Certified Secure Software Lifecycle Professional (CSSLP) Boot Camp

Length: 5 days
Format: Classroom
Time: Day

About This Course

The Certified Secure Software Lifecycle Professional (CSSLP) validates that software professionals have the expertise to incorporate security practices authentication, authorization and auditing into each phase of the software development lifecycle (SDLC), from software design and implementation to testing and deployment.

Required Exams

Candidates earn their CSSLP Certification by successfully completing one exam:

CSSLP Certification exam

Audience Profile

Course Objectives

The broad spectrum of topics included in the CSSLP Common Body of Knowledge (CBK) ensure its relevancy across all disciplines in the field of information security. Successful candidates are competent in the following eight domains:

* Secure Software Concepts
* Secure Software Requirements
* Secure Software Design
* Secure Software Implementation/Programming
* Secure Software Testing
* Secure Lifecycle Management
* Software Deployment, Operations, and Maintenance
* Supply Chain and Software Acquisition

Outline

Domain 1: Secure Software Concepts 1.1 Core Concepts * Confidentiality (e.g., covert, overt, encryption)
* Integrity (e.g., hashing, digital signatures, code signing, reliability, alterations, authenticity)
* Availability (e.g., failover, replication, clustering, scalability, resiliency)
* Authentication (e.g., multifactor authentication, identity & access management, single sign-on, federated identity)
* Authorization (e.g., access controls, entitlements)
* Accountability (e.g., auditing, logging)
* Nonrepudiation (e.g., PKI, digital signatures)

1.2 Security Design Principles * Least privilege (e.g., access control, need-to-know, run-time privileges)
* Separation of duties (e.g., multi-party control, secret sharing and splitting)
* Defense in depth (e.g., layered controls, input validation, security zones)
* Fail safe (e.g., exception handling, non-verbose errors, deny by default)
* Economy of mechanism (e.g., single sign-on)
* Complete mediation (e.g., cookie management, session management, caching of credentials)
* Open design (e.g., peer reviewed algorithm)
* Least common mechanism (e.g., compartmentalization/isolation)
* Psychological acceptability (e.g., password complexity, screen layouts)
* Leveraging existing components (e.g., common controls, libraries)
* Eliminate single point of failure

Domain 2: Secure Software Requirements2.1 Identify Security Requirements * Functional
* Non-functional
* Policy decomposition (e.g., internal and external requirements)
* Legal, regulatory, and industry requirements

2.2 Interpret Data Classification Requirements * Data ownership (e.g., data owner, data custodian)
* Labeling (e.g., sensitivity, impact)
* Types of data (e.g., structured, unstructured data)
* Data life-cycle (e.g., generation, retention, disposal)

2.3 Identify Privacy Requirements * Data anonymization
* User consent
* Disposition

2.4 Develop Misuse and Abuse Cases 2.5 Include Security in Software Requirement Specifications 2.6 Develop Security Requirement Traceability MatrixDomain 3: Secure Software Design3.1 Perform Threat Modeling * Understand common threats (e.g., APT, insider threat, common malware, third party/supplier)
* Attack surface evaluation

3.2 Define the Security Architecture * Control identification and prioritization
* Distributed computing (e.g., client server, peer-to-peer, message queuing)

* Service-oriented architecture (e.g., enterprise service bus, web services)
* Rich internet applications (e.g., client side exploits or threats, remote code execution, constant connectivity)
* Pervasive/ubiquitous computing (e.g., IoT, wireless, location-based, RFID, near field communication, sensor networks)
* Embedded (e.g., control systems, firmware)
* Cloud architectures (e.g., software as a service, platform as a service, infrastructure as a service)
* Mobile applications
* Hardware platform concerns

3.3 Performing Secure Interface Design * Security management interfaces, out-of-band management, log interfaces
* Upstream/downstream dependencies (e.g., key and data sharing between apps)
* Protocol design choices (e.g., APIs, weaknesses, state, models)

3.4 Performing Architectural Risk Assessment 3.5 Modeling (Non-Functional) Security Properties and Constraints 3.6 Model and Classify Data 3.7 Evaluate and Select Reusable Secure Design * Credential management (e.g., X.509 and SSO)
* Flow control (e.g., proxies, firewalls, protocols, queuing)
* Data loss prevention (DLP)
* Virtualization (e.g., software defined network, hypervisor)
* Trusted computing (e.g., TPM, TCB)
* Database security (e.g., encryption, triggers, views, privilege management)
* Programming language environment (e.g., CLR, JVM)
* Operating system controls and services.

3.8 Perform Design Security Review 3.9 Design Secure Assembly Architecture for Component-Based Systems * Client side data storage
* * Network attached storage

3.10 Use Security Enhancing Architecture and Design Tools 3.11 Use Secure Design Principles and PatternsDomain 4: Secure Software Implementation/Programming4.1 Follow Secure Coding Practices * Declarative versus imperative (programmatic) security
* Concurrency
* Output sanitization (e.g., encoding)
* Error and exception handling
* Input validation Logging & auditing
* Session management
* Safe APIs
* Type safety
* Memory management (e.g., locality, garbage collection)
* Configuration parameter management (e.g., start-up options)
* Tokenizing
* Sandboxing
* Cryptography (e.g., storage, agility, encryption, algorithm selection)

4.2 Analyze Code for Security Vulnerabilities * Code reuse
* Vulnerability databases/lists (e.g., OWASP Top 10, CWE)
* Static analysis
* Dynamic analysis
* Manual code review
* Peer review

4.3 Implement Security Controls 4.4 Fix Security Vulnerabilities 4.5 Look for Malicious Code 4.6 Securely Reuse Third Party Code or Libraries 4.7 Securely Integrate Components * Systems-of-systems integration (e.g., security testing and analysis)

4.8 Apply Security during the Build Process * Anti-tampering techniques (e.g., code signing, obfuscation)
* Compiler switches

4.9 Debug Security Errors Domain 5: Secure Software Testing5.1 Develop Security Test Cases * Attack surface validation Penetration
* Fuzzing (e.g., generated, mutated)
* Scanning (e.g., vulnerability, content, privacy)
* Simulation (e.g., environment and data)
* Failure (e.g., fault injection, stress testing, break testing)
* Cryptographic validation (e.g., PRNG)
* Regression
* Continuous (e.g., synthetic transactions)
* Unit testing

5.2 Develop Security Testing Strategy and Plan * Functional security testing (e.g., logic)
* Nonfunctional security testing (e.g., reliability, performance, scalability)
* Testing techniques (e.g., white box and black box)
* Environment (e.g., interoperability, test harness)
* Standards (e.g., ISO, OSSTMM, SEI)

5.3 Identify Undocumented Functionality 5.4 Interpret Security Implications of Test Results 5.5 Classify and Track Security Errors * Bug tracking (e.g., defects, errors and vulnerabilities)
* Risk Scoring (e.g., CVSS)

5.6 Secure Test Data * Privacy
* Referential integrity

5.7 Develop or Obtain Security Test Data 5.8 Perform Verification and Validation Testing (e.g., IV&V) Domain 6: Secure Lifecycle Management6.1 Secure Configuration and Version Control 6.2 Establish Security

Milestones 6.3 Choose a Secure Software Methodology 6.4 Identify Security Standards and Frameworks 6.5 Create Security Documentation 6.6 Develop Security Metrics 6.7 Decommission Software * End of life policies
* Credential removal, configuration removal, license cancellation
* Data destruction

6.8 Report Security Status 6.9 Support Governance, Risk, and Compliance (GRC) * Regulations and compliance
* Legal (e.g., intellectual property, breach notification)
* Standards and guidelines (e.g., ISO, PCI, NIST, OWASP, SAFECODE, OpenSAMM, BSIMM)
* Risk management
* Terminology (e.g., threats, vulnerability, residual risk, controls, probability, impact)
* Technical risk vs business risk
* Strategies (e.g., mitigate, accept, transfer, avoid)

Domain 7: Software Deployment, Operations, and Maintenance7.1 Perform Implementation Risk Analysis 7.2 Release Software Securely 7.3 Securely Store and Manage Security Data * Credentials
* Secrets
* Keys/certificates
* Configurations

7.4 Ensure Secure Installation * Bootstrapping (e.g., key generation, access, management)
* Least privilege
* Environment hardening
* Secure activation (e.g., credentials, white listing, device configuration, network configuration, licensing, etc.)

7.5 Perform Post-Deployment Security Testing 7.6 Obtain Security Approval to Operate * Risk acceptance (e.g., exception policy, sign-off)

7.7 Perform Security Monitoring (e.g., managing error logs, audits, meeting SLAs, CIA metrics) 7.8 Support Incident Response * Root cause analysis

7.9 Support Patch and Vulnerability Management 7.10 Support Continuity of Operations * Backup, archiving, retention
* Disaster recovery.

Domain 8: Supply Chain and Software Acquisition8.1 Analyze Security of Third Party Software 8.2 Verify Pedigree and Provenance * Secure transfer
* System sharing/interconnections
* Code repository security
* Build environment security

* Cryptographically- hashed, digitally signed components

8.3 Provide Security Support to the Acquisition Process * Audit of security policy compliance
* Vulnerability/incident response and reporting
* Service-level agreements (SLAs)
* Maintenance and support structure (e.g., community versus commercial)
* Assessment of software engineering/SDLC approaches
* Information systems security policy compliance
* Security track record

Product deployment and sustainment controls (e.g., upgrades, secure configuration, custom code extension, operational readiness, GPL requirements)