SCOPE Framework: Application Security Control Domain

Table of Contents

Originating Component	SCOPE Framework Governance Committee
Releasability	Cleared for public distribution. Available in the SCOPE Framework Hub at [https://timtiptonjr.com/scope-hub].

Purpose: The Application Security domain establishes the principles, controls, and practices necessary to protect applications throughout their lifecycle—from development and deployment to use and decommissioning. This domain ensures that software is securely engineered, APIs are hardened against misuse, and vulnerabilities are continuously identified and remediated. By embedding security directly into application workflows, the organization reduces its attack surface, safeguards critical information assets, and meets regulatory and contractual obligations with confidence and precision.

Section 1. Secure Software Development Lifecycle (SSD)

The Secure Software Development Lifecycle (SSD) control family defines the minimum security requirements and governance activities necessary to integrate security into each phase of the software development lifecycle—from requirements gathering through design, implementation, testing, deployment, and maintenance. These controls ensure that security is embedded from the outset and not retrofitted, aligning software assurance efforts with organizational risk tolerance, compliance obligations, and operational resiliency goals.

1.1 SSD-01: Security Requirements Definition

Organizations shall define and document security requirements as part of functional and non-functional requirements during the planning and requirements-gathering phase of all software development efforts.

1.2 SSD-02: Secure Architecture Design

Security considerations shall be integrated into software architecture design, ensuring threat modeling, secure design patterns, and mitigation strategies are embedded before development begins.

1.3 SSD-03: Threat Modeling

Threat modeling shall be performed on all applications and systems during the design phase, updated as architecture or code changes, and reviewed regularly to identify emerging risks.

1.4 SSD-04: Secure Coding Standards

Developers shall follow secure coding standards aligned with industry-recognized sources (e.g., OWASP, SEI CERT), and these standards shall be enforced through development workflows.

1.5 SSD-05: Security Training for Developers

All developers shall receive role-specific secure coding and secure development training annually, with training tailored to the languages, frameworks, and platforms used.

1.6 SSD-06: Code Review with Security Focus

All application code shall undergo peer review with specific attention to security flaws, including but not limited to authentication, authorization, input validation, and data handling.

1.7 SSD-07: Static Application Security Testing (SAST)

SAST tools shall be integrated into the CI/CD pipeline to automatically scan source code or binaries for known vulnerabilities and insecure coding patterns before code is merged or deployed.

1.8 SSD-08: Dynamic Application Security Testing (DAST)

DAST shall be conducted on all applications, especially those exposed to public or partner networks, to assess runtime behavior and uncover security vulnerabilities in a deployed state.

1.9 SSD-09: Software Composition Analysis (SCA)

All third-party components, libraries, and dependencies shall be continuously inventoried and scanned for vulnerabilities using SCA tools throughout the development lifecycle.

1.10 SSD-10: Secure Configuration Management

Configuration files, secrets, and infrastructure-as-code artifacts shall be stored securely, version-controlled, and regularly reviewed to ensure alignment with security baselines.

1.11 SSD-11: Least Privilege in Development Environments

Development, testing, and CI/CD environments shall enforce the principle of least privilege, with separate access controls from production and no shared credentials.

1.12 SSD-12: Environment Segregation

Development, test, staging, and production environments shall be logically and physically separated, with controls in place to prevent cross-contamination of data or code.

1.13 SSD-13: Use of Approved Tools and Libraries

Only vetted and approved development tools, libraries, and frameworks shall be used in the software development process, and any exceptions must be documented and risk-assessed.

1.14 SSD-14: CI/CD Pipeline Security Controls

Security controls, including access controls, artifact scanning, and audit logging, shall be implemented across all CI/CD pipelines to prevent unauthorized code changes or artifact promotion.

1.15 SSD-15: Secure Build and Artifact Integrity

Build processes shall ensure reproducibility and integrity verification (e.g., checksums, signing) of artifacts before deployment or release to ensure trust in compiled code.

1.16 SSD-16: Remediation of Identified Vulnerabilities

All vulnerabilities identified during code analysis or testing phases shall be tracked, prioritized, and remediated according to organizational risk tolerance and defined SLAs.

1.17 SSD-17: Pre-Deployment Security Gate Reviews

Security gate reviews shall be conducted prior to any major release or deployment to validate completion of secure development activities and verify risk posture.

1.18 SSD-18: Application Security Metrics & KPIs

Organizations shall define and track key application security metrics (e.g., vulnerability density, time-to-remediate) to continuously measure the effectiveness of secure SDLC efforts.

1.19 SSD-19: Secure Decommissioning of Code and Applications

Procedures shall exist for securely retiring code and decommissioning applications, including proper data sanitization, archive integrity, and revocation of access.

1.20 SSD-20: Periodic SSDLC Process Review

The secure SDLC process shall be reviewed and updated annually or when significant changes occur in the technology stack, development practices, or threat landscape.

1.21 SSD-21: Inclusion of Security in User Story Acceptance Criteria

Security-related acceptance criteria shall be defined and incorporated into user stories and backlog items to ensure security considerations are part of the agile development lifecycle.

1.22 SSD-22: Integration of Security into Agile and DevOps Workflows

Security tasks, testing, and validation shall be embedded into Agile sprints and DevOps workflows, with responsibilities clearly defined across development, security, and operations teams.

Section 2. Software Bill of Materials (SBM)

The Software Bill of Materials (SBM) control family establishes the standards and practices for identifying, tracking, managing, and verifying all components—open-source, proprietary, commercial, and third-party—used in software systems. These controls promote supply chain transparency, vulnerability awareness, and risk-informed decision-making, ensuring organizations can respond effectively to security events tied to the software supply chain.

2.1 SBM-01: SBOM Generation Requirements

Organizations shall require that an SBOM be generated for every internally developed or externally acquired software product prior to production deployment, inclusive of direct and transitive dependencies.

2.2 SBM-02: SBOM Format Standardization

All SBOMs shall adhere to a standardized format such as SPDX, CycloneDX, or SWID to ensure interoperability across tooling and partner ecosystems.

2.3 SBM-03: Component Inventory Accuracy

The SBOM shall provide a complete and accurate inventory of all software components, including version numbers, licenses, publishers, and relationships (e.g., dependency chains).

2.4 SBM-04: Automated SBOM Generation

Organizations shall implement tooling to automate the generation and maintenance of SBOMs during each build or major software version update.

2.5 SBM-05: SBOM Validation and Verification

SBOMs shall undergo validation to verify completeness and integrity, including checks for tampering, falsified component data, or missing dependencies.

2.6 SBM-06: Third-Party Vendor SBOM Provisioning

Vendors and suppliers shall be contractually required to provide SBOMs for all delivered software, and SBOMs shall be reviewed as part of the acquisition or risk onboarding process.

2.7 SBM-07: SBOM Central Repository

Organizations shall maintain a centralized and secure repository for all SBOMs, indexed by software product, version, and deployment environment, with appropriate access controls.

2.8 SBM-08: SBOM Update Frequency

SBOMs shall be updated in alignment with any software release, patch, dependency change, or component replacement that alters the software composition.

2.9 SBM-09: SBOM Lifecycle Retention

SBOMs shall be retained for the entire lifecycle of the associated software and for a minimum of 12 months after decommissioning to support forensic and supply chain risk investigations.

2.10 SBM-10: Dependency Trust Classification

Each component listed in the SBOM shall be assessed and labeled with a trust level based on source reputation, maintenance history, known vulnerabilities, and license compliance.

2.11 SBM-11: SBOM Risk Correlation

Organizations shall integrate SBOM data into vulnerability and risk management platforms to enable automated identification of impacted components when new CVEs are published.

2.12 SBM-12: Binary/Artifact Verification Against SBOM

Where feasible, compiled binaries and application artifacts shall be scanned and compared against the SBOM to ensure alignment with declared components and configurations.

2.13 SBM-13: SBOM Access Controls

Access to SBOMs shall be restricted based on need-to-know, with access logging and periodic review to prevent unauthorized disclosure of sensitive supply chain data.

2.14 SBM-14: Cryptographic Signing of SBOMs

All SBOMs shall be cryptographically signed to ensure authenticity and enable consumers to validate source origin and integrity.

2.15 SBM-15: Legacy Software SBOM Reconstitution

For legacy applications lacking SBOMs, organizations shall use reverse engineering, SCA tools, or vendor outreach to reconstruct a best-effort SBOM to support ongoing risk posture assessments.

2.16 SBM-16: License Compliance Mapping

SBOMs shall include license information for all components, with automated tracking and alerts for potential conflicts, violations, or noncompliant software usage.

2.17 SBM-17: Component Provenance Documentation

Where applicable, SBOMs shall include metadata on component provenance, including origin source, acquisition method, and historical version trajectory.

2.18 SBM-18: Vulnerability Disclosure Mapping to SBOM

Organizations shall map disclosed vulnerabilities and advisories (e.g., CISA KEVs, NVD CVEs) to affected components in the SBOM and initiate appropriate response actions.

2.19 SBM-19: Supply Chain Dependency Visualization

Visualization tools shall be used to render software supply chains graphically, highlighting risk clusters and high-dependency nodes to support analysis and decision-making.

2.20 SBM-20: SBOM Integration into DevSecOps Toolchains

The generation, validation, storage, and risk correlation of SBOMs shall be fully integrated into CI/CD and DevSecOps pipelines to support security automation and early detection of supply chain risks.

2.21 SBM-21: Regulatory Alignment of SBOM Practices

SBOM practices shall be reviewed against relevant regulatory and industry requirements (e.g., Executive Order 14028, NTIA SBOM Minimum Elements) to ensure compliance and readiness.

2.22 SBM-22: Consumer-Side SBOM Utilization Policy

A formal policy shall be established to govern how SBOMs are consumed internally—defining responsibilities for analysis, remediation prioritization, and integration with asset inventories and vulnerability workflows.

Section 3. Encryption & Key Management (EKM)

The Encryption & Key Management (EKM) control family establishes the foundational and operational requirements for the use of cryptography to protect sensitive data in transit, at rest, and in use. This includes the governance, generation, distribution, rotation, storage, and destruction of cryptographic keys. These controls aim to ensure confidentiality, integrity, and authenticity of data while maintaining compliance with regulatory, contractual, and operational obligations.

3.1 EKM-01: Cryptographic Standards Enforcement

All encryption mechanisms shall utilize cryptographic algorithms, key lengths, and protocols that meet or exceed current industry and regulatory standards (e.g., FIPS 140-3, NIST SP 800-131A).

3.2 EKM-02: Data-at-Rest Encryption Enforcement

Sensitive and regulated data at rest, including on databases, file systems, virtual storage, and backups, shall be encrypted using approved encryption methods.

3.3 EKM-03: Data-in-Transit Encryption Enforcement

All sensitive data transmitted across internal or external networks shall be encrypted using secure transport protocols (e.g., TLS 1.2 or higher, IPsec).

3.4 EKM-04: Full-Disk and Volume Encryption

All portable and workstation endpoints, as well as servers handling sensitive data, shall implement full-disk or volume-level encryption where technically feasible.

3.5 EKM-05: Key Management System (KMS) Governance

Organizations shall deploy and maintain a centralized Key Management System (KMS) to govern the lifecycle and usage of cryptographic keys.

3.6 EKM-06: Key Generation Entropy Requirements

All cryptographic keys shall be generated using a source of sufficient entropy and comply with approved standards for randomness and unpredictability.

3.7 EKM-07: Key Rotation Policy

A documented key rotation schedule shall be enforced based on key type, usage, and data sensitivity, with automatic key rollover implemented where feasible.

3.8 EKM-08: Key Expiration and Deactivation

Cryptographic keys shall have defined expiration dates, and expired or unused keys shall be promptly deactivated and removed from active use.

3.9 EKM-09: Segregation of Duties in Key Operations

Key generation, distribution, storage, and destruction responsibilities shall be separated across roles to reduce the risk of insider threats and compromise.

3.10 EKM-10: Encryption Key Access Controls

Access to cryptographic keys shall be strictly controlled through role-based access control, multifactor authentication, and system-level segregation.

3.11 EKM-11: Key Usage Restrictions

Each cryptographic key shall have defined usage parameters (e.g., data encryption only, signing only), and enforcement shall be implemented through technical controls.

3.12 EKM-12: Key Wrapping and Protection Mechanisms

Keys stored within software or hardware shall be encrypted ("wrapped") using a higher-level key, with protection aligned to data classification and threat level.

3.13 EKM-13: Secure Key Distribution Methods

Cryptographic keys shall be distributed using secure channels with mutual authentication and encryption to prevent unauthorized interception or manipulation.

3.14 EKM-14: HSM Utilization for High-Value Keys

Hardware Security Modules (HSMs) or equivalent tamper-resistant environments shall be used to generate, store, and manage cryptographic keys for high-sensitivity assets.

3.15 EKM-15: Certificate and PKI Management

Digital certificates and supporting Public Key Infrastructure (PKI) components shall be centrally managed, monitored, and maintained with defined issuance, renewal, and revocation procedures.

3.16 EKM-16: Key Compromise Response Procedures

In the event of suspected or confirmed key compromise, a defined response plan shall be initiated, including revocation, re-issuance, and impact analysis procedures.

3.17 EKM-17: Key Archival and Recovery

A secure, auditable mechanism shall exist for the archival and recovery of encryption keys used for long-term data access, subject to multi-person approval controls.

3.18 EKM-18: Logging and Auditing of Key Activities

All key management operations, including access, generation, usage, and destruction, shall be logged and monitored for unauthorized or anomalous activity.

3.19 EKM-19: Cryptographic Module Validation

All encryption solutions and key management components shall utilize validated cryptographic modules (e.g., FIPS 140-2/3 validated) appropriate to the data's sensitivity level.

3.20 EKM-20: Data-in-Use Protection Techniques

Where feasible and appropriate, techniques such as homomorphic encryption, secure enclaves, or trusted execution environments (TEEs) shall be evaluated and implemented to protect sensitive data while in use.

3.21 EKM-21: Key Destruction Procedures

Upon key expiration, revocation, or decommissioning, keys shall be destroyed using NIST SP 800-88 or equivalent methods to ensure they cannot be recovered or reused.

3.22 EKM-22: Cryptographic Algorithm Lifecycle Review

Cryptographic algorithms and key lengths in use shall be periodically reviewed in alignment with evolving standards and guidance, and deprecated algorithms shall be phased out.

3.23 EKM-23: Cloud Key Management Strategy

For cloud-hosted workloads, organizations shall define and enforce policies around Bring Your Own Key (BYOK), Hold Your Own Key (HYOK), or cloud-native key usage, ensuring alignment with security and compliance obligations.

Section 4. Vulnerability & Patch Management (VPM)

The Vulnerability & Patch Management (VPM) control family defines the policies, processes, and tools required to identify, assess, prioritize, and remediate vulnerabilities across systems, applications, and infrastructure components. These controls ensure that known weaknesses are addressed in a timely and risk-informed manner to reduce the attack surface, improve operational resilience, and align with regulatory or contractual obligations for system integrity and security hygiene.

4.1 VPM-01: Vulnerability Scanning Policy and Scope

Organizations shall establish a documented policy defining the frequency, scope, and methodology for vulnerability scanning across all asset types and environments.

4.2 VPM-02: Authenticated Scanning Implementation

Where technically feasible, authenticated vulnerability scans shall be conducted on systems to improve accuracy, reduce false positives, and reveal misconfigurations not visible externally.

4.3 VPM-03: External Attack Surface Scanning

All publicly accessible systems, applications, and IP ranges shall undergo routine external scanning to identify internet-facing vulnerabilities and exposure points.

4.4 VPM-04: Internal Network Vulnerability Scanning

Internal assets shall be scanned on a scheduled basis to detect vulnerabilities that may be exploited by insider threats or lateral movement tactics.

4.5 VPM-05: Asset Risk Categorization for Prioritization

Assets shall be categorized by criticality, sensitivity, and exposure to determine prioritization tiers for vulnerability remediation and patch application.

4.6 VPM-06: SLA-Based Remediation Timeframes

Remediation timelines shall be defined based on vulnerability severity and asset classification, with SLAs that account for exploitability and business impact.

4.7 VPM-07: Patch Management Governance

A formal patch management process shall govern how updates and security patches are evaluated, tested, deployed, and verified across operating systems, applications, and third-party tools.

4.8 VPM-08: Vulnerability Exception Handling

A defined process shall exist for approving and documenting exceptions to vulnerability remediation, including compensating controls and periodic re-evaluation.

4.9 VPM-09: Third-Party Software Patching

Organizations shall maintain an inventory of third-party applications and libraries and include them in patching and vulnerability management programs.

4.10 VPM-10: Operating System Patch Baselines

Standardized patch baselines shall be established for all supported operating systems, and deviations shall be documented, tracked, and addressed through governance.

4.11 VPM-11: Patch Testing and Impact Analysis

Patches shall be tested in a controlled environment prior to deployment to production systems to assess for potential impact on system stability and operations.

4.12 VPM-12: Vulnerability Intelligence Integration

Threat intelligence feeds and vulnerability databases (e.g., NVD, CISA KEV) shall be integrated into vulnerability management platforms to support risk-based prioritization.

4.13 VPM-13: Out-of-Band Patch Handling

Processes shall exist to identify and deploy emergency or out-of-band patches for zero-day vulnerabilities or actively exploited flaws, with rapid validation and rollback options.

4.14 VPM-14: Configuration Vulnerability Coverage

Vulnerability scans shall include checks for insecure configurations, missing hardening controls, and deviations from approved configuration baselines.

4.15 VPM-15: Web Application Vulnerability Scanning

Web applications shall be scanned using tools capable of identifying application-specific vulnerabilities such as injection flaws, authentication weaknesses, and insecure direct object references.

4.16 VPM-16: Remediation Verification and Validation

Post-remediation scans or validation checks shall be performed to confirm that vulnerabilities have been successfully mitigated or remediated.

4.17 VPM-17: Vulnerability Risk Register Integration

High-risk vulnerabilities and unpatched systems shall be tracked in the organizational risk register with documented rationale, mitigation plans, and acceptance (if applicable).

4.18 VPM-18: Reporting and Metrics for Vulnerability Management

Organizations shall define and report on key vulnerability management metrics, including mean time to remediate (MTTR), patch compliance rates, and vulnerability recurrence trends.

4.19 VPM-19: Vulnerability Disclosure and Intake Management

A process shall exist for securely receiving and evaluating vulnerability disclosures from external researchers, vendors, or partners, including defined triage and response procedures.

4.20 VPM-20: Container and Image Vulnerability Scanning

All container images and infrastructure-as-code artifacts shall be scanned for vulnerabilities before deployment and on a recurring basis throughout their lifecycle.

4.21 VPM-21: Patch Management for Offline and Isolated Systems

Special procedures shall be established to manage and apply patches to air-gapped, offline, or isolated systems, including validation protocols and manual application tracking.

4.22 VPM-22: Coordination with Change Management

Patch and vulnerability remediation activities shall be integrated with the organization's change management processes to ensure proper scheduling, rollback planning, and impact evaluation.

4.23 VPM-23: Vulnerability Exposure Analysis and Attack Path Modeling

Vulnerability management processes shall incorporate attack path analysis or exploit chain modeling to better understand potential adversary movement through the environment.

Section 5. API Security & Secure Coding Practices (API)

The API Security & Secure Coding Practices (API) control family defines the technical and procedural safeguards required to protect Application Programming Interfaces (APIs) and the secure development behaviors that reduce the introduction of exploitable vulnerabilities in application code. These controls focus on API lifecycle protection, secure interface exposure, input/output sanitization, and developer accountability to ensure resilient applications that uphold organizational security and data integrity objectives.

5.1 API-01: API Inventory and Discovery

Organizations shall maintain an accurate, continuously updated inventory of all APIs (internal, external, and third-party), including documentation of endpoints, functions, protocols, and exposure level.

5.2 API-02: API Authentication Enforcement

All APIs shall enforce authentication using strong, standards-based methods (e.g., OAuth 2.0, JWT), with no unauthenticated endpoints permitted for sensitive or transactional functions.

5.3 API-03: API Authorization and Access Scoping

Access to APIs shall be governed by granular authorization controls (e.g., RBAC, ABAC) and scope limitations that enforce least privilege at the method and resource level.

5.4 API-04: Input Validation and Output Encoding

All APIs shall perform strict input validation (including type, format, length, and encoding) and encode outputs to prevent injection and data exposure vulnerabilities.

5.5 API-05: Rate Limiting and Throttling

APIs shall implement rate limiting and throttling mechanisms to mitigate brute force, enumeration, or denial-of-service attempts.

5.6 API-06: Secure API Gateways and Proxies

All externally exposed APIs shall route through an API gateway or reverse proxy that enforces security policies, monitors traffic, and provides centralized logging.

5.7 API-07: Versioning and Deprecation Strategy

API versioning shall be explicitly managed, with a defined process for deprecation, retirement, and notification to prevent the use of unsupported or vulnerable interfaces.

5.8 API-08: TLS Enforcement for All API Communications

All API traffic—internal and external—shall be encrypted using Transport Layer Security (TLS) with current, approved cipher suites and certificate pinning where applicable.

5.9 API-09: Schema Validation and Contract Enforcement

API requests and responses shall conform to defined schemas (e.g., OpenAPI/Swagger), with automated validation in place to enforce adherence to data contracts.

5.10 API-10: Broken Object and Function Level Authorization Testing

API security testing shall include evaluation of object-level and function-level access controls to prevent unauthorized data access or privilege escalation.

5.11 API-11: Secure Coding Standards for APIs

Developers shall adhere to secure coding standards specific to API development, including guidance on stateless design, idempotency, and secure error handling.

5.12 API-12: Secure Session and Token Management

Session tokens and authentication artifacts used in API communications shall be securely generated, stored, transmitted, and expired in accordance with defined lifespans and revocation procedures.

5.13 API-13: Logging and Monitoring of API Calls

API interactions shall be logged in a tamper-resistant manner, capturing metadata such as source IP, method invoked, parameters used, and response code for anomaly detection.

5.14 API-14: API Security Testing in CI/CD

Security testing tools (e.g., fuzzers, API scanners, SAST/DAST integrations) shall be incorporated into the CI/CD pipeline to detect vulnerabilities before APIs are deployed.

5.15 API-15: Improper Asset Exposure Prevention

APIs shall not expose internal objects, debugging functions, stack traces, or environment variables within responses or error messages.

5.16 API-16: Third-Party API Risk Review

All third-party APIs shall undergo security review and contractual risk assessment prior to integration, including examination of their authentication mechanisms, SLAs, and security posture.

5.17 API-17: CORS and Same-Origin Policy Enforcement

Cross-Origin Resource Sharing (CORS) policies shall be defined and enforced to restrict browser-based access to APIs from untrusted domains.

5.18 API-18: Secrets in Code Prevention

Source code repositories shall be routinely scanned to detect and remove embedded secrets, API keys, tokens, or credentials prior to deployment.

5.19 API-19: Secure Defaults and Configuration Hardening

APIs shall be deployed with secure default settings (e.g., disabled debug mode, restrictive CORS policies, minimal privileges) and hardened configurations to reduce attack surface.

5.20 API-20: Replay Attack and Message Integrity Protection

APIs shall implement mechanisms to detect and prevent replay attacks (e.g., nonces, timestamps) and ensure message integrity through signed requests or tokens.

5.21 API-21: Secure Error Handling and Response Codes

APIs shall return minimal, non-descriptive error messages to clients and use appropriate HTTP status codes to prevent information leakage or behavior inference.

5.22 API-22: API Lifecycle Security Governance

A governance process shall exist to track the security posture of APIs across their lifecycle—from design and development through deployment, maintenance, and retirement.