Security Concepts

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

SECURITY OBJECTIVES

- **Confidentiality** — Protect disclosure
- **Integrity** — Prevent improper change
- **Availability** — Sustain access

01 Secure Software Concepts 12%

Security objectives and identity

- **Confidentiality** prevents unauthorized disclosure. **Integrity** protects against improper creation, alteration or deletion. **Availability** provides timely, reliable access to needed services and data.
- **Identification** claims an identity; **authentication** verifies it; **authorization** decides permitted actions. **Accountability** links actions to an identifiable actor through protected audit evidence.
- **MFA** combines distinct factors: knowledge, possession and inherence. A password plus a smart card uses two factors; two passwords use one. **Mutual authentication** verifies both endpoints.
- **Nonrepudiation** supplies evidence of origin or participation, commonly through digital signatures and protected records. Its strength depends on identity binding, private-key custody and evidence integrity.

Classify controls on separate axes

- **Administrative/procedural** controls include policy, training and incident procedures. **Technical/logical** controls include authentication, access rules and encryption. **Physical** controls include locks, barriers and guards.
- **Preventive** controls reduce the chance of an incident; **detective** controls reveal events; **corrective/recovery** controls restore or repair. A control can serve more than one function.
- **Common controls** serve multiple systems; **system-specific controls** serve one system; **hybrid controls** split responsibility between common and local implementations. Document inherited controls and local obligations.

Principles that reduce exposure

- **Least privilege** limits permissions, resources and duration. **Separation of duties** divides incompatible responsibilities; **separation of privilege** can require multiple independent conditions before access.
- **Economy of mechanism** favors small, understandable protection mechanisms. **Least common mechanism** minimizes shared mechanisms and state that could connect otherwise isolated users.

Security Concepts / Lifecycle Management

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

SECURITY OBJECTIVES

- **Confidentiality** — Protect disclosure
- **Integrity** — Prevent improper change
- **Availability** — Sustain access

- **Complete mediation** checks every relevant access. **Fail-safe defaults** deny unless explicitly allowed; a failure must not silently bypass authorization. Consider human safety as well as confidentiality when choosing failure behavior.
- **Defense in depth** combines independent layers. **Open design** avoids dependence on secret algorithms. **Psychological acceptability** makes secure behavior usable; **attack-surface reduction** removes unnecessary services, entry points and privileges.

Cryptographic services and attack models

- **Symmetric encryption** suits bulk confidentiality; **asymmetric cryptography** supports key establishment and signatures. A **hash** alone does not authenticate a sender; a keyed **MAC** authenticates data to shared-key holders.
- **Digital signatures** use a private signing key and public verification key. **Code signing** establishes publisher identity and detects post-signing changes; it does not prove that code is safe.
- **TLS** protects a connection when peer identity and certificates are validated. **HTTPS** is HTTP over TLS; **IPsec** protects IP traffic; **S/MIME** protects email. Keep keys separate from encrypted data.
- **Cryptanalysis** studies weaknesses. Ciphertext-only attacks observe ciphertext; known-plaintext attacks also know matching plaintext; chosen-plaintext or chosen-ciphertext attacks obtain selected cryptographic results. **Side channels** exploit timing, power or other implementation leakage.

02 Secure Software Lifecycle Management 11%

Governance, policy and responsibility

- **Senior management** sets direction, accountability, resources and risk tolerance. A **CISO** coordinates the security program; a steering committee aligns business priorities; the **CIO** aligns technology delivery with organizational needs.
- **Policy** states management intent; a **standard** mandates specific requirements; a **procedure** gives steps; a **guideline** recommends practices; a **baseline** establishes an approved minimum configuration.
- **Regulatory policies** address required obligations; **advisory policies** define expected behavior and consequences; **informative material** explains organizational context. Define scope, responsibilities, enforcement and exceptions.
- **ISSO** supports day-to-day system security and authorization documentation. **ISSE** engineers security and assesses change impacts. An independent **auditor/assessor** evaluates evidence; the authorized risk owner makes acceptance decisions.
- **Professional ethics** prioritize society and public trust, honest and lawful conduct, diligent and competent service, and advancement and protection of the profession. Escalate conflicts and do not misrepresent assurance evidence.

Lifecycle Management

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

RISK THROUGH THE LIFECYCLE

Identify — Assets and threats → **Assess** — Likelihood and impact → **Respond** — Owned treatment → **Monitor** — Residual risk

Risk vocabulary and analysis

- An **asset** has value; a **threat** can cause harm; a **vulnerability** is an exploitable weakness. Assess likelihood and impact in the actual operating context, including existing controls and dependencies.
- **Inherent risk** precedes treatment; **residual risk** remains after treatment; **secondary risk** arises from a response. **Detection risk** is the chance an audit fails to detect a material issue; sampling and execution errors can contribute.
- **Qualitative analysis** prioritizes risks with defined likelihood and impact scales. **Quantitative analysis** estimates numerical effects, losses or uncertainty. Both require credible data and explicit assumptions.
- **SLE = asset value x exposure factor. ALE = SLE x annualized rate of occurrence.** EF is the fraction lost per event; ARO is expected events per year and can be below or above one. ALE is an estimate, not a maximum loss.
- Compare expected risk reduction with the **total cost of controls**, including implementation, operation and side effects. Reassess after treatment; no universal arithmetic formula accurately determines all residual security risk.

Choose and track risk responses

- For threats, **avoid** removes the risky activity or condition; **mitigate** reduces likelihood or impact; **transfer** allocates specified consequences or response duties to another party; **accept** retains the exposure knowingly.
- Outsourcing does not automatically eliminate risk or accountability. Distinguish shifting contractual consequences from reducing risk by selecting a more capable supplier. For opportunities, **exploit**, **enhance** or **share** pursue beneficial outcomes.
- A **risk management plan** defines methods, roles, thresholds, review frequency and reporting. A **risk register** records particular risks, owners, assessments, treatments and status. Active acceptance prepares contingency resources and triggers.
- Risk management is **continuous**: identify, assess, respond and monitor throughout development and operations. Flowcharts expose dependencies and bottlenecks; cause-and-effect diagrams explore causes; influence diagrams show relationships among uncertain factors and decisions.

Lifecycle Management

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

RISK THROUGH THE LIFECYCLE

Identify — Assets and threats → **Assess** — Likelihood and impact → **Respond** — Owned treatment → **Monitor** — Residual risk

Embed assurance in the lifecycle

- The **SDLC** spans initiation, development/acquisition, implementation, operation/maintenance and disposal. Define security requirements early, review designs, code defensively, test controls and authorize the intended deployment.
- **Waterfall** emphasizes staged deliverables; the **V-model** relates decomposition and design to corresponding verification and validation activities. **Spiral** iterations explicitly evaluate risk; **prototyping/RAD** accelerate feedback but still need security controls.
- In **Agile/DevSecOps**, put security requirements, abuse cases, review and testing into normal iterations and delivery gates. **Verification** checks conformance to specified requirements; **validation** checks fitness for intended use.
- **Traceability** links business needs, requirements, design, implementation and tests in both directions. Set measurable release criteria, remediation ownership and exception expiration; a deadline alone is not assurance evidence.

Improve processes and measure assurance

- **Software assurance** is justified confidence that software performs as intended and resists vulnerabilities and malicious behavior. Combine engineering evidence, testing and operational feedback rather than claiming absolute defect freedom.
- **PDCA**: plan objectives and controls; do implement and operate; check measure and review; act correct and improve. **Benchmarking** compares performance or practices; **Six Sigma** targets defects and process variation.
- **CMM/CMMI** address process capability and improvement. **SSE-CMM** applies capability assessment to security engineering, including quality, configuration, project risk, skills and supplier coordination; capability evidence is distinct from product security certification.
- In historical **FITSAF** terminology, progression is documented policy, documented procedures, implementation, testing/review and integration. **Root cause analysis** examines underlying systemic causes and verifies corrective actions, rather than stopping at symptoms.

Lifecycle Management / Security Requirements

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

REQUIREMENTS TRACEABILITY

Business need → **Security requirement** → **Design and implementation** → **Acceptance evidence**

Assessment and authorization frameworks

- **Certification/assessment** evaluates control evidence. **Accreditation/authorization** is a management decision to permit operation and accept identified residual risk. The **authorizing official (AO)**, historically a **DAA**, owns that decision.
- Historical **DITSCAP/NIACAP** sequence: definition establishes mission, environment and requirements; verification checks evolving implementation; validation evaluates the integrated system; post-accreditation maintains security during operation. The **SSAA** records the system security agreement.
- Historical **DIACAP** activities cover initiation/planning, implementation/validation, determination/accreditation, maintaining authorization and decommissioning. **ATO** permits operation; interim operating/testing approvals are scoped and conditional; denial prohibits the proposed operation.
- **NIST SP 800-37 RMF** links prepare, categorize, select, implement, assess, authorize and monitor. Ongoing monitoring tracks control effectiveness, configuration changes and risk; significant changes can require reassessment or a new authorization decision.

03 Secure Software Requirements 13%

Turn business needs into testable controls

- **Functional requirements** specify behavior, such as authentication or transaction approval. **Nonfunctional requirements** specify qualities and constraints, such as response time, availability, confidentiality and recovery. Make each measurable and testable.
- **CONOPS** describes intended operation from stakeholder viewpoints. Operational scenarios, misuse/abuse cases and threat analysis expose security needs that ordinary happy-path use cases can miss.
- Define **acceptance criteria**, ownership, interfaces, environmental assumptions and negative cases. Maintain a requirements traceability matrix as requirements change; controls and tests should map back to a documented need.

Data ownership, classification and handling

- The **data owner** sets classification, handling requirements and access criteria, approves appropriate access and reviews them as needs change. The **custodian** implements protection, access administration, backup and restoration on the owner's behalf.
- **Users** follow handling rules and exercise due care; **auditors** independently assess implementation. Delegating operational work to a custodian does not remove the owner's accountability.

Security Requirements / Architecture & Design

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

REQUIREMENTS TRACEABILITY

- **Business need** → **Security requirement** → **Design and implementation** → **Acceptance evidence**

- Classification follows **sensitivity, criticality and impact** across creation, use, storage, transfer, retention and disposal. Commercial labels vary by organization. Government labels such as Confidential, Secret and Top Secret belong to a separate classification scheme.
- **FIPS 199** evaluates potential impact separately for confidentiality, integrity and availability: **low** means limited, **moderate** serious and **high** severe or catastrophic adverse effect. Sensitivity and service criticality can differ.

Compliance, privacy and reference families

- Capture applicable **privacy**, contractual, regulatory and industry obligations as requirements. Define authorized purposes, data minimization, disclosure, retention and monitoring notice; encryption alone does not make data use appropriate.
- **NIST** develops standards and guidance; **OMB** provides federal management and budget oversight; **CNSS** addresses national security systems policy. **FISMA** anchors federal information-security program requirements. Assess applicability to the system and data.
- **SP 800-53** catalogs controls; **800-53A** provides assessment procedures; **800-60** maps information types to categorization; **800-59** addresses identifying national security systems. Historical **800-26/800-42** concern self-assessment/network testing.
- **ISO/IEC 27001** specifies ISMS requirements; **27002** gives control guidance; **27003** implementation guidance; **27004** measurement; **27005** risk management; **27006** requirements for ISMS certification bodies. Distinguish a management-system assessment from an application test.

04 Secure Software Architecture and Design 15%

Threat modeling and design review

- Map **assets, actors, data flows, entry/exit points and trust boundaries**. Identify how an adversary could violate security goals, rank the risks, select mitigations and derive tests. Update the model when architecture or exposure changes.
- Review **anonymous entry points**, internet-facing listeners, default-enabled functions, elevated services and sensitive-data handlers first. Complex or frequently changed code and memory-unsafe components deserve added scrutiny; age alone does not determine security.
- **SABSA** derives security architecture from business requirements and risk. Architecture reviews examine trust assumptions, control placement, dependencies, failure modes and operational interfaces before implementation locks in avoidable weaknesses.

Access control and formal models

- **DAC** lets authorized owners delegate access; **MAC** enforces centrally defined labels and clearances; **RBAC** assigns permissions through job roles. Apply least privilege in any model and review role membership and exceptions.
- An **access matrix** maps subjects to objects and permitted operations. An **ACL** organizes permissions by object; a capability organizes authority held by a subject. **Take-grant** uses a directed graph to reason about rights transfer.

Architecture & Design

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

THREAT-INFORMED DESIGN

Assets and data flows → **Trust boundaries** → **Threats and risks** → **Controls and tests**

- **Bell-LaPadula** protects confidentiality: no read up, no write down. The strict **Biba** integrity model reverses the direction: no read down, no write up. Apply the assumptions of the specific model.
- **Clark-Wilson** protects integrity through well-formed transactions, constrained data, certified transformation procedures and separation of duties. **Brewer-Nash/Chinese Wall** restricts access dynamically to prevent conflicts of interest.
- **TCB** is the hardware, firmware and software responsible for enforcing security policy. A **reference monitor** should mediate every access, resist tampering and be small enough to analyze. Historical **TCSEC** classes progress from minimal/discretionary to mandatory and verified protection.

Interfaces, composition and security patterns

- An **API** is a defined interaction contract, not a programming language. Encapsulation hides implementation details, controls permitted interactions and allows implementation changes while preserving the contract. Validate inputs and enforce authorization at each trust boundary.
- **SOA** exposes loosely coupled services with explicit contracts. **SOMF/SOMA** support service modeling and analysis; preserve business traceability, service relationships, message paths and deployment dependencies. **Pipes and filters** pass outputs through successive transformations.
- **Cascading** connects one component's output to another's input; **feedback** returns outputs into earlier inputs. Validate end-to-end security properties: secure components do not automatically form a secure composite system.
- **Partitioned applications** separate responsibilities; a **server sandbox** limits compromise impact; secure assertions check invariants; an **authenticated session** maintains identity across requests. Hidden implementation or obfuscation is only a supplementary obstacle.

Cloud, virtualization and resilience

- **Private cloud** serves one organization; **community cloud** serves organizations with shared concerns; **public cloud** is available for broad use; **hybrid cloud** connects distinct cloud infrastructures. Ownership and physical location are separate questions.
- A **hypervisor/VMM** runs guest operating systems on shared hardware. Virtualization can isolate workloads, enable snapshots and support inspection, but introduces hypervisor compromise, VM escape, shared-resource channels and virtual-network configuration risks.
- Protect **management planes, host systems, images and secrets** as well as guests. Snapshots can contain sensitive data and stale vulnerabilities; attestation depends on a valid trust chain. A snapshot is not automatically an independent backup.
- **Redundancy**, replication, clustering and geographical diversity improve resilience to selected failures. Independent implementations or voting can reduce some common failure risks, but redundancy alone does not prevent tampering or a shared compromise.

Secure Implementation

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

DEFENSIVE IMPLEMENTATION

- **Validate input** → **Enforce permission** → **Process safely** → **Protect output**

05 Secure Software Implementation 14%

Defensive coding and resource safety

- Use **small, cohesive functions**, explicit contracts, consistent style and minimal hidden assumptions. Review code during development and after security-relevant changes; prioritize unsafe interfaces, high privilege and sensitive operations.
- Treat **parameters, query strings, cookies, headers, files and upstream services** as untrusted. Validate type, length, range, syntax and business meaning after controlled canonicalization. Client-side checks do not replace server enforcement.
- Prevent **buffer overflows**, use-after-free, format-string flaws and leaks with safe APIs, checked bounds, clear ownership and runtime analysis. **Race conditions/TOCTOU** require atomic operations, synchronization or appropriate transactional controls.
- **Managed runtimes** provide services such as type checks and memory management; native/unmanaged code exposes more direct resource responsibilities. Both can contain authorization, logic, injection and dependency flaws.

Web attacks and targeted defenses

- **SQL injection** occurs when input becomes query syntax. Use typed bound parameters, safe data-access libraries and minimal database permissions. Stored procedures are safe only when they avoid unsafe dynamic query construction.
- **XSS** executes attacker-controlled content in a victim's browser. Use context-specific output encoding, safe DOM APIs and appropriate sanitization for permitted markup; a content security policy adds defense in depth.
- **CSRF** abuses a browser's ambient credentials to trigger unintended actions. Use anti-CSRF tokens, suitable SameSite cookies and origin checks; reauthenticate sensitive operations. XSS can undermine CSRF defenses.
- **File inclusion/execution and path traversal** arise when input controls file paths or executable content. Use fixed mappings, restricted destinations, safe upload handling and execution isolation; do not concatenate input into file or command operations.

Secure Implementation

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

DEFENSIVE IMPLEMENTATION

- **Validate input** → **Enforce permission** → **Process safely** → **Protect output**

Authentication, sessions and enforcement

- **Declarative security** expresses constraints in metadata or configuration for container/runtime enforcement. **Programmatic security** enforces decisions in code. Use consistent policy and test both paths for bypasses.
- For servlet security, a **web resource collection** identifies URL patterns and HTTP methods; an **authorization constraint** specifies permitted roles; a **user data constraint** specifies transport protection. Configure coverage for all exposed methods and paths.
- Servlet **getRemoteUser()** returns the authenticated name, **getUserPrincipal()** the Principal, and **isUserInRole()** a role check. Request data remains untrusted even after authentication; identity is not proof of permission to every object.
- Use unpredictable **session identifiers**, rotate them after authentication or privilege changes, protect cookies and invalidate sessions on logout/expiry. Rate limiting and careful lockout policies deter guessing without making account denial of service easy.

Exception safety and security tooling

- Handle **exceptions** predictably, release resources and preserve invariants. Return useful generic errors to callers while recording protected diagnostics; avoid stack traces, credentials or internal paths in public responses.
- **Basic exception safety** preserves invariants and prevents resource leaks; the **strong guarantee** leaves state unchanged on failure; **no-throw** operations do not propagate exceptions. A no-throw guarantee does not mean every requested operation succeeds.
- Use compiler warnings, security checks, type analysis, sanitizers and tested **security libraries**. **ESAPI** is an example of a reusable security-control API; library reuse still requires suitability review, correct configuration and maintenance.
- **Obfuscation** can change layout, data representation or control flow to complicate analysis. It does not replace sound cryptography or authorization. Keep build environments, signing keys and artifact promotion paths protected from unauthorized changes.

Security Testing

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

COMPLEMENTARY TESTING

- **Review** — Source and design
- **Execute** — Behavior and faults
- **Attack** — Authorized exploitation
- **Regress** — Change effects

06 Secure Software Testing 14%

Test levels, purposes and evidence

- **Unit tests** isolate a small component; **integration tests** exercise interfaces and interactions; **system tests** evaluate the complete system; **acceptance tests** check agreed business and security criteria. A **driver** calls a tested unit; a **stub** substitutes for a dependency.
- **Regression tests** check for unintended effects of changes. **Retesting/confirmation** verifies a particular defect fix. **Functional tests** check specified behavior; performance, reliability, scalability and recovery tests assess different quality attributes.
- **Black-box/unknown-environment** testing uses external behavior with little internal information; **white-box/known-environment** testing uses internal structure; **gray-box** testing uses partial knowledge. These perspectives can apply at different test levels.
- Trace tests to **requirements and threats**. Include invalid inputs, boundary values, authorization failures and abuse cases. Coverage metrics reveal exercised structures but do not prove absence of vulnerabilities.

Security review, analysis and penetration tests

- **Code review** inspects source for security defects; peer review can also examine requirements or design. An **audit** assesses conformance and evidence. **ST&E** checks control adequacy, consistency between documentation and implementation, and exploitable flaws.
- **SAST** analyzes code without executing the target; **DAST** probes a running application; **IAST** observes execution with instrumentation. **Fuzzing** supplies generated or mutated inputs; fault injection tests corrupted files, resource exhaustion and network failures.
- **Vulnerability scanning** identifies suspected weaknesses. **Penetration testing** attempts authorized exploitation to demonstrate feasibility and business impact. Set rules of engagement, targets, safety limits, data handling, reporting and cleanup; even full-knowledge testing has an agreed scope.
- **Network discovery** finds active hosts; **port scans** identify exposed listeners; service and OS fingerprinting characterize targets. **Active fingerprinting** sends probes; **passive fingerprinting**, such as p0f, observes existing traffic. An open port is not itself proof of exploitability.

Security Testing / Deployment & Operations

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

COMPLEMENTARY TESTING

- **Review** — Source and design
- **Execute** — Behavior and faults
- **Attack** — Authorized exploitation
- **Regress** — Change effects

Recognize attacker behavior

- **Reconnaissance** gathers information; disclosure attacks expose data; sabotage/DoS attacks disrupt service; backdoors enable later access. **Social engineering** manipulates people; dumpster diving examines discarded material; shoulder surfing observes secrets; keyloggers record input.
- **War/demon dialing** searches telephone lines for modems. **Sniffing/eavesdropping** observes communications. **Replay** reuses captured valid messages; **on-path attacks** intercept or modify exchanges; **zero-day exploitation** targets a vulnerability without an available established fix or prior defender awareness.
- **DDoS** uses multiple sources to exhaust resources. **SYN floods** pressure connection state; ping floods send excessive ICMP traffic; **Smurf** uses spoofed-source broadcast reflection; **Teardrop** targets fragment-reassembly weaknesses. Match mitigations to the actual bottleneck and protocol behavior.
- **Dictionary attacks** try likely passwords; brute force explores candidates systematically. Reduce exposure through MFA, throttling, secure password storage and monitoring. Training and verification procedures reduce social engineering risk but cannot guarantee its elimination.

07 Secure Software Deployment, Operations, Maintenance 11%

Configuration and controlled change

- **Configuration identification** defines items, attributes, relationships and baselines. A **configuration item (CI)** is a controlled component or artifact; a variant differs from a related item while retaining core functionality.
- **Configuration change control** evaluates impact, approves, tests, implements and records changes to baselined items. **Status accounting** reports current configurations and change state. Include requirements, code, libraries, build settings, infrastructure and operational documentation.
- **Functional configuration audits** verify required functional/performance characteristics. **Physical configuration audits** compare the as-built product and its identifying artifacts with approved design documentation. Both support consistency and traceability.
- Plan **rollback**, communicate changes, segregate approval from implementation and verify the deployed result. Emergency changes still need recorded justification and prompt review. Reassess threats and residual risk after material changes.

Deployment & Operations

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

CONTROLLED OPERATIONS

Baseline and approve → **Deploy and verify** → **Monitor and respond** → **Recover and improve**

Harden, release and maintain

- Deploy an approved **security baseline** with minimal privileges, disabled unnecessary services, protected administration and appropriate network segmentation. Installation must not silently weaken the environment; verify settings and controls after deployment.
- Verify **artifact origin and integrity**, protect secrets, isolate delivery environments and require release approval at the proper level. Firmware **POST** initializes and checks hardware; **secure boot** verifies authorized boot components. They solve different problems.
- Inventory systems and components, prioritize **patches** by exposure and impact, test compatibility, stage rollout and verify installation. Scan for missing updates, configuration drift and unauthorized software; obsolete scanners do not establish assurance for modern systems.
- For **decommissioning**, revoke credentials and certificates, remove integrations and configurations, preserve required records and sanitize storage. **Data remanence** is recoverable residual data; ordinary deletion or formatting may be insufficient. Choose media-appropriate clearing, purging or destruction.

Logging, detection and response

- An **audit trail** is a chronological record supporting reconstruction. Capture actor, action, target, outcome and synchronized time for relevant access, authentication, changes and administrative events. Restrict alteration, protect sensitive fields and enforce retention.
- **Logging** records events; monitoring reviews them; analysis identifies patterns; alert rules trigger response. A **SIEM** correlates sources with asset context, prioritizes events and supports incident workflows. Rate-limit noisy sources and buffer logs to reduce collection denial of service.
- **Signature detection** matches known patterns; **anomaly detection** compares behavior with a baseline. **IDS** detects and alerts; **IPS** can block. **NIDS/NIPS** inspect network traffic with limited encrypted-content visibility. **HIDS/HIPS** observe the host, potentially after decryption, but need correlation for distributed events.
- **Insertion/evasion** exploits differences between what a sensor and endpoint accept or reconstruct. Normalize protocols and align sensor behavior with protected hosts. **File integrity monitoring** detects baseline changes; **DLP** can alert, quarantine or block policy-violating data movement.

Deployment & Operations / Software Supply Chain

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

CONTROLLED OPERATIONS

- **Baseline and approve** → **Deploy and verify** → **Monitor and respond** → **Recover and improve**

Continuity, recovery and exercises

- **BCP** sustains critical business processes, people, facilities and communications; **DRP** restores IT services; **COOP** sustains essential operations. Contingency plans define scenarios, triggers, responsibilities and recovery actions.
- A **BIA** identifies critical processes, dependencies and disruption impacts; a risk assessment evaluates threats and likelihood. **RTO** targets time to restore service; **RPO** limits data loss measured backward in time. Actual recovery performance must be measured against both.
- **Hot sites** have ready capability; **warm sites** need additional preparation; **cold sites** mainly provide facilities. Cost, capacity, data currency, dependencies and tested activation time determine fitness for recovery objectives.
- **Checklist reviews** examine individual responsibilities; **tabletop/walkthroughs** discuss coordinated action; simulations exercise scenarios; **parallel tests** validate alternate processing while primary operations continue; **full-interruption tests** actually move operations and carry greater disruption risk.
- Test **restoration and failover**, not only backup creation. Protect recovery copies from the same compromise as production, document return-to-service decisions and update plans after exercises, incidents and changes.

08 Secure Software Supply Chain 10%

Acquisition and supplier assurance

- Software assurance acquisition moves through **planning**, **contracting**, **monitoring/acceptance** and **follow-on**. Planning establishes needs, risks, acquisition strategy, security requirements and evaluation criteria before choosing a supplier.
- Contracts should define security deliverables, secure development expectations, assessment rights, vulnerability notification, remediation, support periods and acceptance evidence. Monitor obligations through delivery and maintenance; reassess when suppliers or components change.
- Evaluate **COTS**, open-source and custom components for provenance, security track record, support, licensing and integration risk. A component inventory or **SBOM** supports vulnerability tracking; authenticate transfers and protect repositories and build environments.
- An **SLA** records agreed service targets, hours, maintenance windows and responsibilities. **Customer-based**, **service-based** and **multilevel** structures organize coverage. Internal **OLAs** and external underpinning contracts support delivery of those targets.

Software Supply Chain

Certified Secure Software Lifecycle Professional

KNOWLEDGE CHEATSHEET

ACQUISITION ASSURANCE

- **Plan requirements** → **Contract obligations** → **Monitor and accept** → **Follow-on support**

Intellectual property and ownership

- **Copyright** protects original expression such as source code; **patents** concern qualifying inventions; **trademarks** distinguish the source of goods/services; **trade secrets** protect valuable information kept confidential. Determine ownership and permitted use in contracts and licenses.
- **Cybersquatting** concerns bad-faith registration or use of domains associated with another's mark. Domain availability does not establish trademark rights. Use legal review for jurisdiction-specific disputes and licensing decisions.

Digital content and tamper protection

- **DRM** limits authorized uses of digital content. **Compliance rules** define permitted behavior and license rights; **robustness rules** address resistance to circumvention. Protect content/device keys, provisioning and revocation; device fingerprinting has privacy and reliability limits.
- **Tamper evidence** reveals interference; **tamper resistance** makes it harder; neither guarantees tamper proofing. Watermarking embeds identifying information, steganography hides information, and encryption protects confidentiality. They provide different properties.