EC-Council Certified Secure Programmer (Java)
ECSP Java Course

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 a must for organizations to educate their software developers about secure coding practices.

Attackers try to find security vulnerabilities in the applications or servers and then try 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 can be used to develop high quality code to prevent web application attacks. Secure programming is a defensive measure against attacks targeted towards application systems.

Course Description

Today, Java is embedded in 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 ability to be a platform and architecture independent characteristic that encourages developers and industry alike. Therefore Java is a here to stay promising technology in today’s digital world.

Uses and applications of Java are multifold, working through its speed, reliability and security especially in this networked world implementing business processes, desktop software development, web development, mobile development, etc. Its large set of library classes, applets, APIs, object oriented programming proves to be the lifeline of modern computing.

One of the founding principles laid down by the maker’s of Java while developing it was robustness and code security. After all who needs Java Security? Can there be any difference for an ordinary computer user as a result? Definitely the answer is Yes, Java Security is important for a number of distinct sets of people. Web users are on a constant brink side risk of Internet threats such as malware, hackers, espionage and software vulnerabilities resulting in cumulative multi-billion losses, privacy and confidentiality issues. Software companies that neglect the security aspect while coding repay 100% relative Cost of fixing security defects in maintenance compared to 1% in designing phase. Network administrators are always at the receiving end as security software vulnerabilities and bugs can bring down computing infrastructure altogether. Businesses lose millions of dollars due to information security issues damaging their brand image, customer trust and confidence. For example, Heartland Payment Systems, a payment processing company in 2008 exposed 134 million credit card information due to a security vulnerability in their data systems. The Sony Playstation network, a digital gaming corporation similarly exposed sensitive information worth $101.6 million to hackers. Intruders breached TJX Corporation’s (an apparel and fashion store chain company) databases exposing 95 million credit cards due to an information system bug.

Every other programming language has got strengths and weaknesses of its own, similarly is the case with Java. This comprehensive course deep drives the reader on how Java security works, its security features, policies, strengths, and weaknesses. It helps in how best a programmer can write code to optimize application security. It gives basic to advanced knowledge in various aspects of secure Java development that can effectively prevent hostile and buggy code saving valuable effort, money, time and reputation of organizations as ‘prevention is always better than cure’.
What Will You Learn?

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

1. Advanced knowledge of Java Security principles through advanced concepts and secured coding practices

2. Core Java Security concepts such as Java Security Platform, Sandbox, JVM, Class loading, Bytecode verifier, Security Manager, security policies as well as Java Security Framework


4. Secure File Input/Output as well as secure coding in Serialization process, their best practices, standards and guidelines

5. Input validation in Java, its various techniques, input validation errors and best practices in implementing them

6. Java’s fundamental concepts, the ‘Exceptions’; erroneous behavior, best coding practices in avoiding and handling them

7. Secure Authentication and Authorization process implementation in Java applications, their various standard practices and guidelines.


9. Secure Java Concurrency and Session Management that includes Java Memory Model, Java Thread Implementation methods, secure coding practices, 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. Strengths and weaknesses of Java applications, various Java Application Vulnerabilities namely Cross-Site Scripting (XSS), Cross Site Request Forgery (CSRF), Directory Traversal vulnerability, HTTP Response Splitting attack, Parameter Manipulation, Injection Attacks their prevention techniques, secure coding best practices and countermeasures to avoid Java application vulnerabilities
**Prerequisites**

You must be well-versed with Java programming language.

**Who Should Attend**

The ECSP certification is intended for programmers who are responsible for designing and building secure applications with JAVA. It is designed for developers who have JAVA development skills.

This course will significantly benefit individuals who are entering the world of software programming or professional who want to advance their career in the programming field.

**Duration**

3 Days (9:00 AM – 5:00 PM)

**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 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
  • Sandbox
  • Java Virtual Machine (JVM)
  • Class Loading
  • Bytecode Verifier
  • Class Files
  • Security Manager
  • Java Security Policy
  • Java Security Framework
  • Java Authentication and Authorization Service (JAAS)
  • Java Secure Socket Extension (JSSE)
  • Java Generic Security Service API (JGSS)
  • Simple Authentication and Security Layer API (Java SASL API)
Module 02 Secure Software Development

- Why Secured Software Development is needed?
- Why Security Bugs in SDLC?
- Characteristics of a Secured Software
- Security Enhanced Software Development Life Cycle
- Software Security Framework
- Secure Architecture and Design
- Design Principles for Secure Software Development
- Guidelines for Designing Secure Software
- Threat Modeling
- Threat Modeling Approaches
- Web Application Model
- Threat Modeling Process
  - Security Objectives
  - Application Overview
  - Application Decomposition
  - Identify Threats
  - Identify and Prioritize Vulnerabilities
- SDL Threat Modeling Tool
- Secure Design Considerations
- Secure Java Patterns and Design Strategies
- Secure Java Coding Patterns
- Secure Code Patterns for Java Applications
- Secure Coding Guidelines
- System Quality Requirements Engineering
- System Quality Requirements Engineering Steps
- Software Security Testing
  - Security Testing Objectives
  - Types of Security Testing
  - Prerequisites for Security Testing
- Software Security Testing at Every Phase of SDLC
- Security Testing Web Applications
- Secure Code Review
- Step 1: Identify Security Code Review Objectives
- Step 2: Perform Preliminary Scan
- Step 3: Review Code for Security Issues
- Step 4: Review for Security Issues Unique to the Architecture
- Code Review
  - Input Validation and XSS
  - Buffer Overflow and Command Injection
  - SQL Injection
  - Exception Handling and Authentication
  - Session Management and Cookie Management
  - Denial-of-Service
- Source Code Analysis Tools
  - Advantages and Disadvantages of Static Code Analysis
  - Advantages and Disadvantages of Dynamic Code Analysis
- LAPSE: Web Application Security Scanner for Java
- FindBugs: Find Bugs in Java Programs
- Coverity Static Analysis
- Coverity Dynamic Analysis
- Veracode Static Analysis Tool
- Source Code Analysis Tools For Java
- Fuzz Testing
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
- Ensure Sensitive Log Information is not Leaked outside a Trust Boundary
- File Input/Output Best Practices
- File Input and Output Guidelines
- Serialization
- Implementation Methods of Serialization
- Maintain Compatible Serialization Form

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
- Secure and Insecure Struts Validation Code
- Struts Validator Class
- Secure and Insecure Code for Struts Validator Class
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
  - Suppressing or Ignoring Checked Exceptions
  - Disclosing Sensitive Information
  - Logging Sensitive Data
  - Restoring Objects to Prior State, if a method fails
  - Avoid using Statements that suppress Exceptions
  - Prevent Access to Untrusted Code that terminates JVM
  - Never catch java.lang.NullPointerException
  - Never allow methods to throw RuntimeException, Exception, or Throwable
  - Never throw Undeclared Checked Exceptions
  - Never let Checked Exceptions escape from Finally Block
- 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

- Enable the Struts Validator
- Secure and Insecure Struts Validator Code
- Check for Similar Number of Fields in Action Form and Validation Form
- Secure Code that Implements Similar Number of Fields in Action Form and Validation Form
- HTML Encoding
- Vulnerable and Secure Code for HTML Encoding
- Prepared Statement
- Vulnerable and Secure Code for Prepared Statement
- CAPTCHA
  - Sample Code for Creating CAPTCHA
  - Sample Code for CAPTCHA Verification
  - Sample Code for Displaying CAPTCHA
- Stored Procedures
  - Vulnerable and Secure Code for Stored Procedures
  - Stored Procedure for Securing Input Validation
- Character Encoding
  - Vulnerable and Secure Code for Character Encoding
  - Checklist for Character Encoding
- Input Validation Errors
  - Improper Sanitization of Untrusted Data
  - Improper Validation of Strings
  - Improper Logging of User Inputs
  - Improper Incorporation of Malicious Inputs into Format Strings
  - Inappropriate use of Split Characters in Data Structures
  - Improper Validation of Non-Character Code Points
- Best Practices for Input Validation
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
  - Brute Force Attack
  - Web-based Enumeration Attack
  - Weak Password Attacks
- 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
  - Declarative Security with EJBs
  - Programmatic Security with EJBs
- 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
  - Authentication Steps in JAAS
- Authorization in JAAS
  - Authorization Steps in JAAS
- Subject Methods doAs() and doAsPrivileged()
- Impersonation in JAAS
- JAAS Permissions
- LoginContext in JAAS
  - Creating LoginContext
  - LoginContext Instantiation
- JAAS Configuration
- Locating JAAS Configuration File
• JAAS CallbackHandler and Callbacks
• Login to Standalone Application
• JAAS Client
• LoginModule Implementation in JAAS
  • Methods Associated with LoginModule
  • LoginModule Example
• 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
  • Extending the java.lang.Thread Class
  • Implementing the java.langRunnable Interface
• 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 ThreadGroup

• Use notifyall() 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
• Ordering of Locks
• Lock Timeout
• Deadlock Detection
• Secured Practices for Handling Threads
• Session Management
• Session Tracking
• Session Tracking Methods
• Cookies
• URL Rewriting
• Hidden Fields
• Session Objects
• Session Vulnerabilities
• Types of Session Hijacking Attacks
• Countermeasures for Session Hijacking
• Countermeasures for Session ID Protection
• Best Coding Practices for Session Management
• Checklist to Secure Credentials and Session IDs
• 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
- 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
  - Insecure Code for Signed Code Sources
  - Secure Code for Signed Code Sources
- Code Signing Tool: App Signing Tool
- Java Cryptography Tool: JCrypTool
- Java Cryptography Tools
- Do's and Don'ts in Java Cryptography
  - Avoid using Insecure Cryptographic Algorithms
  - Avoid using Statistical PRNG, Inadequate Padding and Insufficient Key Size
  - Implement Strong Entropy
  - Implement Strong Algorithms
- 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 Scripting (XSS) Countermeasures
  - Cross Site Request Forgery (CSRF) Countermeasures
- Directory Traversal
  - Directory Traversal Countermeasures
- HTTP Response Splitting
  - HTTP Response Splitting Countermeasures
- Parameter Manipulation
  - Parameter Manipulation and Countermeasures
- XML Injection
- SQL Injection
- Command Injection
- LDAP Injection
- XPATH Injection
- Injection Attacks Countermeasures