

# Cloud application security in Java for AWS

**CYDJvCIdAWS5d | 5 days | On-site or online | Hands-on**

Your cloud 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.

The cloud has become a critical aspect of online services. No matter which model you're using (SaaS, PaaS, IaaS), part of your service is now operated by someone else. This may look like a net positive, but it also greatly expands the attack surface and brings in several new risks that may not be obvious. Have you configured all security settings correctly? Are you prepared for supply chain attacks? How can you protect the confidentiality of user data in the cloud? And most importantly: can the bad guys use your exposure to their advantage?

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 AWS cloud platform.

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

So that nothing unexpected happens.

Nothing.

## Cyber security skills and drills



36 LABS



19 CASE STUDIES

### Audience

Java developers working on Web applications and AWS

### Outline

- Cyber security basics
- The OWASP Top Ten 2021
- Cloud security
- Input validation
- Wrap up

### Group size

12 participants

### Preparedness

General Java and Web development

### Standards and references

OWASP, SEI CERT, CWE and Fortify Taxonomy

### What you'll have learned

- Understand cloud security specialties
- 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
- Managing vulnerabilities in third party components
- Learn to deal with cloud infrastructure security
- Input validation approaches and principles
- Identify vulnerabilities and their consequences
- Learn the security best practices in Java

# Table of contents

## Day 1

### › Cyber security basics

What is security?

Threat and risk

#### Cyber security threat types – the CIA triad

Cyber security threat types – the STRIDE model

Consequences of insecure software

#### **Cloud security basics**

- Cloud infrastructure basics
- The Cloud Cube Model and Zero Trust Architecture
- 📖 *Case study – ChaosDB vulnerability in Azure Cosmos DB*

### › The OWASP Top Ten 2021

#### **A01 – Broken Access Control**

- Access control basics
- Failure to restrict URL access
- Confused deputy
  - Insecure direct object reference (IDOR)
  - 🔗 *Lab – Insecure Direct Object Reference*
  - 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*
- Open redirects and forwards
  - 📖 *Case study – Hacking Fortnite accounts*
  - 📖 *Case study – Unvalidated redirect at Epic Games*
    - Open redirects and forwards – best practices
- [Cross-site Request Forgery \(CSRF\)](#)

 *Lab – Cross-site Request Forgery*

- CSRF best practices
- CSRF defense in depth


 *Lab – CSRF protection with tokens*

## 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*

## Day 2

### > [The OWASP Top Ten 2021](#)

## A02 – Cryptographic Failures (continued)

- Cryptography for developers
  - Confidentiality protection (continued)
    - 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
- Public Key Infrastructure (PKI)
  - Some further key management challenges
  - Certificates
  - Certificates and PKI
  - X.509 certificates
  - Chain of trust
- Transport security
  - The TLS protocol
  - TLS basics
  - TLS features (changes in v1.3)
  - The handshake in a nutshell (v1.3)
  - TLS best practices
  - HTTP Strict Transport Security (HSTS)

## A03 – 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*
    - 📖 *Case study – Hacking Fortnite accounts*
  - SQL injection protection and ORM
- NoSQL injection
  - NoSQL injection basics
  - NoSQL injection in MongoDB
  - NoSQL injection in DynamoDB
- Parameter manipulation
  - CRLF injection
- Code injection
  - OS command injection
    - OS command injection best practices
    - Using `Runtime.exec()`
      - 📖 *Case study – Shellshock*
  - 🔗 *Lab – Shellshock*


- 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*
    - 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
- Client-side security
  - Same Origin Policy
    - Simple request
    - Preflight request
    - Cross-Origin Resource Sharing (CORS)
  - Frame sandboxing
    - Cross-Frame Scripting (XFS) attacks
    - 🔗 *Lab – Clickjacking*
      - Clickjacking beyond hijacking a click
      - Clickjacking protection best practices

 *Lab – Using CSP to prevent clickjacking*

## **A05 – Security Misconfiguration**

- Configuration principles
- Server misconfiguration
- AWS configuration best practices
  - Common AWS misconfiguration issues
  - AWS configuration best practices
- 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**

- Authentication
  - Authentication basics
  - Multi-factor authentication (MFA)
  - Time-based One Time Passwords (TOTP)
  - Authentication weaknesses
  - [Spoofing on the Web](#)

- 📖 *Case study – PayPal 2FA bypass*
  - User interface best practices
- 🔗 *Lab – On-line password brute forcing*
- Session management
  - Session management essentials
  - Why do we protect session IDs – Session hijacking
  - Session fixation
  - Session invalidation
  - Session ID best practices
- Identity and access management (IAM)
  - Identity and access management in AWS
  - Groups, roles and credentials
  - Access tokens

## Day 4

### › [The OWASP Top Ten 2021](#)

#### **A07 – Identification and Authentication Failures (continued)**

- Password management
  - Inbound password management
    - Storing account passwords
    - Password in transit
    - 🔗 *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](#)
        - 📖 *Case study – The Ashley Madison data breach*
        - 📖 *The ultimate crack*
        - 📖 *Exploitation and the lessons learned*
  - Password database migration

#### **A08 – Software and Data Integrity Failures**

- Integrity protection
  - Message Authentication Code (MAC)
  - Digital signature
    - Elliptic Curve Cryptography
    - ECC basics



- Digital signature with ECC
- 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
- Detection and monitoring
  - Utilizing AWS monitoring for security
  - Protecting logs
  - The AWS Security Hub

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

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

## **› Cloud security**

### **AWS security**

- Security considerations
  - AWS and security
  - The AWS shared responsibility model
  - AWS cloud compliance
  - AWS hardening

- Security tools for AWS
- Data security in AWS
- Policies
- Protecting data at rest
- Storing cryptographic keys on AWS
- Protecting data in transit

## JSON security

- Best practices

 *Case study – ReactJS vulnerability in HackerOne*

# Day 5

## > Cloud security




### Container security

- Container security concerns
- Containerization, virtualization and security
- The attack surface
- Docker security
  - Docker and security
  - Docker security best practices
-  *Lab – Static analysis of Docker images*
  - Hardening Docker
- Kubernetes security
  - The Kubernetes architecture and security
  - Securing Kubernetes hosts
  - Best practices for Kubernetes access control
  - Building secure Kubernetes images
  - Secure deployment of Kubernetes containers
  - Protecting Kubernetes deployments at runtime
-  *Case study – Azurescape*

## > The OWASP Top Ten 2021

### Web application security beyond the Top Ten

- Code quality
  - Data handling
  - Initialization and cleanup
    - Constructors and destructors

- Class initialization cycles
  -  *Lab – Initialization cycles*
  - The finalize() method – best practices
- Language elements
  - Using obsolete language elements
  - Portability flaw
  - Misusing Java language elements
- Object oriented programming pitfalls
  - Inheritance and overriding
  - Mutability
    -  *Lab – Mutable object*
- Denial of service
  - Flooding
  - Resource exhaustion
  - Algorithmic complexity issues
    - Regular expression denial of service (ReDoS)
      -  *Lab – ReDoS*
    - Dealing with ReDoS

## › 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*

Validation with regex

### **Integer handling problems**


- Representing signed numbers
- Integer visualization
- Integer overflow

 *Lab – Integer overflow*


- Signed / unsigned confusion in Java

 *Case study – The Stockholm Stock Exchange*

- Integer truncation

- Best practices
    - Upcasting
    - Precondition testing
    - Postcondition testing
    - Integer handling in Java
-  *Lab – Integer handling*

### **Unsafe reflection**

- Reflection without validation
-  *Lab – Unsafe reflection*

## **> Wrap up**

### **Secure coding principles**

- Principles of robust programming by Matt Bishop

### **And now what?**

- Software security sources and further reading
- Java resources