

Understanding Secure Coding and Threat Models

Focusing on Java and C++ as the target example - 1 day

FA-0482 · 1 day

Software Development · Cybersecurity

Introduction

This one-day intensive training is tailored for developers, security professionals, and team leads who work with **Java** and **C++**, equipping them with essential knowledge on writing secure code in these languages.

By focusing on the OWASP Top 10 vulnerabilities, secure coding practices, and key threat models like STRIDE and DREAD, this course emphasizes practical techniques for improving security in both **Java** and **C++** codebases.

Whether you're experienced in software development or just starting with secure coding, this training will arm you with real-world examples, industry-demanded practices, and insight into the risks of poor security practices.

Led by an instructor with over 30 years of industry experience, this course skips academic theory to deliver actionable, practical skills that will enable you to identify and address vulnerabilities, implement best practices, and ensure compliance with modern security standards across Java and C++ projects.

Learning outcomes

By the end of this training, participants will:

- Understand each item in the OWASP Top 10 2021 list and how these vulnerabilities can manifest in **Java** and **C++** code.
- Learn secure coding practices specific to **Java** and **C++** development environments.
- Recognize common pitfalls in **Java** and **C++** that lead to insecure code.

- Apply security checklists and versioning control strategies to consistently enforce secure coding practices in **Java** and **C++** projects.
- Use threat modeling techniques (STRIDE and DREAD) to identify risks in **Java** and **C++** applications.
- Gain insight into real-world examples of attacks and prevention methods tailored to **Java** and **C++** systems.

Prerequisites

- Solid experience of **Java** or Modern **C++** programming.
- Experience with web applications and cybersecurity principles.
- Development experience with restful headless applications.
- Working knowledge on Linux

Training outline

1. Overview of OWASP Top 10 2021

1. Introduction to OWASP

Understanding the mission of OWASP and its significance in secure coding for web and desktop applications built in **Java** and **C++**.

2. Importance of the OWASP Top 10 in Secure Coding

Why **Java** and **C++** developers must consider these vulnerabilities when building secure applications.

2. OWASP Top 10 2021 Vulnerabilities and Associated Attacks

(with Focus on Java and C++)

Broken Access Control

1. **Java Example:** Misconfigured access controls in Spring-based web apps.
2. **C++ Example:** Insufficient validation in backend services leading to privilege escalation.

Cryptographic Failures

1. **Java Example:** Improper use of the java.security package leading to insecure encryption.
2. **C++ Example:** Usage of weak encryption libraries in embedded systems.

Injection

1. **Java Example:** SQL injection vulnerabilities through improper handling of user inputs in JDBC.
2. **C++ Example:** Command injection in applications that use system calls.

Insecure Design

1. **Java:** Poor design in microservices architecture leading to data leakage.
2. **C++:** Unchecked buffer sizes in C++ resulting in stack-based buffer overflows.

Security Misconfiguration

1. **Java:** Exposing sensitive endpoints without authentication in Servlet configurations.
2. **C++:** Improper configuration of file permissions in Linux-based systems.

Vulnerable and Outdated Components

1. **Java:** Usage of vulnerable third-party libraries in Maven or Gradle.
2. **C++:** Linked outdated C++ libraries that lack modern security patches.

Identification and Authentication Failures

1. **Java:** Insecure session handling in web applications.
2. **C++:** Weak password handling mechanisms in GUI-based applications.

Software and Data Integrity Failures

1. **Java:** Attacks exploiting insecure software update mechanisms.
2. **C++:** Supply chain attacks through outdated binaries.

Security Logging and Monitoring Failures

1. **Java:** Lack of logging in Spring Boot applications leading to unnoticed intrusions.
2. **C++:** Missing error logs for failed authentications.

Server-Side Request Forgery (SSRF)

1. **Java:** Improper validation of HTTP request handling in APIs.
2. **C++:** SSRF attacks exploiting legacy communication protocols in embedded systems.

3. What is Secure Coding in Java and C++?

Definition and Importance of Secure Coding

How secure coding differs for **Java** (managed memory, object-oriented) and **C++** (manual memory management, performance optimization).

1. The high stakes of security lapses: financial, legal, and reputational damages.

4. Compliance for Secure Coding

1. Security Standards and Regulatory Compliance

How secure coding practices fit into regulations like GDPR, HIPAA, and PCI-DSS for **Java** and **C++** applications.

2. Internal vs. External Compliance

Ensuring adherence to both company-specific and broader industry standards.

5. Common Mistakes in Secure Coding for Java and C++

Misunderstanding of Input Validation

1. Improper validation leading to injection attacks in both languages.

Incorrect Use of Cryptography

1. Use of deprecated cryptographic libraries in older **C++** systems and improper cryptographic initialization in **Java**.

Overly Permissive Security Configurations

1. Default configurations that expose unnecessary functionalities in **Java** and **C++** applications.

6. Secure Coding Checklist

1. Comprehensive Checklist to Ensure Secure Coding Practices in Java and C++

Include input validation, cryptography, authentication, error handling, and versioning control.

2. Versioning Control and Best Practices for Security

Applying security patches using version control tools like Git, and regular audits for **Java** and **C++** codebases.

7. Threat Modeling Techniques: STRIDE and DREAD for Java and C++

1. STRIDE Model

Application of STRIDE to identify threats in **Java** web applications and **C++** system software.

DREAD Threat Risk Ranking Model

Quantifying and ranking potential risks in **Java** and **C++** applications using the DREAD methodology.

1. Practical examples of prioritizing security patches and mitigations for both **Java** and **C++** systems.

This comprehensive one-day training provides actionable techniques to implement secure coding practices in **Java** and **C++**. You'll gain confidence to minimize vulnerabilities, comply with industry standards, and build robust, secure applications.