CSSLP®

Certified Secure Software Lifecycle Professional

An (ISC)² Certification

Certification **Exam Outline**

Effective Date: September 15, 2020

(ISC)² INSPIRING A SAFE AND SECURE CYBER WORLD.

About CSSLP

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.

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 Architecture and Design
- Secure Software Implementation
- Secure Software Testing
- Secure Software Lifecycle Management
- Secure Software Deployment, Operations, Maintenance
- Secure Software Supply Chain

Experience Requirements

A candidate is required to have a minimum of four years of cumulative paid Software Development Lifecycle (SDLC) professional work experience in one or more of the eight domains of the (ISC)² CSSLP CBK, or three years of cumulative paid SDLC professional work experience in one or more of the eight domains of the CSSLP CBK with a four-year degree leading to a Baccalaureate, or regional equivalent in Computer Science, Information Technology (IT) or related fields.

If you don't have the required experience to become a CSSLP, you may become an Associate of (ISC)² by successfully passing the CSSLP examination. You will then have five years to earn the four years required experience. You can learn more about CSSLP experience requirements and how to account for part-time work and internships at www.isc2.org/Certifications/CSSLP/experience-requirements.

Accreditation

CSSLP is in compliance with the stringent requirements of ANSI/ISO/IEC Standard 17024.

Job Task Analysis (JTA)

(ISC)² has an obligation to its membership to maintain the relevancy of the CSSLP. Conducted at regular intervals, the Job Task Analysis (JTA) is a methodical and critical process of determining the tasks that are performed by CSSLP credential holders. The results of the JTA are used to update the examination. This process ensures that candidates are tested on the topic areas relevant to the roles and responsibilities of today's practicing information security professionals.

CSSLP Examination Information

Length of exam	3 hours
Number of items	125
Item format	Multiple choice
Passing grade	700 out of 1000 points
Exam availability	English
Testing center	Pearson VUE Testing Center

CSSLP Examination Weights

Domains	Weight
1. Secure Software Concepts	10%
2. Secure Software Requirements	14%
3. Secure Software Architecture and Design	14%
4. Secure Software Implementation	14%
5. Secure Software Testing	14%
6. Secure Software Lifecycle Management	11%
7. Secure Software Deployment, Operations, Maintenance	12%
8. Secure Software Supply Chain	11%
Total:	**100%**

Domain 1:
Secure Software Concepts

1.1 Core Concepts

» Confidentiality (e.g., covert, overt, encryption)
» Integrity (e.g., hashing, digital signatures, code signing, reliability, modifications, authenticity)
» Availability (e.g., redundancy, replication, clustering, scalability, resiliency)
» Authentication (e.g., multifactor authentication (MFA), identity & access management (IAM), single sign-on (SSO), federated identity)
» Authorization (e.g., access controls, permissions, entitlements)
» Accountability (e.g., auditing, logging)
» Nonrepudiation (e.g., digital signatures, block chain)

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 split knowledge)
» Defense in depth (e.g., layered controls, input validation, security zones)
» Resiliency (e.g., fail safe, fail secure, no Single Point of Failure (SPOF))
» Economy of mechanism (e.g., Single Sign-On (SSO), password vaults, resource)
» Complete mediation (e.g., cookie management, session management, caching of credentials)
» Open design (e.g., Kerckhoffs's principle)
» Least common mechanism (e.g., compartmentalization/isolation, white-listing)
» Psychological acceptability (e.g., password complexity, screen layouts, Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA), biometrics)
» Component reuse (e.g., common controls, libraries)
» Diversity of defense (e.g., geographical diversity, technical diversity, distributed systems)

Domain 2:
Secure Software Requirements

2.1 Define Software Security Requirements

» Functional (e.g., business requirements, use cases, stories)
» Non-functional (e.g., operational, deployment, systemic qualities)

2.2 Identify and Analyze Compliance Requirements

2.3 Identify and Analyze 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)

2.4 Identify and Analyze Privacy Requirements

» Data anonymization
» User consent
» Disposition (e.g., right to be forgotten)
» Data retention
» Cross borders (e.g., data residency, jurisdiction, multi-national data processing boundaries)

2.5 Develop Misuse and Abuse Cases

2.6 Develop Security Requirement Traceability Matrix (STRM)

2.7 Ensure Security Requirements Flow Down to Suppliers/Providers

Domain 3:
Secure Software Architecture and Design

3.1 Perform Threat Modeling

- Understand common threats (e.g., Advance Persistent Threat (APT), insider threat, common malware, third-party/supplier)
- Attack surface evaluation
- Threat intelligence (e.g., Identify credible relevant threats)

3.2 Define the Security Architecture

- Security control identification and prioritization
- Distributed computing (e.g., client server, peer-to-peer (P2P), message queuing)
- Service-oriented architecture (SOA) (e.g., Enterprise Service Bus (ESB), web services)
- Rich internet applications (e.g., client-side exploits or threats, remote code execution, constant connectivity)
- Pervasive/ubiquitous computing (e.g., Internet of Things (IoT), wireless, location-based, Radio-Frequency Identification (RFID), near field communication, sensor networks)
- Embedded (e.g., secure update, Field-Programmable Gate Array (FPGA) security features, microcontroller security)
- Cloud architectures (e.g., Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS))
- Mobile applications (e.g., implicit data collection privacy)
- Hardware platform concerns (e.g., side-channel mitigation, speculative execution mitigation, embedded Hardware Security Modules (HSM))
- Cognitive computing (e.g., Machine Learning (ML), Artificial Intelligence (AI))
- Control systems (e.g., industrial, medical, facility-related, automotive)

3.3 Performing Secure Interface Design

- Security management interfaces, Out-of-Band (OOB) management, log interfaces
- Upstream/downstream dependencies (e.g., key and data sharing between apps)
- Protocol design choices (e.g., Application Programming Interface (APIs), weaknesses, state, models)

3.4 Performing Architectural Risk Assessment

3.5 Model (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 Single Sign-On (SSO))
- Flow control (e.g., proxies, firewalls, protocols, queuing)
- Data loss prevention (DLP)
- Virtualization (e.g., software defined infrastructure, hypervisor, containers)
- Trusted computing (e.g., Trusted Platform Module (TPM), Trusted Computing Base (TCB))
- Database security (e.g., encryption, triggers, views, privilege management)
- Programming language environment (e.g., Common Language Runtime (CLR), Java Virtual Machine (JVM))
- Operating System (OS) controls and services
- Secure backup and restoration planning
- Secure data retention, retrieval, and destruction

3.8 Perform Security Architecture and Design Review

3.9 Define Secure Operational Architecture (e.g., deployment topology, operational interfaces)

3.10 Use Secure Architecture and Design Principles, Patterns, and Tools

Domain 4: Secure Software Implementation

4.1 Adhere to Relevant Secure Coding Practices (e.g., standards, guidelines and regulations)

- Declarative versus imperative (programmatic) security
- Concurrency (e.g., thread safety, database concurrency controls)
- Output sanitization (e.g., encoding, obfuscation)
- Error and exception handling
- Input validation
- Secure logging & auditing
- Session management
- Trusted/Untrusted Application Programming Interface (APIs), and libraries
- Type safety
- Resource management (e.g., compute, storage, network, memory management)
- Secure configuration management (e.g., parameter, default options, credentials)
- Tokenizing
- Isolation (e.g., sandboxing, virtualization, containers, Separation Kernel Protection Profiles (SKPP))
- Cryptography (e.g., payload, field level, transport, storage, agility, encryption, algorithm selection)
- Access control (e.g., trust zones, function permissions, Role Based Access Control (RBAC))
- Processor microarchitecture security extensions (e.g., Software Guard Extensions (SGX), Advanced Micro Devices (AMD) Secure Memory Encryption(SME)/Secure Encrypted Virtualization(SEV), ARM TrustZone)

4.2 Analyze Code for Security Risks

- Secure code reuse
- Vulnerability databases/lists (e.g., Open Web Application Security Project (OWASP) Top 10, Common Weakness Enumeration (CWE))
- Static Application Security Testing (SAST) (e.g., automated code coverage, linting)
- Dynamic Application Security Testing (DAST)
- Manual code review (e.g., individual, peer)
- Look for malicious code (e.g., backdoors, logic bombs, high entropy)
- Interactive Application Security Testing (IAST)

4.3 Implement Security Controls (e.g., watchdogs, File Integrity Monitoring (FIM), anti-malware)

4.4 Address Security Risks (e.g. remediation, mitigation, transfer, accept)

4.5 Securely Reuse Third-Party Code or Libraries (e.g., Software Composition Analysis (SCA))

4.6 Securely Integrate Components

- Systems-of-systems integration (e.g., trust contracts, security testing and analysis)

4.7 Apply Security During the Build Process

- Anti-tampering techniques (e.g., code signing, obfuscation)
- Compiler switches
- Address compiler warnings

Domain 5: Secure Software Testing

5.1 Develop Security Test Cases

- Attack surface validation
- Penetration tests
- Fuzzing (e.g., generated, mutated)
- Scanning (e.g., vulnerability, content, privacy)
- Simulation (e.g., simulating production environment and production data, synthetic workloads)
- Failure (e.g., fault injection, stress testing, break testing)
- Cryptographic validation (e.g., Pseudo-Random Number Generator (PRNG), entropy)
- Regression tests
- Integration tests
- Continuous (e.g., synthetic transactions)

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., International Organization for Standardization (ISO), Open Source Security Testing Methodology Manual (OSSTMM), Software Engineering Institute (SEI))
- Crowd sourcing (e.g., bug bounty)

5.3 Verify and Validate Documentation (e.g., installation and setup instructions, error messages, user guides, release notes)

5.4 Identify Undocumented Functionality

5.5 Analyze Security Implications of Test Results (e.g., impact on product management, prioritization, break build criteria)

5.6 Classify and Track Security Errors

- Bug tracking (e.g., defects, errors and vulnerabilities)
- Risk Scoring (e.g., Common Vulnerability Scoring System (CVSS))

5.7 Secure Test Data

- Generate test data (e.g., referential integrity, statistical quality, production representative)
- Reuse of production data (e.g., obfuscation, sanitization, anonymization, tokenization, data aggregation mitigation)

5.8 Perform Verification and Validation Testing

Domain 6:
Secure Software Lifecycle Management

6.1 Secure Configuration and Version Control (e.g., hardware, software, documentation, interfaces, patching)

6.2 Define Strategy and Roadmap

6.3 Manage Security Within a Software Development Methodology

- Security in adaptive methodologies (e.g., Agile methodologies)
- Security in predictive methodologies (e.g., Waterfall)

6.4 Identify Security Standards and Frameworks

6.5 Define and Develop Security Documentation

6.6 Develop Security Metrics (e.g., defects per line of code, criticality level, average remediation time, complexity)

6.7 Decommission Software

- End of life policies (e.g., credential removal, configuration removal, license cancellation, archiving)
- Data disposition (e.g., retention, destruction, dependencies)

6.8 Report Security Status (e.g., reports, dashboards, feedback loops)

6.9 Incorporate Integrated Risk Management (IRM)

- Regulations and compliance
- Legal (e.g., intellectual property, breach notification)
- Standards and guidelines (e.g., International Organization for Standardization (ISO), Payment Card Industry (PCI), National Institute of Standards and Technology (NIST), OWASP, Software Assurance Forum for Excellence in Code (SAFECode), Software Assurance Maturity Model (SAMM), Building Security In Maturity Model (BSIMM))
- Risk management (e.g., mitigate, accept, transfer, avoid)
- Terminology (e.g., threats, vulnerability, residual risk, controls, probability, impact)
- Technical risk vs. business risk

6.10 Promote Security Culture in Software Development

- Security champions
- Security education and guidance

6.11 Implement Continuous Improvement (e.g., retrospective, lessons learned)

Domain 7: Secure Software Deployment, Operations, Maintenance

7.1 Perform Operational Risk Analysis

- Deployment environment
- Personnel training (e.g., administrators vs. users)
- Safety criticality
- System integration

7.2 Release Software Securely

- Secure Continuous Integration and Continuous Delivery (CI/CD) pipeline
- Secure software tool chain
- Build artifact verification (e.g., code signing, checksums, hashes)

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)
- Security policy implementation
- Secrets injection (e.g., certificate, Open Authorization (OAUTH) tokens, Secure Shell (SSH) keys)

7.5 Perform Post-Deployment Security Testing

7.6 Obtain Security Approval to Operate (e.g., risk acceptance, sign-off at appropriate level)

7.7 Perform Information Security Continuous Monitoring (ISCM)

- Collect and analyze security observable data (e.g., logs, events, telemetry, and trace data)
- Threat intel
- Intrusion detection/response
- Secure configuration
- Regulation changes

7.8 Support Incident Response

- Root cause analysis
- Incident triage
- Forensics

7.9 Perform Patch Management (e.g. secure release, testing)

7.10 Perform Vulnerability Management (e.g., scanning, tracking, triaging)

7.11 Runtime Protection (e.g., Runtime Application Self-Protection (RASP), Web Application Firewall (WAF), Address Space Layout Randomization (ASLR))

7.12 Support Continuity of Operations

- Backup, archiving, retention
- Disaster Recovery (DR)
- Resiliency (e.g., operational redundancy, erasure code, survivability)

7.13 Integrate Service Level Objectives (SLO) and Service Level Agreements (SLA) (e.g., maintenance, performance, availability, qualified personnel)

Domain 8:
Secure Software Supply Chain

8.1 Implement Software Supply Chain Risk Management

- Identify
- Assess
- Respond
- Monitor

8.2 Analyze Security of Third-Party Software

8.3 Verify Pedigree and Provenance

- Secure transfer (e.g., interdiction mitigation)
- System sharing/interconnections
- Code repository security
- Build environment security
- Cryptographically-hashed, digitally-signed components
- Right to audit

8.4 Ensure Supplier Security Requirements in the Acquisition Process

- Audit of security policy compliance (e.g., secure software development practices)
- Vulnerability/incident notification, response, coordination, and reporting
- Maintenance and support structure (e.g., community versus commercial, licensing)
- Security track record

8.5 Support contractual requirements (e.g., Intellectual Property (IP) ownership, code escrow, liability, warranty, End-User License Agreement (EULA), Service Level Agreements (SLA))