ZAP's automated scanner to crawl through the application, identifying common vulnerabilities such as SQL injection, cross-site scripting, and broken authentication mechanisms.
- **Manual Exploration**: Inspired by scenarios encountered in the OWASP WebGoat[4] tutorials, we employed ZAP's manual exploration to simulate sophisticated attack vectors.
- **Network Interception**: With ZAP's network interception capabilities, we rigorously tested our application's resistance to various network-based attacks. This included verifying the robustness of our session management and token-based authentication against potential hijacking or spoofing.

Discussion: Learning Outcomes

This project was a comprehensive exercise in applying software security practices throughout the entire development lifecycle. Key learnings include:

Importance of a Security-First Approach: Integrating security from the earliest stages of development, rather than as an afterthought, significantly reduces vulnerabilities and potential exploits in the final product.

Value of Iterative Development: Utilizing the Spiral model emphasized the importance of regular assessments and revisions, allowing us to address risks promptly and iteratively.

Practical Security Testing: Using tools like ZAP played the critical role of both automated and manual testing in identifying vulnerabilities that could be overlooked by a less thorough examination.

Conclusion

The completion of this secure bookstore web application has underlined the need of integrating security practices throughout the development lifecycle. By following the Spiral model and incorporating rigorous security measures, we tried establishing robust defense against common cyber threats. The use of ZAP for testing and the insights from OWASP and CAPEC standards were crucial in enhancing our application’s security. This project has not only strengthened our technical capabilities but also solidified our understanding of the fundamental importance of software security. Moving forward, the knowledge and experience gained here will be invaluable for future software development endeavors. Software security is a continuous process and its difficult to attain 100% secure application, still we should strive towards achieving maximum security and defence mechanism.

Appendix

The source code of this entire project can be found here: https://github.com/karthik-bhat-b/simple-bookstore/tree/main

References

[1] B. W. Boehm, "A spiral model of software development and enhancement," in Computer, vol. 21, no. 5, pp. 61-72, May 1988, doi: 10.1109/2.59.

[2] CAPEC List https://capec.mitre.org/data/index.html

[3] Attack Surface Analysis Cheat Sheet, OWASP https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html

[4] Web goat, OWASP training https://owasp.org/www-project-webgoat/

[5] OWASP ZAP Training material https://www.zaproxy.org/getting-started/

[6] Role-Based Access Control, Auth0 Okta documentaion

https://auth0.com/docs/manage-users/access-control/rbac#:~:text=Role%2Dbased%20access%20control%20(RBAC,assigning%20permissions%20to%20users%20individually.

[7] Spring boot documentation https://spring.io/projects/spring-boot

[8] NextJs Documentation https://nextjs.org/docs