Java Security

Duration: 3 Days

Day 1

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
- Use Proper Signatures Methods
- Avoid Serializing Sensitive and Unencrypted Data
- Perform Security Manager Checks
- Avoid Serialization of Inner classes
- Maintain a copy of Private Mutable Components while Deserialization
- Avoid Calling Overridable Methods
- Prevent Memory or Resource Leaks
- Avoid Overwriting of Externalizable Objects
- Serialization Best Practices
- Secure Coding Guidelines in Serialization

Day 2

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

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 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

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

Day 3

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.lang.Runnable 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 Methods of KeyGenerator Class
- Creating SecretKeys with KeyGenerator Class
- Key Generation Tool: RSA Key Generation Utility
- The Cipher Class
- Implementation Methods of Cipher Class
- Insecure Code for Cipher Class using DES Algorithm
- Secure Code for Cipher Class using AES Algorithm
- Attack Scenario: Man-in-the-Middle Attack
- Digital Signatures
- The Signature Class
- Implementation Methods of Signature Class
- The SignedObjects
- Implementing Methods of SignedObjects
- The SealedObjects
- Implementation Methods of SealedObject
- Insecure and Secure Code for Signed/Sealed Objects
- Digital Signature Tool: DigiSigner

- Secure Socket Layer (SSL)
- Java Secure Socket Extension (JSSE)
- SSL and Security
- Example 1
- Example 2
- 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
- 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)
- 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

Application-Level Cryptography

- The Java Cryptography Architecture
- Secure Random Number Generation
- The KeyStore API
- Digital Signature
- Hashing
- Password Hashing
- Why Hashing Isn't Enough
- Salts
- Key Lengthening and Key Strengthening
- Slow Algorithms
- The Java Cryptography Extensions
- The SecretKey and KeyGenerator Types
- Symmetric Encryption
- Choosing Algorithms and Key Sizes
- Dangerous Practices
- Storing and Managing Keys

Secure Development Practices

- Secure Development Cycle
- Penetration Testing
- Secure Code Review
- Error Handling and Information Leakage
- Failing to a Secure Mode

- Designing for Failure
- Back Doors
- Logging Practices
- Appropriate Content for Logs
- Auditing Strategies