Stop the Buffer Overflows. Stop the Vulnerabilities. Start Writing Secure Code. Become an ECSP
Software defects, bugs, and flaws in the logic of the program are consistently the cause for software vulnerabilities. Analysis by software security professionals has proven that most vulnerabilities are due to errors in programming. Hence, it has become crucial for organizations to educate their software developers about secure coding practices.

Attackers scan for security vulnerabilities in applications and servers and attempt to use these vulnerabilities to steal secrets, corrupt programs and data, and gain control of computer systems and networks. Sound programming techniques and best practices should be used to develop high quality code to prevent web application attacks. Secure programming is a defensive measure against attacks targeted towards application systems.
Today, Java is embedded in more than 3 billion devices such as laptops, data centers, game consoles, super computers, mobile phones, smart cards, and many more. Java is widely adopted because of its platform and architecture independent characteristics that encourages developers and industry alike.

ECSP-Java is comprehensive course that provides hands-on training covering Java security features, policies, strengths, and weaknesses. It helps developers understand how to write secure and robust Java applications and provides advanced knowledge in various aspects of secure Java development that can effectively prevent hostile and buggy code. The end result of security Java coding practices include saving valuable effort, money, time, and possibly the reputation of organizations using Java-coded applications.
What Will You Learn?

Students in this course will acquire knowledge in the following areas:

01 Java security principles and secure coding practices

02 Java Security Platform, Sandbox, JVM, Class loading, Bytecode verifier, Security Manager, security policies, and Java Security Framework

03 Secure Software Development Lifecycle, threat modelling, software security frameworks, and secure software architectures

04 Best practices and standards and guidelines for secure file input/output and serialization

05 Java input validation techniques, validation errors, and best practices

06 Java exceptions, erroneous behaviors, and the best practices to handle or avoid them

07 Secure authentication and authorization processes

08 Java Authentication and Authorization Service (JAAS), its architecture, Pluggable Authentication Module (PAM) Framework, and access permissions through Java Security Model

09 Secure Java concurrency and session management that includes Java Memory Model, Java Thread Implementation methods, secure coding practices, and guidelines for handling threads, race conditions, and deadlocks

10 Core security coding practices of Java Cryptography that includes Encryption, KeyGenerator, implementation of Cipher Class, Digital Signatures, Secret Keys, and key management

11 Various Java application vulnerabilities such as Cross-Site Scripting (XSS), Cross Site Request Forgery (CSRF), Directory Traversal vulnerability, HTTP Response Splitting attack, Parameter Manipulation, Injection Attacks and their countermeasures

12 Coding testing and review techniques and practices

EC-Council Certified Secure Programmer (JAVA)
Prerequisites
You must be well-versed with Java programming language.

Duration
3 Days (9:00 AM – 5:00 PM)

Who Should Attend
The ECSP certification is intended for programmers who are responsible for designing and building secure Windows/Web based applications with Java. It is designed for developers who have Java development skills.

EC-Council Certified Secure Programmer (JAVA)
EC-Council Certified Secure Programmer (Java)

Exam Information

Number of Questions: 50
Passing Score: 70%
Test Duration: 2 Hours
Test Format: Multiple Choice
Test Delivery: EC-Council Exam Center, Prometric
Exam Prefix: 312-94

Certification

The ECSP Java 312-94 exam will be conducted on the last day of training. Students need to pass the online exam to receive the ECSP certification.
Course Outline

Module 01 Introduction to Java Security

• Vulnerability Disclosure Growth
• Impact of Vulnerabilities and Associated Costs
• Security Incidents
• Software Security Failure Costs
• Need for Secure Coding
• Java Security Overview
• Java Security Platform
• Java Virtual Machine (JVM)
• Class Loading
• Bytecode Verifier
• Class Files
• Security Manager
• Java Security Policy
• Java Security Framework
| Why Secured Software Development is needed? | System Quality Requirements Engineering Steps |
| Characteristics of a Secured Software | Secure Code Review |
| Software Security Framework | Step 2: Perform Preliminary Scan |
| Guidelines for Designing Secure Software | Code Review |
| Threat Modeling | Source Code Analysis Tools |
| Threat Modeling Approaches | Advantages and Disadvantages of Static Code Analysis |
| Web Application Model | Advantages and Disadvantages of Dynamic Code Analysis |
| Threat Modeling Process | LAPSE: Web Application Security Scanner for Java |
| SDL Threat Modeling Tool | FindBugs: Find Bugs in Java Programs |
| Secure Design Considerations | Coverity Static Analysis |
| Secure Java Patterns and Design Strategies | Coverity Dynamic Analysis |
| Secure Java Coding Patterns | Veracode Static Analysis Tool |
| Secure Code Patterns for Java Applications | Source Code Analysis Tools For Java |
| Secure Coding Guidelines | Fuzz Testing |
| System Quality Requirements Engineering | |
Module 03 File Input and Output and Serialization

- File Input and Output in Java
- The java.io package
- Character and Byte Streams in Java
- Reader and Writer
- Input and Output Streams
- All File creations should Accompany Proper Access Privileges
- Handle File-related Errors cautiously
- All used Temporary Files should be removed before Program Termination
- Release Resources used in Program before its Termination
- Prevent exposing Buffers to Untrusted Code
- Multiple Buffered Wrappers should not be created on a single InputStream
- Capture Return Values from a method that reads a Byte or Character to an Int
- Avoid using write() Method for Integer Outputs ranging from 0 to 255
- Ensure Reading Array is fully filled when using read() Method to Write in another Array
- Raw Binary Data should not be read as Character Data
- Ensure little endian data is represented using read/write methods
- Ensure proper File Cleanup when a Program Terminates
- File Input/Output Best Practices
- File Input and Output Guidelines
- Serialization
- Implementation Methods of Serialization
- Serialization Best Practices
- Secure Coding Guidelines in Serialization
Module 04 Input Validation

- Percentage of Web Applications Containing Input Validation Vulnerabilities
- Input Validation Pattern
- Validation and Security Issues
- Impact of Invalid Data Input
- Data Validation Techniques
- Whitelisting vs. Blacklisting
- Input Validation using Frameworks and APIs
- Regular Expressions
- Vulnerable and Secure Code for Regular Expressions
- Servlet Filters
- Struts Validator
- Struts Validation and Security
- Data Validation using Struts Validator
- Avoid Duplication of Validation Forms
- Struts Validator Class
- Enable the Struts Validator
- Secure and Insecure Struts Validator Code
- HTML Encoding
- Vulnerable and Secure Code for HTML Encoding
- Vulnerable and Secure Code for Prepared Statement
- CAPTCHA
- Stored Procedures
- Character Encoding
- Input Validation Errors
- Best Practices for Input Validation
Module 05 Error Handling and Logging

- Exception and Error Handling
- Example of an Exception
- Handling Exceptions in Java
- Exception Classes Hierarchy
- Exceptions and Threats
- Erroneous Exceptional Behaviors
- Do's and Don’ts in Exception Handling
- Best Practices for Handling Exceptions in Java
- Logging in Java
- Example for Logging Exceptions
- Logging Levels
- Log4j and Java Logging API
- Java Logging using Log4j
- Vulnerabilities in Logging
- Logging: Vulnerable Code and Secure Code
- Secured Practices in Logging
Module 06 Authentication and Authorization

- Percentage of Web Applications Containing Authentication Vulnerabilities
- Percentage of Web Applications Containing Authorization Bypass Vulnerabilities
- Introduction to Authentication
- Java Container Authentication
- Authentication Mechanism Implementation
- Declarative v/s Programmatic Authentication
- Declarative Security Implementation
- Programmatic Security Implementation
- Java EE Authentication Implementation Example
- Basic Authentication
- How to Implement Basic Authentication?
- Form-Based Authentication
- Form-Based Authentication Implementation
- Implementing Kerberos Based Authentication
- Secured Kerberos Implementation
- Configuring Tomcat User Authentication Setup
- Client Certificate Authentication in Apache Tomcat
- Client Certificate Authentication
- Certificate Generation with Keytool
- Implementing Encryption and Certificates in Client Application
- Authentication Weaknesses and Prevention
- Introduction to Authorization
- JEE Based Authorization
- Access Control Model
- Discretionary Access Control (DAC)

- Mandatory Access Control (MAC)
- Role-based Access Control (RBAC)
- Servlet Container
- Authorizing users by Servlets
- Securing Java Web Applications
- Session Management in Web Applications
- EJB Authorization Controls
- Common Mistakes
Module 07 Java Authentication and Authorization Service (JAAS)

- Java Authentication and Authorization (JAAS)
- JAAS Features
- JAAS Architecture
- Pluggable Authentication Module (PAM) Framework
- JAAS Classes
- JAAS Subject and Principal
- Authentication in JAAS
- Subject Methods doAs() and doAsPrivileged()
- Impersonation in JAAS
- JAAS Permissions
- LoginContext in JAAS
- JAAS Configuration
- Locating JAAS Configuration File
- JAAS CallbackHandler and Callbacks
- Login to Standalone Application
- JAAS Client
- LoginModule Implementation in JAAS
- Phases in Login Process
- Java EE Application Architecture
- Java EE Servers as Code Hosts
- Tomcat Security Configuration
- Best Practices for Securing Tomcat
- Declaring Roles
- HTTP Authentication Schemes
- Securing EJBs
Module 08 Java Concurrency and Session Management

- Percentage of Web Applications Containing a Session Management Vulnerability
- Java Concurrency/Multithreading
- Concurrency in Java
- Different States of a Thread
- Java Memory Model: Communication between Memory of the Threads and the Main Memory
- Creating a Thread
- Thread Implementation Methods
- Threads Pools with the Executor Framework
- Concurrency Issues
- Do not use Threads Directly
- Avoid calling Thread.run() Method directly
- Use ThreadPool instead of Thread Group
- Use notify all() for Waiting Threads
- Call await() and wait() methods within a Loop
- Avoid using Thread.stop()
- Gracefully Degrade Service using Thread Pools
- Use Exception Handler in Thread Pool
- Avoid Overriding Thread-Safe Methods with the non Thread-Safe Methods
- Use this Reference with caution during Object Construction
- Avoid using Background Threads while Class Initialization
- Avoid Publishing Partially Initialized Objects
- Race Condition
- Secure and Insecure Race Condition Code
- Deadlock
- Avoid Synchronizing high level Concurrency Objects using Intrinsic Locks
- Avoid Synchronizing Collection View if the program can access Backing Collection
- Synchronize Access to Vulnerable Static fields prone to Modifications
- Avoid using an Instance Lock to Protect Shared Static Data
- Avoid multiple threads Request and Release Locks in Different Order
- Release Actively held Locks in Exceptional Conditions
- Ensure Programs do not Block Operations while Holding Lock
- Use appropriate Double Checked Locking Idiom forms
- Class Objects that are Returned by getClass() should not be Synchronized
- Synchronize Classes with private final lock Objects that Interact with Untrusted Code
- Objects that may be Reused should not be Synchronized
- Be Cautious while using Classes on Client Side that do not Stick to their Locking Strategy
- Deadlock Prevention Techniques
- Secured Practices for Handling Threads
- Session Management
- Session Tracking
- Session Tracking Methods
- Types of Session Hijacking Attacks
- Countermeasures for Session Hijacking
- Countermeasures for Session ID Protection
- Guidelines for Secured Session Management
Module 09 Java Cryptography

- Percentage of Web Applications Containing Encryption Vulnerabilities
- Need for Java Cryptography
- Java Security with Cryptography
- Java Cryptography Architecture (JCA)
- Java Cryptography Extension (JCE)
- Attack Scenario: Inadequate/Weak Encryption
- Encryption: Symmetric and Asymmetric Key
- Encryption/Decryption Implementation Methods
- SecretKeys and KeyGenerator
- The Cipher Class
- Attack Scenario: Man-in-the-Middle Attack
- Digital Signatures
- The Signature Class
- The SignedObjects
- The SealedObjects
- Insecure and Secure Code for Signed/Sealed Objects
- Digital Signature Tool: DigiSigner
- Secure Socket Layer (SSL)
- Java Secure Socket Extension (JSSE)
- SSL and Security
- JSSE and HTTPS
- Insecure HTTP Server Code
- Secure HTTP Server Code
- Attack Scenario: Poor Key Management
- Keys and Certificates
- Key Management System
- KeyStore
- Implementation Method of KeyStore Class
- KeyStore: Temporary Data Stores
- Secure Practices for Managing Temporary Data Stores
- KeyStore: Persistent Data Stores
- Key Management Tool: KeyTool
- Digital Certificates
- Certification Authorities
- Signing Jars
- Signing JAR Tool: Jarsigner
- Signed Code Sources
- Code Signing Tool: App Signing Tool
- Java Cryptography Tool: JCrypTool
- Java Cryptography Tools
- Do's and Don’ts in Java Cryptography
- Best Practices for Java Cryptography
Module 10 Java Application Vulnerabilities

- Average Number of Vulnerabilities Identified within a Web Application
- Computers reporting Exploits each quarter in 2011, by Targeted Platform or Technology
- Introduction to Java Application
- Java Application Vulnerabilities
- Cross-Site Scripting (XSS)
- Cross Site Request Forgery (CSRF)
- Directory Traversal
- HTTP Response Splitting
- Parameter Manipulation
- XML Injection
- SQL Injection
- Command Injection
- LDAP Injection
- XPATH Injection
- Injection Attacks Countermeasures