Secure Coding in Java/JEE

Developing Defensible Applications

Duration: 5 Days

Course Summary

This 5-day hands-on programme teaches Java and JEE developers how to identify, exploit, and remediate the most common security vulnerabilities found in real-world applications. The course covers data validation and injection prevention, authentication and session security, Java platform and API security, and secure development lifecycle practices including code review. Each day combines concept instruction with practical labs against intentionally vulnerable applications, culminating in a full capstone code review and bug hunt. All tools used are open source.

Target Audience

This programme is designed for:

- Java and JEE developers building web-based applications
- Application security engineers and security-focused developers
- Developers preparing for secure coding certifications
- Teams responsible for code review and vulnerability remediation

Day 1: Data Validation and Injection Prevention

Module 1: Web Attack Surface and Input-Based Vulnerabilities

- Web attack proxy testing
- Cross-site scripting (XSS)
- Cross-site request forgery (CSRF)
- SQL injection
- Common injection patterns

Lab Exercises:

1. Use OWASP ZAP (open source) to intercept and analyse traffic from a deliberately vulnerable Java web application and identify 3 injection points.
2. Exploit a provided XSS vulnerability in a test application, then apply output encoding to fix it and verify the exploit no longer works.

Module 2: Input Validation and Output Encoding in Java

- Blacklist vs whitelist validation
- Servlet filtering for validation
- JSR-303 Bean Validation
- Output encoding (HTML, JavaScript)
- Prepared statements vs dynamic SQL

Lab Exercises:

3. Implement a JSR-303 Bean Validation annotation set on a provided Java model class and verify invalid input is correctly rejected.
4. Refactor a provided servlet that uses dynamic SQL query construction to use prepared statements, then test it against a SQL injection payload.

Day 2: Authentication, Session Management and CSRF Defence

Module 3: Authentication and Transport Security

- Multi-factor authentication
- Standard JEE authentication
- TLS for data in transit
- Password storage best practices
- Authorisation with JEE roles

Lab Exercises:

5. Configure a provided Java web application to enforce TLS and verify that plaintext HTTP requests are rejected or redirected.
6. Implement secure password hashing (bcrypt or PBKDF2, open source libraries) on a provided login module and verify stored passwords are never in plaintext.

Module 4: Session Security and CSRF Protection

- Session hijacking
- Session fixation
- Clickjacking defence
- HTML security headers
- CSRF token implementation

Lab Exercises:

7. Demonstrate a session fixation attack on a provided vulnerable application, then implement a fix that regenerates the session ID on login.
8. Add CSRF tokens to a provided form-based Java application and verify a forged cross-site request is correctly rejected.

Day 3: Java Platform and API Security

Module 5: Java Platform Security Internals

- Java Security Manager basics
- JAR signing
- Logging and log forgery prevention
- Java Secure Sockets Extension (JSSE)
- Java Cryptography Architecture (JCA)

Lab Exercises:

9. Sign a sample JAR file using the Java keytool and jarsigner (built-in JDK tools) and verify the signature using jarsigner -verify.
10. Implement secure logging in a provided Java application that sanitises user input before writing to logs, preventing log forgery via newline injection.

Module 6: Numeric Safety, Thread Safety and Web Services Security

- Numeric overflow and precision issues
- Safe arithmetic practices
- Thread safety fundamentals
- Web services security basics
- OAuth overview

Lab Exercises:

11. Identify and fix an integer overflow vulnerability in a provided Java financial calculation method using BigDecimal for precise arithmetic.
12. Review a provided multi-threaded Java class for race conditions, apply synchronisation, and verify the fix using a concurrent test harness.

Day 4: Secure Development Lifecycle and Code Review

Module 7: Secure Software Development Lifecycle (SDLC)

- SDLC security touchpoints
- Threat modelling basics
- Secure coding standards
- Security requirements gathering
- SDLC maturity models

Lab Exercises:

13. Map security touchpoints onto a provided standard SDLC diagram, identifying where threat modelling, code review, and testing should occur.
14. Conduct a lightweight threat model for a provided Java web application feature using STRIDE and document the top 3 identified threats.

Module 8: Automated and Manual Security Code Review

- Static analysis tools
- False positives and false negatives
- Manual code inspection techniques
- Code review checklists
- Triaging findings

Lab Exercises:

15. Run SpotBugs with the Find Security Bugs plugin (open source) against a provided Java codebase and triage the findings, separating true positives from false positives.
16. Conduct a manual security code review of a provided Java module using a structured checklist and document 3 findings the automated scan missed.

Day 5: Applied Security Code Review and Capstone

Module 9: Vulnerability Remediation Workshop

- Common vulnerability patterns recap
- Remediation prioritisation
- Defence-in-depth principles
- Regression testing after fixes
- Documenting security fixes

Lab Exercises:

17. Given a provided Java application with 5 seeded vulnerabilities across the categories covered this week, identify and remediate at least 3 of them.
18. Write regression tests using JUnit (open source) that verify each remediated vulnerability remains fixed and cannot be reintroduced silently.

Module 10: Capstone — Secure Code Review and Bug Hunt

- Full application code review
- Bug hunt simulation
- Exploit verification
- Fix implementation
- Findings report writeup

Lab Exercises:

19. Conduct a full security code review and bug hunt on a provided deliberately vulnerable Java/JEE application, identifying as many distinct vulnerabilities as possible within the session.
20. Select 3 identified vulnerabilities, implement working fixes, verify each exploit no longer succeeds, and produce a findings report summarising the vulnerability, impact, and remediation for each.