

Web application security for PCI DSS 4.0 compliance

Customized for training requirements compliance (6.2.2)

CYDPCIDSSMs | 2-5-days | On-site or online | Hands-on

The course aligns PCI DSS Requirements 4.0 with foundational concepts of secure coding, and thus natively serves the compliance with secure coding training requirement (6.2.2).

The comprehensive journey starts with laying down the basics of security, cyber security and secure coding, as well as PCI DSS itself. Participants then delve deep into secure configuration, cryptography and protection against malicious software, aligned to the Requirements.

Requirement 6 specifically focuses on development and maintenance of secure systems and software, and the corresponding chapter is therefore the broadest one. Topics include bug categorization, secure design and implementation principles. Approaches to input validation are followed up by some specific issues, like integer handling, injection or XSS. We also discuss common software security weaknesses, like error handling or code quality, as well as security of some commonly used data structures like XML or JSON.

The curriculum continues with a thorough examination of authentication, authorization and accountability challenges, and concludes with security testing methodology and specific testing techniques.

The course goes beyond theory, providing hands-on labs and real-world case studies from the financial sector. Participants emerge with a heightened understanding of secure coding best practices, ensuring the development of applications that safeguard sensitive payment card data and comply with the stringent requirements of PCI DSS 4.0 on a yearly basis.

Note: This course is customized for PCI DSS Requirement 6.2.2 compliance, concerning both the content and the delivery structure. The table of contents reflects the Java version, but the course can also come with C#, Python and Node content.

Aligned to the compliance requirements, the delivery of the training days can be done separately, breaking the course into separate events that can span across year boundaries, aligned to your long-term compliance plans.

Please contact us to customize the course to your technology stack and compliance needs.

Cyber security skills and drills



38 LABS



17 CASE STUDIES

Audience

Managers and developers working on Web applications in banking and finance

Outline

- Cyber security basics
- PCI DSS 4.0 introduction
- Requirement 1
- Requirement 2
- Requirement 3 and 4
- Requirement 5
- Requirement 6
- Requirement 7
- Requirement 8
- Requirement 9
- Requirement 10
- Requirement 11
- Requirement 12
- Wrap up

Group size

12 participants

Preparedness

General development

Standards and references

OWASP, SEI CERT, CWE and Fortify Taxonomy

What you'll have learned

- Getting familiar with essential cyber security concepts
- Having essential understanding of PCI DSS requirements
- Understanding how cryptography supports security
- Input validation approaches and principles
- Access control design and implementation guidelines
- Understanding security testing methodology and approaches
- Getting familiar with security testing techniques and tools

Table of contents

Day 1

› **Cyber security basics**

What is security?

Threat and risk

Cyber security threat types – the CIA triad

Consequences of insecure software

› **PCI DSS 4.0 introduction**

Overview

Data elements

Control objectives and requirements

- Build and Maintain a Secure Network and Systems
- Protect Account Data
- Maintain a Vulnerability Management Program
- Implement Strong Access Control Measures
- Regularly Monitor and Test Networks
- Support Information Security with Organizational Policies and Programs

The PCI Software Security Framework (SSF)

- Overview
- Secure Software Lifecycle
- Secure Software Standard

› Requirement 1

Requirement 1: Install and Maintain Network Security Controls

› Requirement 2

Requirement 2: Apply Secure Configurations to All System Components

Configuration principles

Server misconfiguration


Cookie security

- Cookie security best practices
- Cookie attributes

XML parsing

- 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*

› Requirement 3 and 4

Requirement 3: Protect Stored Account Data

Requirement 4: Protect Cardholder Data with Strong Cryptography ...

Information exposure

- Exposure through extracted data and aggregation

 *Case study – Strava data exposure*

- System information leakage
 - Leaking system information
- Information exposure best practices

Cryptography for developers

- Cryptography basics
- Java Cryptographic Architecture (JCA) in brief



- Elementary algorithms
 - Hashing
 - Hashing basics
 - Common hashing mistakes
 - Hashing in Java
 - 🔗 *Lab – Hashing in JCA*
- 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
 - Key exchange and agreement
 - Key exchange
 - Diffie–Hellman key agreement algorithm
 - Key exchange pitfalls and best practices

Day 2

› Requirement 3 and 4

Cryptography for developers

- Elementary algorithms
 - Random number generation
 - Pseudo random number generators (PRNGs)
 - Cryptographically secure PRNGs
 - Seeding
 - Using virtual random streams
 - Weak and strong PRNGs in Java
 - 🔗 *Lab – Using random numbers in Java*
 - True random number generators (TRNG)
 - Assessing PRNG strength
 - 📖 *Case study – Equifax credit account freeze*
- Integrity protection
 - Message Authentication Code (MAC)

- Calculating MAC in Java
 -  *Lab – Calculating MAC in JCA*
- Digital signature
 - Digital signature with RSA
 - Elliptic Curve Cryptography
 - ECC basics
 - Digital signature with ECC
 - Digital signature in Java
 -  *Lab – Digital signature with ECDSA in JCA*
- Public Key Infrastructure (PKI)
 - Some further key management challenges
 - Certificates
 - Certificates and PKI
 - X.509 certificates
 - Chain of trust
 - PKI actors and procedures
 - Certificate revocation

› Requirement 5

Requirement 5: Protect All Systems and Networks from Malicious Software

Intrusion detection

- Firewalls and Web Application Firewalls (WAF)
- Intrusion detection and prevention

 *Case study – The Marriott Starwood data breach*

› Requirement 6

Requirement 6: Develop and Maintain Secure Systems and Software

Security in the Software Development Lifecycle

- Securing the SDLC
- OWASP Software Assurance Maturity Model (SAMM)
- Microsoft Security Development Lifecycle (MS SDL)
- Build Security In Maturity Model (BSIMM)

Categorization of bugs



- The Seven Pernicious Kingdoms
- Common Weakness Enumeration (CWE)
- CWE Top 25 Most Dangerous Software Weaknesses

- SEI CERT
- SEI CERT Coding Standards
- Rules and recommendations

Security by 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


Input validation

- Input validation principles
- Denylists and allowlists
-  *Case study – Missing input validation in Upserve*
- 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

Day 3

› Requirement 6

Input validation


- 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
 - Using big integer libraries
 - Integer handling in Java
- 🔗 *Lab – Integer handling*
- Files and streams
 - Path traversal
 - 🔗 *Lab – Path traversal*
 - Path traversal best practices
 - 🔗 *Lab – Path canonicalization*
- Unsafe reflection
 - Reflection without validation
 - 🔗 *Lab – Unsafe reflection*
- Unsafe native code
 - Native code dependence
 - 🔗 *Lab – Unsafe native code*
 - Best practices for dealing with native code

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*
 - Additional considerations
 - 📖 *Case study – Hacking Fortnite accounts*
- Code injection
 - OS command injection
 - OS command injection best practices
 - Using `Runtime.exec()`

 *Case study – Shellshock*


 *Lab – Shellshock*



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

XML security

- XML validation
- XML injection
 - XPath injection
 - Blind XPath injection

JSON security

- JSON validation
- JSON injection
- Best practices

 *Case study – ReactJS vulnerability in HackerOne*

Day 4

> Requirement 6

Code quality

- Code quality and security
- Data handling
 - Initialization and cleanup
 - Constructors and destructors
 - Class initialization cycles

 *Lab – Initialization cycles*

- Unreleased resource
 - The finalize() method – best practices
- Object oriented programming pitfalls
 - Accessibility modifiers
 - Are accessibility modifiers a security feature?
 - Accessibility modifiers – best practices
 - Overriding and accessibility modifiers
 - Inheritance and overriding
 - Mutability

 *Lab – Mutable object*

- Cloning

Errors

- Error and exception handling principles
- Error handling
 - Returning a misleading status code
 - Reachable assertion
 - Information exposure through error reporting
 - Information leakage via error pages
- Exception handling
 - In the catch block. And now what?
 - Catching NullPointerException
 - Empty catch block
 - Overly broad throws
 - Improper completing of the finally block
 - Throwing undeclared checked exceptions
 - Swallowed ThreadDeath
 - Throwing RuntimeException, Exception, or Throwable


 *Lab – Exception handling mess*

> Requirement 7

Requirement 7: Restrict Access to System Components and Cardholder Data...

Authorization

- 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*

➤ Requirement 8

Requirement 8: Identify Users and Authenticate Access to System Components

Authentication

- Authentication basics
- Multi-factor authentication (MFA)
- Time-based One Time Passwords (TOTP)
- Authentication weaknesses
- 📖 *Case study – Equifax Argentina*
- [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
 - [Cross-site Request Forgery \(CSRF\)](#)
 - 🔗 *Lab – Cross-site Request Forgery*
 - CSRF best practices
 - CSRF defense in depth
 - 🔗 *Lab – CSRF protection with tokens*

Day 5

> Requirement 8

Authentication

- 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](#)
 - Password change
 - Password recovery issues
 - Password recovery best practices
 - 🔗 *Lab – Password reset weakness*
 - 📖 *Case study – The Ashley Madison data breach*
 - 📖 *The ultimate crack*
 - 📖 *Exploitation and the lessons learned*
 - Password database migration
 - Outbound password management
 - Hard coded passwords
 - Best practices
 - 🔗 *Lab – Hardcoded password*
 - Protecting sensitive information in memory
 - Challenges in protecting memory
 - Storing sensitive data in memory
 - 🔗 *Lab – Using secret-handling classes in Java*





> Requirement 9

Requirement 9: Restrict Physical Access to Cardholder Data

> Requirement 10

Requirement 10: Log and Monitor All Access to System Components and Cardholder Data



Logging and Monitoring

- 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
- Testing for logging issues

> Requirement 11

Requirement 11: Test Security of Systems and Networks Regularly

Security testing methodology

- Security testing – goals and methodologies
- Overview of security testing processes
- Identifying assets
- Assigning security requirements
-  *Lab – Identifying and rating assets*
- Threat modeling
 - Attacker profiling
 - SDL threat modeling
 - Mapping STRIDE to DFD
 - DFD example
-  *Lab – SDL threat modelling with OWASP Threat Dragon*
- Attack trees
- Attack tree example
- Misuse cases
- Misuse case examples

- Risk analysis

Security testing techniques and tools

- Fuzzing

> Requirement 12

Requirement 12: Support Information Security with Organizational Policies and Programs

> Wrap up

Secure coding principles

- Principles of robust programming by Matt Bishop

And now what?

- Software security sources and further reading
- Java resources