

# Web application security in Java

**CELJvWeb | 1 year subscription | e-Learning | Online VM**

The course provides a comprehensive exploration of secure coding principles and practices tailored specifically for Java developers. Starting off from the foundations of cybersecurity, you will understand the consequences of insecure code by examining threats through the lens of the CIA triad.

In the main part of the material, you will systematically walk through the various vulnerabilities outlined in the OWASP Top Ten. As you progress through the modules investigating the intricacies of authentication and authorization, through realizing the practical aspects of cryptography, to tackling injection attacks, you will gain a deep understanding of both theoretical concepts and practical skills for securing Java web applications. Further subjects are aligned to some common software security weakness types, such as error handling, code quality or denial of service.

These modules go beyond just the theory. Not only do they identify vulnerabilities, show their consequences, and detail the best practices, but - through hands-on labs and real-world case studies - they offer practical experience in identifying, exploiting, and mitigating these security risks within Java-based web applications.

So that you are prepared for the forces of the dark side.

So that nothing unexpected happens in your code.

Nothing.

*Note: This course content is available as an e-learning subscription. We reserve a period of 3 months to digest the foundational material, after which we activate shorter learning units on a monthly basis. This gives secure coding efforts an initial boost, and builds up sustained readiness over time. These learning units are indicated in red in the table of contents below.*

## Cyber security skills and drills

### Foundational material



50 LABS



20 CASE STUDIES

### Monthly learning units



2-3 LABS



CASE STUDY

### Audience

Java developers working on Web applications

### Outline

- Cyber security basics
- The OWASP Top Ten
- A01 - Broken Access Control
- A02 - Cryptographic Failures
- Cryptography
- A03 - Injection
- A04 - Insecure Design
- A05 - Security Misconfiguration
- A06 - Vulnerable and Outdated Components
- A07 - Identification and Authentication Failures
- A08 - Software and Data Integrity Failures
- A09 - Security Logging and Monitoring Failures
- A10 - Server-Side Request Forgery
- Wrap up

### Preparedness

General Java and Web development

### Standards and references

OWASP, SEI CERT, CWE and Fortify Taxonomy

### What you'll have learned

- Getting familiar with essential cyber security concepts
- Identify Web application vulnerabilities and their consequences
- Learn the security best practices in C#
- Input validation approaches and principles
- Understanding Web application security issues
- Detailed analysis of the OWASP Top Ten elements
- Putting Web application security in the context of Java
- Going beyond the low hanging fruits

# Table of contents

## › Cyber security basics

### Security basics

- What is security?
- Threat and risk

### Cyber security threats

- [Cyber security threat types – the CIA triad](#)

### Insecure software

- Consequences of insecure software
- Constraints and the market
- The dark side

## › The OWASP Top Ten

### Introduction

- OWASP and the Top 10
- Is it a standard?
- Methodology
- [The OWASP Top Ten 2021](#)

## › A01 – Broken Access Control

### Authorization

- Access control basics
- Confused deputy

### Insecure Direct Object Reference

- Insecure direct object reference (IDOR)
- Path traversal

 *Lab – Insecure Direct Object Reference*

- Path traversal best practices

### Horizontal authorization

- Authorization bypass through user-controlled keys

 *Case study – Authorization bypass on Facebook*

 *Lab – Horizontal authorization*

## File upload

- Unrestricted file upload
- Good practices

 *Lab – Unrestricted file upload*

## Cross-site Request Forgery (CSRF)

 *Lab – Cross-site Request Forgery*

## Cross-site Request Forgery (CSRF) best practices

- CSRF best practices
- CSRF defense in depth

 *Lab – CSRF protection with tokens*

## › A02 – Cryptographic Failures

### › Cryptography

#### Information exposure

- Exposure through extracted data and aggregation

 *Case study – Strava data exposure*

- Leaking system information

#### Cryptography for developers

- Cryptography basics
- Java Cryptographic Architecture (JCA) in brief

#### PRNG

- Random number generation
- Pseudo random number generators (PRNGs)
- Cryptographically secure PRNGs
- Using virtual random streams

 *Case study – Equifax credit account freeze*

#### PRNG in Java

- Weak and strong PRNGs in Java
- Using random numbers in Java

 *Lab – Using random numbers in Java*

#### Hashing

- Hashing basics
- Common hashing mistakes
- Hashing in Java

 *Lab – Hashing in JCA*

## Encryption

- Confidentiality protection
- Symmetric encryption
- [Block ciphers](#)
- Modes of operation
- Modes of operation and IV – best practices

## Encryption in Java

- Symmetric encryption in Java
- Symmetric encryption in Java with streams

 *Lab – Symmetric encryption in JCA*

## Asymmetric encryption

- The RSA algorithm
- Using RSA – best practices
- RSA in Java
- Combining symmetric and asymmetric algorithms

## › A03 – Injection

### Injection problems

- Injection principles
- Injection attacks

### SQL injection

- SQL injection basics

 *Lab – SQL injection*

### SQL injection attack techniques

- Attack techniques
- Content-based blind SQL injection
- Time-based blind SQL injection

### SQL injection best practices

- Input validation
- Parameterized queries

 *Lab – Using prepared statements*

### SQL injection additional considerations

- Database defense in depth

 *Case study – Hacking Fortnite accounts*

## Code injection – OS command injection

- Code injection
- OS command injection

## OS command injection best practices

- Using `Runtime.exec()`
- Using `ProcessBuilder`

 *Case study – Shellshock*

 *Lab – Shellshock*

## HTML injection – Cross-site scripting (XSS)

- [Cross-site scripting basics](#)
- Persistent cross-site scripting
- Reflected cross-site scripting
- Client-side (DOM-based) cross-site scripting

## XSS attacks

 *Lab – Stored XSS*

 *Lab – Reflected XSS*

 *Case study – XSS in Fortnite accounts*


## XSS best practices 1

- Protection principles – escaping
- XSS protection APIs in Java

 *Lab – XSS fix / stored*

## XSS best practices 2

- XSS protection in JSP

 *Lab – XSS fix / reflected*

- Additional protection layers – defense in depth

## Advanced XSS (Unit 5)

- Additional XSS exploitation and protection

 *Lab – Stored XSS in attribute*

 *Lab – XSS fix / stored (in HTML attributes)*

 *Lab – XSS fix / stored (selective)*

- Client-side protection principles

## Template injection (Unit 6)

- Script injection
- Server-side template injection (SSTI)

 *Case study – Template injection in Shopify leading to RCE*

- Client-side template injection (CSTI) in Angular
- Client-side template injection (CSTI) in React

## Expression language (EL) injection (Unit 6)

- SpEL injection

 *Lab – SpEL injection*

- SpEL injection – testing and best practices

## Input validation principles 1 (Unit 1)

- Input validation principles
- Denylists and allowlists
- Data validation techniques

 *Lab – Input validation*

## Input validation principles 2 (Unit 1)

- What to validate – the attack surface
- Where to validate – defense in depth
- When to validate – validation vs transformations

 *Case study – Missing input validation in Upserve*

## Input validation principles 3 (Unit 1)

- Output sanitization
- Encoding challenges
- Unicode challenges

 *Lab – Encoding challenges*

- Validation with regex

## Integer handling problems (Unit 2)

- Representing signed numbers
- Integer visualization
- Integer overflow

 *Lab – Integer overflow*

## Integer handling problems 2 (Unit 2)

- Signed / unsigned confusion in Java

 *Case study – The Stockholm Stock Exchange*

- Integer truncation

## Integer best practices (Unit 2)

- Upcasting
- Precondition testing
- Postcondition testing

- Using big integer libraries

### **Integer best practices in Java (Unit 2)**

- Integer handling in Java

 *Lab – Integer handling*

### **Other numeric problems (Unit 2)**

- Division by zero
- Working with floating-point numbers

### **Path traversal and file validation (Unit 3)**

- Path traversal

 *Lab – Path traversal*

- Path traversal-related examples
- Additional challenges in Windows
- Virtual resources
- Path traversal best practices

 *Lab – Path canonicalization*

### **Unsafe reflection (Unit 4)**

- Reflection without validation

 *Lab – Unsafe reflection*

### **Unsafe native code (JNI) (Unit 4)**

- Native code dependence

 *Lab – Unsafe native code*

- Best practices for dealing with native code

## **› A04 – Insecure Design**

### **Insecure design**

- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder

### **Insecure design – Saltzer and Schroeder 1**

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design

### **Insecure design – Saltzer and Schroeder 2**

- Separation of privilege
- Least privilege



- Least common mechanism
- Psychological acceptability

### **Client-side security**

- Same Origin Policy
- Simple request
- Preflight request
- Cross-Origin Resource Sharing (CORS)

### **Clickjacking**

- Frame sandboxing
- Cross-Frame Scripting (XFS) attacks

 *Lab - Clickjacking*

- Clickjacking beyond hijacking a click

### **Anti-clickjacking best practices**

- Clickjacking protection best practices

 *Lab - Using CSP to prevent clickjacking*

## › **A05 - Security Misconfiguration**

### **Misconfiguration and XML parsing**

- Configuration principles
- XML Entities
- DTD and the entities
- Entity expansion

### **XML External Entity (XXE)**

- File inclusion with external entities
- Server-Side Request Forgery with external entities

 *Lab - External entity attack*

 *Case study - XXE vulnerability in SAP Store*

### **XXE best practices**

- Preventing XXE

 *Lab - Prohibiting DTD*

## › **A06 - Vulnerable and Outdated Components**

### **Vulnerable components and dependencies**

- Using vulnerable components
- Assessing the environment

- Hardening
- Untrusted functionality import

## **Vulnerability management**

- Patch management
- [Vulnerability management](#)
- Vulnerability databases

 *Lab – Finding vulnerabilities in third-party components*

## **> A07 – Identification and Authentication Failures**

### **Authentication**

- Authentication basics
- Multi-factor authentication (MFA)
- Authentication weaknesses

 *Case study – PayPal 2FA bypass*

### **Session security**

- Session management essentials
- Why do we protect session IDs – Session hijacking
- Session fixation
- Session ID best practices

### **Password management**

- Storing account passwords
- Password in transit

 *Lab – Is just hashing passwords enough?*

### **Password storage**

- [Dictionary attacks and brute forcing](#)
- Salting
- Adaptive hash functions for password storage

 *Lab – Using adaptive hash functions in JCA*

### **Password policy**

- [NIST authenticator requirements for memorized secrets](#)

### **Password storage – a case study**

 *Case study – The Ashley Madison data breach*

 *The dictionary attack*

 *The ultimate crack*

 *Exploitation and the lessons learned*

## Additional password management challenges

- Password database migration
- (Mis)handling null passwords

### Password auditing (Unit 7)

- Using password cracking tools
- Password cracking in Windows

 *Lab – Password audit with John the Ripper*

- On-line password brute forcing

 *Lab – On-line password brute forcing*

- Password recovery issues
- Password recovery best practices

 *Lab – Password reset weakness*

### Outbound password management (Unit 7)

- Hard coded passwords
- Best practices

 *Lab – Hardcoded password*

 *Case study – Hard-coded credentials in MyCar vehicle control app*

### Protecting secrets in memory (Unit 7)

- Challenges in protecting memory
- Storing sensitive data in memory

 *Lab – Using secret-handling classes in Java*

## › A08 – Software and Data Integrity Failures

### › Cryptography

#### Integrity protection and MAC

- Integrity protection
- Message Authentication Code (MAC)
- Calculating MAC in Java

 *Lab – Calculating MAC in JCA*

#### Digital signatures

- Digital signature
- Digital signature with RSA
- ECC basics
- Digital signature with ECC

 *Lab – Digital signature with ECDSA in JCA*

## Subresource integrity

- Importing JavaScript

 *Lab – Importing JavaScript*

 *Case study – The British Airways data breach*

## Insecure deserialization


- Serialization and deserialization challenges
- Integrity – deserializing untrusted streams
- Integrity – deserialization best practices
- Using readObject
- Look ahead deserialization

## Property Oriented Programming

- Property Oriented Programming (POP)
- Creating a POP payload

 *Lab – Creating a POP payload*

## POP exploitation and best practices

 *Lab – Using the POP payload*


- Summary – POP best practices
- DoS with deserialization

 *Lab – Billion laughs with deserialization*

## › A09 – Security Logging and Monitoring Failures

### Logging


- Logging and monitoring principles
- Insufficient logging

 *Case study – Plaintext passwords at Facebook*

- Logging best practices


### Log forging (Unit 6)

- CRLF injection
- Log forging
- Log forging – best practices

 *Case study – Log interpolation in log4j*

 *Case study – The Log4Shell vulnerability (CVE-2021-44228)*

 *Case study – Log4Shell follow-ups (CVE-2021-45046, CVE-2021-45105)*

 *Lab – Log4Shell*

## Monitoring (Unit 6)


- Monitoring best practices
- Firewalls and Web Application Firewalls (WAF)
- Intrusion detection and prevention

 *Case study – The Marriott Starwood data breach*

## › A10 – Server-Side Request Forgery

### SSRF

- Server-side Request Forgery (SSRF)

 *Case study – SSRF and the Capital One breach*

## › Error handling (Unit 8)

### Principles

- Error and exception handling principles
- Information exposure through error reporting
- Information leakage via error pages
- Returning a misleading status code
- Reachable assertion

### Exception handling

- In the catch block. And now what?
- Catching NullPointerException
- Empty catch block
- Overly broad throws
- Improper completing of the finally block
- Throwing undeclared checked exceptions
- Swallowed ThreadDeath
- Throwing RuntimeException, Exception, or Throwable

 *Lab – Exception handling mess*

## › Denial of service (Unit 8)

- Flooding
- Resource exhaustion

### Sustained client engagement

- Infinite loop
- Denial of service problems in Java

 *Case study – DoS against Tesla GUI via malicious web page*

- Economic Denial of Sustainability (EDoS)

## Amplification

- Some amplification examples

## Algorithmic complexity issues

- Regular expression denial of service (ReDoS)

### Lab – ReDoS

- Dealing with ReDoS
- Hash table collision
- How do hash tables work?
- Hash collision against hash tables

## > Code quality (Unit 9)

### Introduction

- Code quality and security

### Data – initialization

- Uninitialized variable
- Constructors and destructors
- Class initialization cycles

### Lab – Initialization cycles

### Object Oriented Programming

- Are accessibility modifiers a security feature?
- Accessibility modifiers – best practices
- Overriding and accessibility modifiers
- Inheritance and overriding
- Implementing equals()
- Mutability

### Lab – Mutable object

## > Wrap up

### Software security principles

- Principles of robust programming by Matt Bishop

### Sources and further readings

- Software security sources and further reading
- Java resources