

# Extended Web application security in Java

**CYDJvWeb4d | 4 days | On-site or online | Hands-on**

Your Web application written in Java works as intended, so you are done, right? But did you consider feeding in incorrect values? 16Gbs of data? A null? An apostrophe? Negative numbers, or specifically -1 or -2^31? Because that's what the bad guys will do – and the list is far from complete.

Handling security needs a healthy level of paranoia, and this is what this course provides: a strong emotional engagement by lots of hands-on labs and stories from real life, all to substantially improve code hygiene. Mistakes, consequences, and best practices are our blood, sweat and tears.

The curriculum goes through the common Web application security issues following the OWASP Top Ten but goes far beyond it both in coverage and the details.

All this is put in the context of Java, and extended by core programming issues, discussing security pitfalls of the Java language and the runtime environment.

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

So that nothing unexpected happens.

Nothing.

## Cyber security skills and drills



31 LABS



15 CASE STUDIES

### Audience

Java developers working on Web applications

### Group size

12 participants

### Preparedness

General Java and Web development

### Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Wrap up

### Standards and references

OWASP, SEI CERT, CWE and Fortify Taxonomy

### What you'll have learned

- Getting familiar with essential cyber security concepts
- Understanding how cryptography supports security
- Learning how to use cryptographic APIs correctly in Java
- 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
- Input validation approaches and principles
- Managing vulnerabilities in third party components

# Table of contents

## Day 1

### › Cyber security basics

What is security?





Threat and risk

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


Consequences of insecure software

### › [The OWASP Top Ten 2021](#)

#### **A01 – Broken Access Control**

- Access control basics
- Confused deputy
  - Insecure direct object reference (IDOR)
  - Path traversal
  -  *Lab – Insecure Direct Object Reference*
  - Path traversal best practices
  - 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*

#### **A02 – Cryptographic Failures**

- Information exposure
  - Exposure through extracted data and aggregation
  -  *Case study – Strava data exposure*
- Cryptography for developers
  - Cryptography basics
  - Java Cryptographic Architecture (JCA) in brief
  - Elementary algorithms
    - Random number generation
    - Pseudo random number generators (PRNGs)

- Cryptographically secure PRNGs
- Weak and strong PRNGs in Java
- 🔗 *Lab – Using random numbers in Java*
  - 📖 *Case study – Equifax credit account freeze*
- Hashing
  - Hashing basics
  - Hashing in Java
- 🔗 *Lab – Hashing in JCA*
- Confidentiality protection
  - Symmetric encryption
  - [Block ciphers](#)
  - Modes of operation
  - Modes of operation and IV – best practices
  - 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

## Day 2

### › [The OWASP Top Ten 2021](#)

#### **A03 – Injection**

- Input validation
  - Input validation principles
  - Denylists and allowlists
  - What to validate – the attack surface
  - Where to validate – defense in depth
  - When to validate – validation vs transformations
  - Output sanitization
  - Encoding challenges
  - Unicode challenges
- 🔗 *Lab – Encoding challenges*
- Unsafe reflection
  - Reflection without validation
- 🔗 *Lab – Unsafe reflection*
- Injection

- Injection principles
- Injection attacks
- [SQL injection](#)
  - SQL injection basics
  - 🔗 *Lab – SQL injection*
  - Attack techniques
  - Content-based blind SQL injection
  - Time-based blind SQL injection
  - SQL injection best practices
    - Input validation
    - Parameterized queries
  - 🔗 *Lab – Using prepared statements*
  - Database defense in depth
    - 📖 *Case study – Hacking Fortnite accounts*
  - SQL injection protection and ORM
- Parameter manipulation
  - CRLF injection
  - HTTP header manipulation
    - HTTP response splitting
  - HTTP parameter manipulation
    - HTTP parameter pollution
    - Value shadowing
    - Variable shadowing
- Code injection
  - OS command injection
    - OS command injection best practices
    - Using `Runtime.exec()`
    - Using `ProcessBuilder`
      - 📖 *Case study – Shellshock*
  - 🔗 *Lab – Shellshock*
- Script injection
  - Server-side template injection (SSTI)
- Dangerous file inclusion
- HTML injection – Cross-site scripting (XSS)
  - [Cross-site scripting basics](#)
  - Cross-site scripting types
    - Persistent cross-site scripting
    - Reflected cross-site scripting
    - Client-side (DOM-based) cross-site scripting
  - 🔗 *Lab – Stored XSS*
  - 🔗 *Lab – Reflected XSS*
  - 📖 *Case study – XSS in Fortnite accounts*



- XSS protection best practices
  - Protection principles – escaping
  - XSS protection APIs in Java
  - 🔗 *Lab – XSS fix / stored*
  - 🔗 *Lab – XSS fix / reflected*
  - Client-side protection principles
  - Additional protection layers – defense in depth

## Day 3




### › The OWASP Top Ten 2021

#### **A04 – Insecure Design**


- The STRIDE model of threats
- Secure design principles of Saltzer and Schroeder
  - Economy of mechanism
  - Fail-safe defaults
  - Complete mediation
  - Open design
  - Separation of privilege
  - Least privilege
  - Least common mechanism
  - Psychological acceptability
- Additional principles
  - Work factor
  - Compromise recording
- Client-side security
  - 🔗 *Lab – Client-side security*
    - Same Origin Policy
      - Simple request
      - Preflight request
      - Cross-Origin Resource Sharing (CORS)
      - Relaxing the Same Origin Policy
    - Frame sandboxing
      - Cross-Frame Scripting (XFS) attacks
  - 🔗 *Lab – Clickjacking*
    - Clickjacking beyond hijacking a click
    - Clickjacking protection best practices
  - 🔗 *Lab – Using CSP to prevent clickjacking*
- Some further best practices
  - HTML5 security best practices

- CSS security best practices
- Ajax security best practices
- JSON security
  - JSON validation
  - JSON injection
  - Best practices
    -  *Case study – ReactJS vulnerability in HackerOne*
- XML security
  - XML validation
  - XML injection
  - XPath injection
  - Blind XPath injection
  - XSLT injection
- XML signature
  - XML signature wrapping
    -  *Case study – signature wrapping in single sign-on solutions*

## A05 – Security Misconfiguration

- Configuration principles
- Server misconfiguration
- Cookie security
  - Cookie attributes
- XML entities
  - DTD and the entities
  - Entity expansion
  - External Entity Attack (XXE)
    - File inclusion with external entities
    - Server-Side Request Forgery with external entities
  -  *Lab – External entity attack*
    -  *Case study – XXE vulnerability in SAP Store*
  - Preventing XXE
    -  *Lab – Prohibiting DTD*

## A06 – Vulnerable and Outdated Components

- 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*

- [DevOps, the CI / CD build process and Software Composition Analysis](#)
- Dependency checking in Java
- 🔗 *Lab – Detecting vulnerable components*

## A07 – Identification and Authentication Failures

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

## Day 4

### › [The OWASP Top Ten 2021](#)

## A07 – Identification and Authentication Failures (continued)

- Authentication basics
- Multi-factor authentication (MFA)
- 📖 *Case study – PayPal 2FA bypass*
- Password management
  - Inbound password management
    - Storing account passwords
    - 🔗 *Lab – Is just hashing passwords enough?*
    - [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](#)

## A08 – Software and Data Integrity Failures

- Integrity protection
  - Message Authentication Code (MAC)
    - Calculating MAC in Java
    - 🔗 *Lab – Calculating MAC in JCA*
  - Digital signature
    - Digital signature with RSA
    - Elliptic Curve Cryptography
    - ECC basics
    - Digital signature with ECC
    - Digital signature in Java



- 🔗 *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
  - Look ahead deserialization
  - Property Oriented Programming (POP)
    - Creating a POP payload
- 🔗 *Lab – Creating a POP payload*
- 🔗 *Lab – Using the POP payload*

## **A09 – Security Logging and Monitoring Failures**

- Logging and monitoring principles
- 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*
- Logging best practices

## **A10 – Server-side Request Forgery (SSRF)**

- Server-side Request Forgery (SSRF)
- 📖 *Case study – SSRF and the Capital One breach*

## **Web application security beyond the Top Ten**

- Denial of service
  - Flooding
  - Resource exhaustion
  - Algorithmic complexity issues
    - Regular expression denial of service (ReDoS)
- 🔗 *Lab – ReDoS*
  - Dealing with ReDoS

## **› Wrap up**

### **Secure coding principles**

- Principles of robust programming by Matt Bishop

## **And now what?**

- Software security sources and further reading
- Java resources