ISAUnited
INSTITUTE OF SECURITY ARCHITECTURE UNITED

Defensible 10

# Annex J (Normative): D10-DevSecOps & Secure SDLC Engineering

Technical Standards

Standards Committee
1-14-2026

**About ISAUnited**

The Institute of Security Architecture United is the first dedicated Standards Development Organization (SDO) focused exclusively on cybersecurity architecture and engineering through security-by-design. As an international support institute, ISAUnited helps individuals and enterprises unlock the full potential of technology by promoting best practices and fostering innovation in security.

Technology drives progress; security enables it. ISAUnited equips practitioners and organizations across cybersecurity, IT operations, cloud/platform engineering, software development, data/AI, and product/operations with vendor-agnostic standards, education, credentials, and a peer community—turning good practice into engineered, testable outcomes in real environments.

Headquartered in the United States, ISAUnited is committed to promoting a global presence and delivering programs that emphasize collaboration, clarity, and actionable solutions to today's and tomorrow's security challenges. With a focus on security by design, the institute champions integrating security into every stage of architectural and engineering practice, ensuring robust, resilient, and defensible systems for organizations worldwide.

## Disclaimer

ISAUnited publishes the ISAUnited Defensible 10 Standards Technical Guide to provide information and education on security architecture and engineering practices. While efforts have been made to ensure accuracy and reliability, the content is provided “as is,” without any express or implied warranties. This guide is for informational purposes only and does not constitute legal, regulatory, compliance, or professional advice. Consult qualified professionals before making decisions.

## Limitation of Liability

ISAUnited - and its authors, contributors, and affiliates - shall not be liable for any direct, indirect, incidental, consequential, special, exemplary, or punitive damages arising from the use of, inability to use, or reliance on this guide, including any errors or omissions.

## Operational Safety Notice

Implementing security controls can affect system behavior and availability. First, validate changes in non-production, use change control, and ensure rollback plans are in place.

## Third-Party References

This guide may reference third-party frameworks, websites, or resources. ISAUnited does not endorse and is not responsible for the content, products, or services of third parties. Access is at the reader’s own risk.

## Use of Normative Terms (“Must”, “Should”)

- Must: A mandatory requirement for conformance to the standard.
- Must Not: A prohibition; implementations claiming conformance shall not perform the stated action.
- Should: A strong recommendation; valid reasons may exist to deviate in particular circumstances, but the full implications must be understood and documented.

## Acceptance of Terms

By using this guide, readers acknowledge and agree to the terms in this disclaimer. If you disagree, refrain from using the information provided.

For more information, please visit our Terms and Conditions page

## License & Use Permissions

The Defensible 10 Standards (D10S) are owned, governed, and maintained by the Institute of Security Architecture United (ISAUnited.org).



## Standards Development & Governance Notice

This standard is one of the ten Parent Standards in the Defensible 10 Standards (D10S) series. Each Parent Standard is governed by ISAUnited's Standards Committee, peer-reviewed by the ISAUnited Technical Fellow Society, and maintained in the Defensible 10 Standards GitHub repository for transparency and version control.

## Contributions & Collaboration

ISAUnited maintains a public GitHub repository for standards development. Practitioners may view and clone materials, but contributions require:

- ISAUnited registration and vetting
- Approved Contributor ID
- Valid GitHub username

All Sub-Standard contributions must follow the Defensible Standards Submission Schema (D-SSF) and are peer-reviewed by the Technical Fellow Society during the annual Open Season.

Obsolete and withdrawn documents should not be used; please use replacements.

**Abstract**

The ISAUnited Defensible 10 Standards provide a structured, engineering-grade framework for implementing robust and measurable cybersecurity architecture and engineering practices. The guide outlines the frameworks, principles, methods, and technical specifications required to design, build, verify, and operate reliable systems.

Developed under the ISAUnited methodology, the standards align with modern enterprise realities and integrate Security by Design, continuous technical validation, and resilience-based engineering to address emerging threats. The guide is written for security architects and engineers, IT and platform practitioners, software and product teams, governance and risk professionals, and technical decision-makers seeking a defensible approach that is testable, auditable, and scalable.

| This document includes a series of Practitioner Guidance, Cybersecurity Students & Early-Career Guidance, and Quick Win Playbook callouts. | |
|---|---|
|  | **Practitioner Guidance-** Actionable steps and patterns to apply the technical standards in real environments. |
|  | **Cybersecurity Student & Early-Career Guidance-** Compact, hands-on activities that turn each section's ideas into a small, verifiable artifact. |
|  | **Quick Win Playbook-** Immediate, evidence-driven actions that improve posture now while reinforcing good engineering discipline. |

Together, these elements help organizations translate intent into engineered outcomes and sustain long-term protection and operational integrity.

# Foreword

## Message from ISAUnited Leadership

Cybersecurity is at a turning point. As digital systems scale, reactive and checklist-driven practices do not keep pace with adversaries. The ISAUnited position is clear: security must be practiced as engineered design, grounded in scientific principles, structured methods, and defensible evidence. Our mission is to professionalize cybersecurity architecture and engineering with standards that are actionable, testable, and auditable.

ISAUnited Defensible 10 Standards: First Edition is a practical framework for that shift. The standards in this book are not theoretical. They translate intent into measurable specifications, controls, and verification, and enable teams to design and operate resilient systems at enterprise scale.

## About This First Edition

This edition publishes 10 Parent Standards, one for each core domain of security architecture and engineering. Sub-standards will follow in subsequent editions, contributed by ISAUnited members and reviewed by our Technical Fellow Society, to provide focused, technology-aligned detail. Adopting the Parent Standards now positions organizations for seamless integration of Sub Standards as they are released on the ISAUnited annual update cycle.

## Why “Defensible Standards”

Defensible means the work can withstand technical, operational, and adversarial scrutiny. These standards are designed to be demonstrated with evidence, featuring clear architecture, measurable specifications, and verification, so that practitioners can confidently stand behind their designs.

# Contents

# Annex J (Normative): D10-DevSecOps & Secure SDLC Engineering

**ISAUnited's Defensible 10 Standards**
**Parent Standard:** D10-DevSecOps & Secure SDLC Engineering
**Document:** ISAU-DS-DSS-1000
**Last Revision Date:** January 2026
**Peer-Reviewed By:** ISAUnited Technical Fellow Society
**Approved By:** ISAUnited Standards Committee

# Section 1. Standard Introduction

The DevSecOps and Secure SDLC Engineering Parent Standard establishes the architectural expectations and engineering discipline required to build and operate software with validated security at delivery speed. It defines ISAUnited's position on securing source code, pipelines, artifacts, and runtime promotion across contemporary compute environments. The standard also serves as the authoritative foundation for all related sub-standards, whose technical requirements and verification methods derive from it.

The document presents a structured and defensible model for secure software design and delivery. It positions security as an engineering activity expressed in code, continuously validated, and supported by measurable evidence at each release stage. The standard provides engineers with a clear framework for designing, implementing, and governing secure delivery systems. This standard governs the enforcement of delivery and the production of evidence. Secure coding and application design requirements are defined in the Application Security parent standard.

### Objective

This Parent Standard embeds security by design throughout the software lifecycle, including planning, design, coding, build, testing, release preparation, deployment, and post-deployment operations. Its objective is to enable teams to deliver software that withstands adversarial conditions without sacrificing delivery velocity. To achieve this, the standard defines the architectural patterns and engineering controls needed to:

1. Express policies, infrastructure configurations, and tests as code and subject them to the same review and validation practices as application code
2. Enforce non-bypassable security gates within CI/CD workflows
3. Validate artifact integrity and provenance through SBOMs, signatures, and attestations
4. Incorporate application security testing and threat modeling early in development and sustain them throughout delivery
5. Maintain operational safety through progressive delivery, automated rollback, and verifiable promotion criteria

### Justification

Modern software delivery pipelines frequently encounter recurring failure modes that traditional compliance guidance does not address. Dependency chains, build systems,

and registries remain frequent points of compromise, necessitating signed artifacts, attestations, and verifiable provenance. Secrets embedded in code, images, or pipelines create avenues for identity misuse, necessitating short-lived credentials, centralized issuance, and full auditability. Infrastructure-as-code misconfigurations and platform drift introduce exploitable conditions unless policy-as-code is used to evaluate and block violations during build and deploy stages. API-centric architectures expose authorization weaknesses and token misuse risks that necessitate continuous testing, rather than a one-time review at release.

Delivery speed without engineering safeguards converts pipelines into rapid distribution channels for security defects. To remain defensible, organizations require a standard that specifies observable outputs, repeatable verification methods, and release evidence that withstands technical scrutiny. This Parent Standard establishes the engineering model and provides the structural backbone from which domain sub-standards, such as API security, supply chain integrity, policy enforcement for IaC, and runtime protection, derive specific and testable requirements.

**Evidence**

Evidence Packs (EPs) provide the proof layer for adopting this Parent Standard. For Domain 10, the Evidence Pack repository is EP-10 (D10) and is organized to mirror the sections that drive traceability and adoption:

- EP-10.1 Requirements (Inputs)
- EP-10.2 Technical Specifications (Outputs)
- EP-10.3 Foundational Standards
- EP-10.4 Control Mappings
- EP-10.5 Verification and Validation activities.

This structure links architectural intent in Section 5 to measurable implementation in Section 6, and then to Verification and Validation in Section 12, enabling organizations to demonstrate conformance through repeatable, time-bound artifacts rather than declarations.

## Section 2. Definitions

These definitions ensure a consistent understanding and interpretation across ISAUnited members, implementers, and peer reviewers, supporting defensible

engineering and implementation practices. Where possible, definitions align with industry-recognized terminology from NIST, ISO, and ISAUnited's internal frameworks and methodologies.

Admission control – Deploy-time enforcement that evaluates artifacts, configuration, and policy compliance before promotion is permitted.

Admission controller – The enforcement component that performs admission control checks and rejects non-compliant artifacts or deployments.

Application and API security testing – Negative and positive tests validating authentication, authorization, and token handling for APIs and services, used in this standard as release enforcement signals.

Architecture Decision Record (ADR) – A structured, versioned record of an architectural decision, rationale, and consequences, linked to change records and Evidence Pack references.

Artifact signing – Cryptographic signing of artifacts and associated metadata, with verification enforced during deployment.

Breach and attack simulation (BAS) – Adversary simulation used to validate detection coverage and operational response behaviors.

Broken function level authorization (BFLA) – An API authorization failure class that occurs when functions are accessible without proper authorization checks.

Broken object level authorization (BOLA) – An API authorization failure class that occurs when object-level access is not enforced correctly.

Broken object property level authorization (BOPLA) – An API authorization failure class that occurs when access to sensitive object properties is not enforced correctly.

Canary deployment – A progressive delivery pattern where a small subset receives a new version first, with measured rollback criteria.

CAP_SYS_ADMIN – A privileged Linux capability that provides broad administrative power and is prohibited for hardened containers in this standard.

Common vulnerability scoring system (CVSS) – An industry standard for scoring vulnerability severity used in classification and gating thresholds.

Common vulnerabilities and exposures (CVE) – An industry taxonomy for enumerating publicly known vulnerabilities.

Common weakness enumeration (CWE) – An industry taxonomy for enumerating weakness classes.

Continuous delivery or continuous deployment (CD) – The promotion and deployment portion of CI/CD that publishes and deploys artifacts into environments with admission checks, promotion criteria, and rollback controls.

Continuous integration (CI) – The build and test portion of CI/CD, typically executed on pull requests and merges to validate code, dependencies, and infrastructure definitions.

Continuous integration and continuous delivery (CI/CD) – Automated workflows that build, test, and promote changes through defined stages, with gates and evidence produced at merge and deploy boundaries.

Container hardening – Security posture for build and run images, including non-root execution, minimized base images, restricted Linux capabilities, seccomp or AppArmor profiles, read-only root filesystem where feasible, and resource limits.

CODEOWNERS – A repository rule set that assigns required reviewers for defined paths and enforces ownership-based approval.

Deterministic build – A build process that produces stable outputs from defined inputs and eliminates non-deterministic sources such as unpinned dependencies or variable build metadata.

DevSecOps – An engineering discipline that expresses security as code across planning, build, test, release, and runtime workflows, enforcing non-bypassable gates and producing release-ready evidence.

Drift detection – Automated identification of configuration divergence from declared infrastructure and policy baselines, triggering reconciliation or rollback.

Dynamic application security testing (DAST) – Testing of running applications and APIs to identify exploitable conditions in staging or pull request environments.

Egress allowlist – Explicit, minimal outbound destinations permitted for workloads and CI/CD runners, enforced and validated prior to deploy.

Environment parity – The degree to which staging mirrors production control posture, including authorization, transport, egress, and logging schema, so tests remain predictive.

Evidence Pack (EP) – The evidence repository structure used to demonstrate conformance. For Domain 10, EP-10 is organized into five section-aligned locations: EP-10.1 Requirements (Inputs), EP-10.2 Technical Specifications (Outputs), EP-10.3

Foundational Standards, EP-10.4 Control Mappings, and EP-10.5 Verification and Validation activities. Evidence Pack references are used to link prerequisites, implementation proof, and test outcomes.

Interactive application security testing (IAST) – Instrumented testing that observes code behavior at runtime to locate vulnerabilities during functional tests.

Infrastructure as code (IaC) – Declarative definitions of infrastructure and platform resources that are version-controlled and validated prior to deployment.

Key performance indicator (KPI) – A measurement used to track delivery integrity and control effectiveness over time.

Known exploited vulnerability (KEV) – A vulnerability with credible evidence of active exploitation that requires elevated prioritization and pipeline blocking in this standard.

MITRE ATT&CK – A knowledge base of adversary tactics and techniques used for validation scenarios and threat alignment.

Mutual TLS (mTLS) – Transport security where both client and server present certificates, used for service and administrative channels.

Policy as code (PaC) – Machine-enforced rules that validate configurations for network, identity, cryptography, logging and telemetry, and platform hardening in CI/CD and admission paths.

Progressive delivery – Controlled deployment strategies such as canary and blue-green releases with health SLOs and automatic rollback conditions.

Provenance – Cryptographically verifiable statements about how and by whom an artifact was built.

Reproducible build – A build process that deterministically produces bit-identical artifacts from the same source and inputs, enabling integrity verification.

Rollback – Automated reversion to a prior known-good version when health checks, tests, or security gates fail.

Runner – An execution environment that runs CI/CD jobs and requires isolation, scoped identity, and controlled egress.

Runner isolation – Controls that prevent cross-job contamination and limit lateral movement from runner environments, including workspace and cache separation.

Seccomp and AppArmor – Linux security mechanisms used to constrain container behavior through syscall filtering and mandatory access control profiles.

Secrets management – Issuance, storage, delivery, rotation, and revocation of credentials via a centralized system. This standard prohibits secrets in repositories and container layers and requires short-lived scoped credentials.

Secure software development lifecycle (SSDLC) – A lifecycle that integrates security engineering activities and verification checkpoints into standard SDLC phases.

Service level objective (SLO) – A measurable reliability or security performance target used for gating and acceptance decisions.

Software bill of materials (SBOM) – A machine-readable inventory of components and versions for each build artifact, produced at build time and retained with the artifact.

Software composition analysis (SCA) – Identification of third-party components, versions, licenses, and known vulnerabilities, with policy-driven gating.

Supply-chain levels for software artifacts (SLSA) – A framework for supply chain integrity that defines build provenance expectations and aligns with signed attestations.

Threat modeling – A structured analysis, often STRIDE-based, that identifies assets, trust boundaries, threats, and mitigations. In this standard, threat model deltas are recorded with material architectural pull requests.

Trace ID and control ID – Standard log fields for correlating operations and referencing specific control checks or policy evaluations within evidence.

Transport layer security (TLS) 1.3 – The required modern TLS protocol version for edge transport where feasible within scope.

Version control system (VCS) – A system that tracks changes to code and configuration, including branch controls, review workflows, and audit history.

Verify-on-pull – Deployment enforcement that verifies signatures and attestations when an artifact is pulled for deployment, rejecting artifacts that fail validation.

## Section 3. Scope

DevSecOps and Secure SDLC Engineering covers the engineering practices, delivery platforms, and control mechanisms required to design, build, test, release, and operate

software with verifiable security at delivery speed. Modern enterprises distribute workloads across on-premises environments, public and private cloud platforms, SaaS systems, edge architectures, and multi-tenant computing environments. These environments increase the difficulty of securing pipelines, dependencies, build systems, artifacts, promotion paths, and runtime behavior. This Parent Standard defines the architectural expectations and technical guardrails that establish a defensible DevSecOps posture across these environments. The scope ensures that software delivery systems block supply chain compromise, enforce non-bypassable security gates, validate artifact integrity and provenance, eliminate secrets sprawl, maintain environment parity, and produce auditable evidence while remaining aligned with organizational risk tolerance and delivery objectives.

## Applicability

- All Software Delivery Artifacts and Paths – Applies to source code, infrastructure as code, policy repositories, CI/CD pipelines, build artifacts such as containers or images, SBOMs, signatures and attestations, deployment manifests, and operational configuration.
- Enterprise and Academic Environments – Intended for software engineers, application security teams, DevSecOps engineers, SRE and platform engineers, detection and incident response teams, and academic programs advancing secure software engineering education.
- Hybrid and Multi-Platform Architectures – Governs DevSecOps controls across data centers, multiple cloud providers, orchestration platforms, serverless and edge compute, and shared build or artifact systems.
- Environment Coverage – Applies to development, test, staging, production, and regulated environments. Exceptions for legacy or constrained systems require compensating controls and time-bound remediation.

## Key Focus Areas

- Everything as Code Governance – Version control with protected branches and signed commits, traceable architectural decisions, and automated rollback definitions.
- Gated CI/CD – Non-bypassable SAST, SCA, and IaC policy evaluations, image scanning, and fail-closed gates at merge and release stages. Artifact signing and verify-on-pull enforcement are mandatory at deployment.
- Software Supply Chain Integrity – Reproducible builds, SBOM generation and retention, provenance and attestation validation, and KEV-aware prioritization and block conditions.

Obsolete and withdrawn documents should not be used; please use replacements.

- Secrets and Pipeline Identity – Central issuance of short-lived and scoped credentials, zero secrets in repositories and images, full auditability of access, and controlled rotation upon compromise.
- Application and API Security Testing – Continuous SAST, DAST, and IAST, API authentication and authorization tests including BOLA and BOPLA, and token lifecycle validation.
- Dependency and Container Security – License and vulnerability policy enforcement, controlled base-image lifecycle, and hardened container images with non-root execution, minimal capabilities, seccomp or AppArmor profiles, and read-only root filesystems.
- Environment Parity and Transport Controls – Staging environments mirror production controls. TLS 1.3 is required at edges, mutual TLS is required for service and administrative paths, egress allowlists are defined, and CI/CD runners are isolated.
- Observability and Evidence – Unified logging schema, immutable Evidence Pack per release containing SBOMs, signatures, test logs, parity results, and rollback artifacts, with explicit SLO met or not met status.
- Post-Deploy Validation – Progressive delivery with automatic rollback, and breach and attack simulation or ATT and CK scenarios to validate detection coverage and operational readiness.

**Outcomes**

By defining this scope, the standard ensures DevSecOps and Secure SDLC Engineering produce repeatable outcomes across the Defensible Loop:

- **Define:** Bound pipeline stages and release boundaries. Identify trusted sources, authoritative registries, promotion paths, and the minimum evidence set required to prove delivery integrity.
- **Design:** Specify secure pipeline architecture and provenance intent. Define gate logic, policy-as-code enforcement points, identity trust boundaries, and acceptance criteria for promotion and rollback.
- **Deploy:** Implement non-bypassable gates, signing and attestations, controlled deployments, environment parity controls, and automated rollback and revocation workflows as engineered delivery behaviors.
- **Detect:** Instrument gate outcomes and integrity signals across build, registry, deploy, and runtime promotion. Detect policy violations, drift, secret exposure, and anomalies in artifact verification or promotion chains.
- **Defend:** Execute operational containment actions for delivery compromise, including artifact quarantine, signing key revocation, credential rotation, rollback execution, and exception closure with time bounds and compensating controls.
- **Demonstrate:** Produce release-grade proof, including attestations and deployment trace evidence that links requirements to outputs and to verification and validation artifacts stored in the Evidence Pack structure for Domain 10.

Obsolete and withdrawn documents should not be used; please use replacements.

Together, these phases provide the foundation for resilient, auditable, and high-velocity software delivery that preserves integrity across source, pipeline, artifact, and promotion workflows, and that concludes with proof rather than confidence statements.

## Section 4. Use Case

Achieving resilient software delivery requires more than scanners and policies—it demands engineered practice across source, pipelines, artifacts, and runtime. The following consolidated use case reflects a complex, real-world scenario typical of organizations that deliver services across on-premises, multi-cloud, and SaaS platforms. It exposes common supply chain and pipeline weaknesses, ties them to concrete DevSecOps controls (gated CI/CD, SBOM/provenance, secrets discipline, environment parity), and maps each weakness to targeted, testable outcomes. The result is an operational playbook that links day-to-day delivery actions—build, test, release, deploy—to measurable, defensible reductions in exploitability and time-to-rollback.

**Table J-1:**

| **Use Case Name** | **Securing the Software Supply Chain and CI/CD at Enterprise Scale** |
|---|---|
| Objective | Prevent supply chain compromise, eliminate secret sprawl, enforce artifact integrity/provenance, and ensure fast and safe rollbacks—without sacrificing delivery velocity. |
| Scenario | A global SaaS provider ships weekly across multiple clouds. Recent incidents included an unsigned image reaching production, long-lived credentials in a build container, and a dependency added to the KEV list later. Outages and emergency patches eroded trust. The organization lacked non-bypassable gates, SBOM/provenance coverage, and parity between staging and production. |
| Actors | Principal Software Engineer, DevSecOps Lead, AppSec Engineer, SRE/Platform Engineer, Release Manager, Product Team Lead |
| Adversary mapping | Design-time Threat Models: STRIDE categories – Spoofing, Tampering, Repudiation, Information Disclosure, Elevation of Privilege. |

| | |
|---|---|
| | ATT&CK examples: T1195 Supply Chain Compromise; T1552 Unsecured Credentials; T1555 Credentials from Password Stores; T1078 Valid Accounts; T1098 Account Manipulation; T1562.001 Impair Defenses: Disable or Modify Tools; T1609 Container Administration Command; T1610 Deploy Container; T1485 Data Destruction.<br><br>Kill Chain Phases: Weaponization (malicious dependency or build tooling), Delivery (registry or pipeline insertion), Exploitation (credential misuse or gate bypass), Installation (artifact substitution or secret persistence), C2 (abuse of trusted service identities), Actions on Objectives (production compromise through unverified promotion, lateral movement, or data access).<br><br>Failure Vectors Addressed: Unsigned or un-attested artifacts, compromised build runners, secret sprawl in repositories or images, advisory-only gates, registry tampering, environment parity gaps, and manual rollback delays. |
| Challenges Identified | • No verify-on-pull at deploy; unsigned artifacts accepted<br>• Secrets in repos and container layers; long-lived tokens in CI<br>• SAST/SCA gates advisory only; merges proceeded on High findings<br>• No SBOM; no provenance/attestation; weak image hygiene<br>• Staging lacked TLS/mTLS parity with production; egress open<br>• Rollbacks manual; evidence packs incomplete |
| Technical Solution | 1) Gated CI/CD: Fail-closed SAST, SCA, infrastructure as code, and image scanning gates. High and Critical findings at merge equal 0. KEV block list enforced.<br>2) SBOM, provenance, and signing: Generate an SBOM for 100 % of artifacts. Sign and attest at publish. Enforce verify-on-pull at deploy.<br>3) Secrets and identity: Central secrets platform with short-lived, scoped tokens, with a TTL of 24 h or less. Pre-commit and CI scanners block secrets in repositories and images.<br>4) API and application security testing for release enforcement: DAST and IAST are executed in pull request or staging for internet-exposed services. API authentication and authorization tests cover BOLA and BOPLA. Token expiry and rotation validated.<br>5) Environmental parity and transport controls: Staging mirrors production controls. TLS 1.3 is enforced at edges. Mutual TLS is enforced for service and administrative paths. Egress allowlists enforced. Runners isolated.<br>6) Build hygiene: Reproducible builds with pinned inputs and no latest tags. Images execute as non-root. Restricted capabilities enforced. seccomp or AppArmor profiles defined. Read-only root filesystem enforced where feasible.<br>7) Observability and evidence production: Unified logging schema. Evidence Pack captured per release, including SBOMs, signatures and attestations, scan results, test logs, parity results, and rollback logs.<br>8) Post-deploy validation and rollback: Canary or blue-green deployments with health SLOs. Adversary simulation scenarios validate detection behavior. Automatic rollback triggers on failure. |
| Expected Outcome | • Artifact integrity: 100% SBOM/provenance coverage; deploy blocks unsigned/unstamped artifacts<br>• Vulnerability posture: KEV at release = 0; base images updated ≤ 30 days |

| | |
|---|---|
| | • Secrets discipline: 0 secrets in repos/images; token TTL ≤ 24h; rotation on compromise ≤ 15 min<br>• Pipeline quality: signed-commit rate ≥ 95% (30-day rolling); High/Critical at merge = 0<br>• Transport parity: mTLS coverage ≥ 98% for service/admin paths; TLS 1.3 at edges in staging and prod<br>• Operations safety: automated rollback executes < 5 min; BAS detections fire ≤ 10 min end-to-end<br>• Evidence: complete Evidence Pack per release with “SLO met / not met” status for gates, transport, and rollback |
| Evidence Artifacts | Signed commit enforcement reports; pipeline gate run logs for SAST, dependency scanning, and infrastructure-as-code policy checks; SBOM exports for released artifacts; signature and attestation bundles; registry audit trails and verify-on-pull denial logs for an unsigned artifact test; short-lived credential issuance and rotation logs; runner isolation and restricted egress validation outputs; staging-to-production parity checks for transport and logging; canary or blue-green promotion records and rollback drill logs; breach and attack simulation outputs validating alert timing and rollback triggers.<br><br>Evidence Pack ID: EP-10.5 (verification and validation artifacts cross-linked to EP-10.2 for technical specifications evidence). |

**Key Takeaways**

- Gates must fail closed – Advisory scans allow defects and exploitable states to ship. Mandatory blocking gates are required to produce a defensible release.
- Provenance and SBOM coverage must be universal – Partial coverage creates blind spots for supply chain compromise. Every artifact must be signed, attested, and accompanied by an SBOM.
- Secrets discipline is foundational – Secrets in repositories, images, or CI variables create persistent risk. Only short-lived, centrally issued identities are acceptable.
- Environment parity determines predictability – Security controls validated in staging cannot be trusted unless staging matches production transport and authorization boundaries.
- Rollback must be automated and reversible at speed – Manual rollback introduces downtime and uncertainty. Automated rollback with evidence-backed triggers is required for operational safety.
- Evidence is the only defensible output – A release is considered secure only when proof exists. Pipelines must produce immutable evidence packs showing what ran, what passed, and what was blocked.

These takeaways reinforce that secure delivery is not a set of tools but an engineered system of constraints, validations, identities, and measurable outputs.

**Practitioner Guidance:**

The following guidance supports engineering teams adopting the patterns demonstrated in this use case:

- Instrument pipeline controls as code – Express required controls as code, store them in version control, and tie them to measurable SLOs. Avoid informal policies that cannot be audited.
- Establish a required baseline – Confirm that branch protections, artifact signing and attestations, centralized secret issuance, and environment parity controls are in place before adopting sub-standards.
- Prove gates early and routinely – Introduce seeded failures such as an unsigned artifact or an embedded secret to confirm fail-closed behavior. Store results in the Evidence Pack.
- Treat identity as an attack surface – Minimize and monitor pipeline, workload, and builder identities, enforce short-lived credentials, and validate rotation. Identity drift creates the same risk profile as configuration drift.
- Validate transport guarantees continuously – Verify TLS and mutual TLS using automated scans and parity checks rather than manual inspection.
- Make exceptions time-bound and controlled – Require a sunset date, compensating controls, and Evidence Pack inclusion for any bypass path, allowlist expansion, or disabled gate.
- Integrate post-deployment validation into release workflows – Run breach and attack simulation scenarios to confirm detection and rollback paths function under adversarial conditions.

## Section 5. Requirements (Inputs)

This section outlines the essential architectural and environmental prerequisites for the successful implementation of this Parent Standard and its associated sub-standards. These inputs are not recommendations; they are baseline conditions that enable the defensibility and enforceability of technical specifications defined across the domain.

### 5.1 Version Control and Branch Protection

All source code, infrastructure as code, policies, pipelines, and playbooks Must be in version control with protected branches, required reviews (CODEOWNERS), and signed commits enabled.

### 5.2 CI/CD Platform with Non-Bypassable Security Gates

Build and deployment pipelines Must exist for all services and enforce security

gates (SAST, SCA, IaC policy checks, image scan) prior to merge and release, with automated rollback steps defined.

**5.3 Trusted Artifact Repositories and Provenance**
A central artifact registry service Must enforce signature verification at publish and pull, store SBOMs alongside artifacts, and record provenance and attestations for all released components.

**5.4 Secrets and Pipeline Identity Management**
A central secrets platform Must issue short-lived, scoped credentials to CI/CD systems and workloads; secrets are never stored in repositories or container layers; access is fully audited.

**5.5 Application Security Test Capability**
SAST, DAST and IAST, API contract testing, and unit and integration test harnesses Must be available; organization-wide severity thresholds and coverage expectations are defined.

**5.6 Policy-as-Code and IaC Guardrails**
Policy engines and rulesets Must exist for network, identity, cryptography, logging and telemetry, and platform hardening; pipelines are integrated to block critical violations.

**5.7 Dependency and Container Security Readiness**
SCA for operating system and application dependencies, and container image scanning, Must be integrated; exploit-in-the-wild and KEV lists are synced for prioritization; a base image lifecycle policy is enforced.

**5.8 Threat Modeling Practice and PR Delta**
A documented threat modeling process Must exist; pull requests that change architecture include a threat model delta and mapped mitigations.

**5.9 Environment Parity and Transport Controls**
A staging environment Must mirror production control posture (authorization, egress, TLS and mutual TLS, logging schemas) so security tests are predictive; environment drift is monitored.

**5.10 Logging Schema and Evidence Store**
A unified log schema Must be defined and enforced; a tamper-evident evidence store is available to retain release artifacts (configurations, SBOMs, signatures, scan results, test logs) for audit.

**5.11 Runner Isolation and Build Execution Security**
CI/CD execution environments Must support runner isolation, scoped identity, restricted egress, and teardown guarantees for build workspaces and caches.

**5.12 Deployment Admission and Promotion Enforcement**
Deployment platforms Must support admission or promotion enforcement capable of rejecting artifacts that fail signature validation, provenance checks, SBOM presence, or vulnerability thresholds.

## Evidence Pack

Record evidence Must be collected for Section 5 prerequisites in EP-10.1 (Requirements). Each requirement in 5.1 through 5.10 Must have at least one dated artifact that identifies the owner, the current status, and the enforcement boundary. Evidence Must be version-controlled and retained according to organizational audit requirements.

Minimum evidence expectations for EP-10.1 include:

- Source and change governance artifacts include protected branch configuration, required review settings, CODEOWNERS rules, and signed-commit enforcement evidence with an audit extract.
- Pipeline readiness artifacts include CI/CD workflow definitions showing non-bypassable gate execution points, rollback steps, and required job enforcement for merge and release stages.
- Artifact registry and provenance readiness artifacts include registry configuration baselines for signature verification and retention, SBOM attachment policy, and provenance or attestation enablement evidence at publish and pull boundaries.
- Secrets and identity readiness artifacts include secrets platform configuration baseline, credential issuance policy for short-lived scoped access, rotation configuration, and audit logging configuration proving traceable access.
- Policy-as-code and infrastructure guardrail artifacts include policy bundles and rule references for network, identity, cryptography, logging and telemetry, and platform hardening, including evidence that critical violations block promotion.
- Dependency and build hygiene readiness artifacts include dependency and image scanning configuration, KEV synchronization evidence, base image lifecycle policy, and approved base image inventory or baseline.
- Environment parity readiness artifacts include staging and production control posture comparison for transport, egress, and logging schema, with drift monitoring configuration evidence.
- Logging and evidence store readiness artifacts include unified log schema definition, schema validation configuration, tamper-evident evidence store configuration baseline, and retention settings.

EP-10.1 entries Must link forward to implementation proof in EP-10.2 (Technical Specifications) and to test results in EP-10.5 (Verification and Validation) where applicable.

**Practitioner Guidance:**

- **Readiness Gate (one page):** List 5.1–5.10 with an owner, current status, and a link to proof. Do not proceed to adopt sub-standards until every row is green with a dated Evidence Pack ID.
- **Baseline First:** Capture current metrics for SAST/DAST fail rates, KEV exposure, mTLS coverage, signed-commit rate, and SBOM coverage. These serve as the control group for measuring §6 outputs.
- **Blockers = Stop Work:** If any of 5.2 (gated CI/CD), 5.3 (signed/provenanced artifacts), or 5.4 (central secrets) is missing, pause downstream tasks and open a tracked risk—§6 cannot be defensible without them.
- **Prove It in Pipeline:** Add quick failing tests now (seeded secret, unsigned image, KEV vuln) to show that gates actually block merges/releases; attach those failing runs to the Evidence Pack.

## Section 6. Technical Specifications (Outputs)

Technical specifications define the defensible engineering outputs required to implement this Parent Standard. Each specification represents a distinct delivery engineering area that translates security-by-design intent into measurable, auditable software factory behaviors across CI/CD, source control, build systems, registries, orchestrators, and runtime environments.

**Outputs must be**:

- **Measurable**: validated by scans, logs, audits, or tests
- **Actionable**: implementation-ready, not policy slogans
- **Aligned**: traceable to §5 Requirements and sub-standards

### 6.1 Everything as Code Governance

- All application, infrastructure as code, policy, and pipeline changes Must occur through version-controlled pull requests with required review enforcement.

Obsolete and withdrawn documents should not be used; please use replacements.

- Protected branches Must enforce commit signing for merge paths.
- Architecture Decision Records or equivalent change records Must be linked to the change that introduced the decision.
- Release processes Must include an automated rollback definition stored as code and promoted with the same governance as deployment changes.

### 6.2 Secure Pipeline Gates

- Pipeline gates Must fail closed for merge and release promotion stages.
- SAST thresholds Must block merge when Critical or High findings are present, and coverage requirements Must be defined for changed code.
- Dependency scanning and image scanning Must block KEV items and Critical or High findings unless a time-bounded exception exists with compensating controls.
- Infrastructure as code policy evaluation Must block Critical violations prior to merge and prior to deploy.
- SBOMs and signed attestations Must be produced for deployable release artifacts, and verify-on-pull Must be enforced at deploy.
- Build identities and deploy identities Must be separated, and build jobs Must not hold production write privileges.

### 6.3 Software Supply Chain Integrity and Build Hygiene

- Builds Must be reproducible and deterministic for in-scope deployment artifacts.
- Release workflows Must not rely on unpinned dependencies or “latest” tags for base images and build inputs.
- Base image lifecycle Must be defined and enforced, including update timelines and inventory control.
- Containers Must execute as non-root and use restricted Linux capabilities with validated seccomp or AppArmor profiles.

### 6.4 Secrets and CI/CD Identity

- Secrets Must not be present in repositories or container layers used for deployable artifacts.
- CI/CD identities Must use short-lived, scoped credentials, and rotation procedures Must exist for compromise triggers.
- Secret access Must be attributable to service identities through complete audit logs.

### 6.5 Application Security Testing as Release Enforcement

- Testing outputs used for promotion decisions Must be integrated into pipeline gates for internet-exposed services.
- Dynamic testing coverage requirements Must be defined for staging or pull-request environments where applicable.

- API negative and positive tests Must validate authentication, authorization, and token lifecycle behavior where services are exposed through APIs.

**6.6 Dependency and Container Security**

- Policies for dependency governance Must define license thresholds, vulnerability thresholds, and KEV blocking behavior.
- Container hardening expectations Must include read-only root filesystem where feasible, resource limits, and prevention of privilege escalation such as CAP_SYS_ADMIN.
- Deployment pathways Must reject artifacts that fail provenance, signature, or attestation validation.

**6.7 Environment Parity and Transport Controls**

- Staging control posture Must mirror production control posture for authorization, logging schema, egress controls, and transport security.
- TLS 1.3 Must be enforced at edges, and mutual TLS Must be used for service and administrative paths where required by the architecture.
- Runner isolation and outbound egress restriction Must be enforced for CI/CD execution environments.

**6.8 Observability and Evidence**

- A unified logging schema Must be implemented across build, deploy, and runtime promotion workflows.
- Release processes Must produce a complete Evidence Pack record set containing the artifacts required to demonstrate output enforcement and release decisions.

**6.9 Threat Modeling and Pull Request Delta**

- Each service Must maintain an updated threat model.
- Architectural pull requests Must include a threat model delta with mitigations mapped to tests or enforcement controls.

**6.10 Post-Deploy Validation and Rollback**

- Deployments Must use progressive delivery patterns where risk and criticality justify controlled promotion.
- Rollback paths Must be automated, and rollback triggers Must be bound to defined health criteria.
- Adversary simulation or equivalent validation activities Must be executed on a defined cadence for services with internet exposure or elevated risk.

**Evidence Pack**

Evidence Must be collected for Section 6 technical specifications in EP-10.2 (Technical Specifications). Each output in 6.1 through 6.10 Must include at least one dated artifact that demonstrates implementation, enforcement, and the applicable measurement point. Evidence Must be version-controlled and retained according to organizational audit requirements.

Minimum evidence expectations for EP-10.2 include:

- Everything as code governance evidence includes branch protection settings, commit signing enforcement reports, pull request audit extracts, ADR links, and rollback definition artifacts.
- Secure pipeline gate evidence includes pipeline run logs, gate results, blocked merge records, KEV blocking outputs, and verify-on-pull denial logs from an unsigned artifact attempt.
- Supply chain and build hygiene evidence includes SBOM exports, attestation bundles, rebuild parity outputs where applicable, base image inventory and lifecycle records, and container hardening scan outputs.
- Secrets and identity evidence includes secret scanning outputs, credential issuance and TTL policy evidence, rotation logs for compromise triggers, and secret access audit trails.
- Testing as release enforcement evidence includes promotion gate configuration tied to test outputs, test result summaries for in-scope services, and artifact links to the release record.
- Dependency and container security evidence includes dependency policy baselines, vulnerability and license threshold enforcement outputs, and provenance verification results prior to deployment.
- Environment parity and transport evidence includes parity checks for staging versus production, TLS and mutual TLS validation outputs, runner isolation evidence, and outbound egress restriction validation.
- Observability and evidence production include unified logging schema definition, schema validation results, and release Evidence Pack completeness records.
- Threat modeling delta evidence includes threat model artifacts, pull request delta records, and mitigation to test mappings.
- Post-deploy validation and rollback evidence includes progressive delivery logs, rollback trigger configuration and execution logs, and simulation results where required by scope.

Entries in EP-10.2 Must link back to EP-10.1 (Requirements) to demonstrate prerequisite readiness and Must link forward to EP-10.5 (Verification and Validation) for test execution artifacts and formal acceptance results.

**Practitioner Guidance:**

- Instrument failing tests early. Seed a High-severity SAST flaw, a KEV dependency, and an embedded secret to confirm pipeline gates block merges or releases.
- Elevate transparency. Maintain service dashboards tracking signed-commit rate, SBOM and provenance coverage, mutual TLS coverage, KEV exposure, and exception counts.
- Maintain exception discipline. Any bypass, allowlist expansion, or disabled rule requires a compensating control, a sunset date, an approval record, and inclusion in the Evidence Pack.
- Integrate identity and transport controls. Link CI/CD identities to the IAM domain policy, and ensure that interservice traffic adheres to CEK-aligned TLS configurations.
- Validate runtime conditions. Simulation outputs and rollback behavior should be included in release acceptance reviews for in-scope services.

**Quick Win Playbook**:

**Title:** Artifact Signing and Promotion Integrity Enforcement

**Objective:** Establish a deploy-time enforcement path that blocks unsigned or un-attested artifacts, validates provenance at promotion boundaries, and produces evidence artifacts that support release defensibility.

**Target**: Enforce artifact signing, attestation validation, and verify-on-pull in one critical deployment environment.

**Component/System**: CI/CD pipelines, artifact registry, admission enforcement point, deployment stack.

**Protects**: Prevents unsigned or tampered artifacts from entering promotion paths and blocks supply chain insertion through artifact substitution.

**Stops and Detects**: unsigned artifacts, revoked signing keys, repackaged artifacts, and missing attestations.

**Action**: Enable signing and attestation at build. Enforce verify-on-pull at deploy. Execute one negative test by attempting to deploy an unsigned artifact and confirming denial.

**Proof**: Artifacts stored in EP-10.2 and cross-linked to EP-10.5 include signing policy diffs, attestation bundles, denial logs, provenance verification logs, and a

| | |
|---|---|
| | successful deploy record for a valid artifact.<br><br>**Metric**: 100 % deployable artifacts are signed and attested, 0 unsigned artifacts are admitted, and verify-on-pull checks succeed for accepted artifacts.<br><br>**Rollback**: Revert enforcement only through a time-bounded exception and record compensating controls and a sunset date in the Evidence Pack. |

## Section 7. Cybersecurity Core Principles

This section identifies the foundational security architecture and engineering principles that guide the intent, design, and implementation of this Parent Standard. These principles are drawn from the ISAUnited Recommended Principles (ISAU-RP) catalog and represent the enduring philosophies that shape secure system architecture and defensible engineering practices across all domains.

### Purpose and Function

Security principles provide more than technical direction—they embed discipline, clarity, and foresight into every recommendation. By grounding technical specifications and implementation strategies in well-defined principles, ISAUnited ensures that sub-standards do not merely respond to threats tactically but are built to withstand architectural risk over time.

**Table J-2. Examples of Applicable Principles:**

| ISAU-RP ID | Principle Name | Justification for DevSecOps & Secure SDLC Engineering |
|---|---|---|
| ISAU-RP-01 | Least Privilege | Enforces tightly scoped permissions for CI/CD identities, runners, build systems, registries, and deployment controllers. Prevents horizontal and vertical escalation within pipelines and source repositories. |
| ISAU-RP-02 | Zero Trust | Requires explicit verification of artifacts, identities, and actions at every stage of delivery. Aligns directly to verify-on-pull, SBOM, and provenance validation, attestation checks, and mutual TLS across service and administrative paths. |

| ISAU-RP ID | Principle Name | Justification for DevSecOps & Secure SDLC Engineering |
|---|---|---|
| | | |
| ISAU-RP-03 | Complete Mediation | Ensures that pipeline gates, admission controllers, provenance validators, and signature checks cannot be bypassed. Supports fail-closed behavior for SAST, SCA, IaC checks, and image scanning at merge and deploy. |
| ISAU-RP-04 | Defense in Depth | Provides layered security across source control, build steps, artifact storage, admission pathways, runtime environments, and post-deploy validation. Reinforces multi-point enforcement of trust and integrity. |
| ISAU-RP-05 | Secure by Design | Embeds security considerations at planning, design, coding, building, testing, and deployment phases. Supports threat modeling, PR deltas, structured rollback definitions, and supply chain engineering. |
| ISAU-RP-10 | Secure Defaults | Drives default-deny settings for pipeline gates, unsigned-artifact rejection, secret scanning, runner isolation, and environment parity. Ensures that unsafe configurations require explicit override and justification. |
| ISAU-RP-12 | Security as Code | Central to DevSecOps. Requires policies, guardrails, controls, tests, and evidence generation to be automated, version-controlled, peer-reviewed, and executed via CI/CD. Enables enforceable, auditable security at scale. |
| ISAU-RP-15 | Evidence Production | Directly aligns with Evidence Packs, SBOM retention, provenance bundles, scan results, and traceable release decisions. Ensures every release includes verifiable artifacts proving conformance and defensibility. |
| ISAU-RP-14 *(Recommended)* | Resilience and Recovery | Supports progressive delivery, automated rollback, health SLO validation, and recovery actions required for safe deployment in multi-cloud and distributed architectures. |
| ISAU-RP-16 *(Recommended)* | Make Compromise Detection Easier | Justifies mandatory unified logging schemas, trace identifiers, mTLS attribution, BAS and ATT, and CK validation, and runtime observability throughout the delivery chain. |

***Note**: Organizations may include a matrix mapping each selected principle to its associated technical outputs or control mappings, further demonstrating traceability.*

# Section 8. Foundational Standards Alignment

This section identifies the internationally recognized foundational frameworks that support and align with the architectural direction of this Parent Standard. These foundational standards provide essential baselines for security, governance, and risk management that ISAUnited builds upon to define defensible, engineering-driven standards.

### Purpose and Function

While ISAUnited does not duplicate existing compliance frameworks, it acknowledges their critical role in shaping baseline expectations for cybersecurity architecture and control design. This section:

- Demonstrates alignment with global best practices
- Bridges compliance frameworks with ISAUnited's engineering-focused approach
- Enhances credibility and traceability for enterprise adoption and audit-readiness
- Establishes a consistent reference point for sub-standards to map technical controls

**Table J-3. Applicable Foundation Standards:**

| Framework | Standard ID | Reference Focus |
|---|---|---|
| NIST | SP 800-218 | Secure Software Development Framework (SSDF) – secure SDLC tasks and practices |
| NIST | SP 800-53 Rev. 5 | Security & Privacy Controls – AC, AU, CM, RA, SA, SI families relevant to delivery pipelines and assurance |
| NIST | SP 800-160 Vol. 1 | Systems Security Engineering – life-cycle engineering, evidence, and trustworthiness |

| Framework | Standard ID | Reference Focus |
|---|---|---|
| NIST | SP 800-161 Rev. 1 | Cyber Supply Chain Risk Management – supplier, component, and artifact assurance |
| NIST | SP 800-204 Series | Microservices/Container/Kubernetes security architecture and hardening (as applicable) |
| ISO/IEC | 27001:2022 | ISMS requirements – governance and risk integration for software delivery controls |
| ISO/IEC | 27002:2022 | Control catalog – implementation guidance mapped to pipeline, runtime, logging, and change control |
| ISO/IEC | 27034-1 | Application Security – organizational processes for secure application development and operation |

***NOTE**: As detailed sub-standards are developed under this parent standard, specific references to NIST and ISO will be incorporated to provide control-level alignment and practical implementation guidance for DSS practitioners.*

***NOTE**: ISAUnited Charter Adoption of Foundational Standards.*

*Per the ISAUnited Charter, the institute formally adopts the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) and the National Institute of Standards and Technology (NIST) as its foundational standards bodies, consistent with their public encouragement of organizational adoption. Parent Standards align with ISO/IEC and NIST for architectural grounding and auditability, and this alignment cascades down to Sub-Standards as invariant, minimum requirements that may be tightened but not weakened. ISAUnited does not restate or speak on behalf of ISO/IEC or NIST; practitioners shall consult the official publications and terminology of these organizations, verify scope and version currency against the latest materials, and implement controls in a manner consistent with ISAUnited security invariants and the requirements of this standard.*

**Sub-Standard Expectations:**

Sub-standards developed under this Parent Standard must:

Obsolete and withdrawn documents should not be used; please use replacements.

- Reference one or more of the foundational standards above, where applicable
- Extend these foundational expectations into domain-specific engineering controls
- justify any intentional divergence from foundational principles or models

**Evidence Pack**

Evidence Must be collected for Section 8 foundational standards alignment in EP-10.3 (Foundational Standards). Each referenced NIST or ISO standard in Table J-3 Must have at least one dated mapping artifact that identifies the applicable clause or practice, the affected §6 output, the enforcement mechanism, and the Evidence Pack cross-reference for proof.

Minimum evidence expectations for EP-10.3 include:

- A clause-level mapping sheet linking §6 outputs to NIST SP 800-218 practices and to applicable clauses in the remaining foundational standards in Table J-3.
- Version-controlled records showing when mappings were created or updated, including pull request references and change rationale.
- A divergence register documenting any intentional deviation from a referenced clause or model, including compensating controls, a sunset date, and the verification method used to demonstrate an equivalent or stronger effect.
- Cross-links from the mapping sheet to implementation artifacts in EP-10.2 (Technical Specifications) and to verification and validation artifacts in EP-10.5 (Verification and Validation) where applicable.

Entries in EP-10.3 Must remain current. Any change to a gate, policy, signing rule, or transport profile that impacts a mapped clause Must update the mapping in the same change record and include an updated evidence reference.

**Practitioner Guidance**:

- Map at clause level: For each §6 output (for example, 6.2 pipeline gates and 6.3 supply chain integrity), add a row to your service mapping sheet that lists the specification identifier, the NIST or ISO clause, how the clause is enforced, and the Evidence Pack reference. Store this mapping in EP-10.3 and cross-link it to EP-10.2 and EP-10.5 where applicable.
- SSDF as backbone: Ensure every relevant SSDF practice (SP 800-218) is backed by a concrete §6 output and passing artifacts in §12.
- Keep mappings current: When a gate or policy changes, update the NIST or ISO clause reference in the same pull request and store the diff in EP-10.3.
- Divergence discipline: If a clause cannot be met verbatim, record the compensating control, the sunset date, and a verification result

| | |
|---|---|
| | demonstrating an equivalent or stronger effect. Store the record in EP-10.3 and cross-link validation artifacts in EP-10.5. |

# Section 9. Security Controls

This section identifies the technical control families and control references that this Parent Standard directly supports or enforces. These controls map the standard's architectural and engineering guidance to recognized cybersecurity frameworks, enabling traceability, auditability, and consistent implementation across diverse environments.

### Purpose and Function

Security controls translate architectural intent into actionable safeguards. They provide the tactical grounding for how a system enforces confidentiality, integrity, availability, authentication, authorization, and auditability.

By mapping to widely accepted frameworks such as CSA CCM, CIS Controls v8, and OWASP ASVS or API Top 10, ISAUnited ensures:

- Alignment with industry best practices and compliance frameworks
- Cross-organizational interoperability and third-party validation
- Reusability across sub-standards under the same Parent domain

These mappings also help engineers and reviewers understand the defensibility of each technical output within the domain.

### Implementation Guidance

When defining DSS sub-standards or producing implementation evidence

- Reference at least three technical controls from one or more authoritative control frameworks of CIS, CSA, and or OWASP.
- Provide the framework acronym, control ID, and optionally a short label or explanation.
- Select controls that support the technical outputs or security principles defined in this Parent Standard.

Obsolete and withdrawn documents should not be used; please use replacements.

- Avoid vague or policy-level controls—focus on implementation-level or enforceable technical safeguards.

**Table J-4. Control Mappings for DevSecOps & Secure SDLC Engineering:**

| Framework | Control ID | Control Name / Description |
|---|---|---|
| CSA CCM | IAM-05 | Least-Privilege Access – Enforce scoped, short-lived identities for CI/CD, registries, and platforms. |
| CSA CCM | SEF-01 / SEF-02 | Security Event Logging & Management – Centralize and monitor delivery-system and application security events. |
| CSA CCM | TVM-01 | Threat & Vulnerability Management – Gate builds and deploys on risk (SCA, image scans, KEV blocking). |
| CSA CCM | DCS-03 | Data Integrity & Provenance – Require artifact signing, attestations, and verification upon deployment pull. |
| CSA CCM | CCC-03 | Change Management Technology - Change control and configuration management for organizational assets, including applications, systems, infrastructure, and configuration. |
| CIS v8 | 2.x | Inventory and Control of Software Assets – Track code, images, packages, and manifests across environments. |
| CIS v8 | 4.x | Secure Configuration of Enterprise Assets and Software - Establish, enforce, and continuously validate secure configuration baselines for CI/CD platforms, runners, registries, and deployment systems, including drift detection and corrective actions. |
| CIS v8 | 5.x | Account Management – Govern CI/CD and platform accounts, service identities, and role scopes. |
| CIS v8 | 8.x | Audit Log Management – Collect, retain, and protect logs for build, deploy, and application events. |
| CIS v8 | 16.x | Application Software Security – Integrate SAST/DAST/IAST and SDLC governance with non-bypassable gates. |

| Framework | Control ID | Control Name / Description |
|---|---|---|
| | | |
| OWASP ASVS | V2.x | Authentication Architecture – Token lifecycle, session management, and credential handling in services/APIs. |
| OWASP ASVS | V4.x | Access Control – Enforce and verify authorization decisions (incl. object/function-level). |
| OWASP ASVS | V14.x | Configuration & Operations – Secure configuration, environment parity, and logging/monitoring verification. |
| OWASP API Top 10 | API1 | Broken Object Level Authorization (BOLA) – Prevent and test for object-level authorization flaws. |
| OWASP API Top 10 | API2 | Broken Authentication – Prevent and test for auth weaknesses and token misuse. |
| OWASP API Top 10 | API5 | Broken Function Level Authorization (BFLA) – Enforce and test function-level authorization. |

***NOTE**: NIST and ISO are Foundational Standards in §8. Use CSA/CIS/OWASP here in §9 for control implementation. Adversary-technique mapping (e.g., ATT&CK) belongs in §12 and sub-standards' test plans.*

***NOTE:** Use of External Control Frameworks.*

*ISAUnited maps to external control frameworks to provide alignment and traceability, but does not speak on behalf of those organizations. Practitioners shall consult and follow the official practices, recommendations, and implementation guidance of the Center for Internet Security (CIS), the Cloud Security Alliance (CSA), and the Open Worldwide Application Security Project (OWASP) when applying controls. Always verify control identifiers, scope, and version currency against the publishers' latest materials. Where wording differs, use the framework's official documentation while maintaining consistency with ISAUnited security invariants and this standard's requirements.*

### Additional References

As the DevSecOps domain matures or as additional authoritative frameworks become relevant, authors and contributors may include supplementary CSA/CIS/OWASP controls where sub-standards directly enforce them.

### Sub-Standard Expectations

Sub-standards developed under the DevSecOps & Secure SDLC Engineering Parent Standard are required to:

- Select and enforce explicit controls relevant to their scope (e.g., artifact signing & verify-on-pull, SCA/KEV blocking, API authorization tests).
- Provide detailed mappings of these controls to §6 outputs and to defined verification/validation criteria in §12.
- Justify and document any deviation from the Parent-level control families with compensating controls and a sunset date, ensuring transparency and defensibility.

### Evidence Pack

Evidence Must be collected for Section 9 control mappings in EP-10.4 (Control Mappings). Each control referenced in Table J-4 Must have a dated mapping record that identifies the control identifier, the related §6 output, the enforcement mechanism, and the associated verification and validation activity in §12.

Minimum evidence expectations for EP-10.4 include:

- A Controls-to-Outputs mapping sheet that links each CSA, CIS, and OWASP control to one or more §6 outputs and to the responsible owner.
- A control enforcement record that identifies where the control is implemented, including policy-as-code rules, pipeline gates, admission enforcement points, identity constraints, or logging and telemetry configurations.
- An exception register documenting any deviation from a mapped control, including compensating controls, a sunset date, approval record, and the Evidence Pack cross-reference for proof.
- Cross-links from EP-10.4 to implementation artifacts in EP-10.2 (Technical Specifications) and to test results in EP-10.5 (Verification and Validation) that demonstrate pass or fail outcomes for the mapped control.

Entries in EP-10.4 Must remain current. Any change to a gate, policy, signing rule, or promotion pathway that affects a mapped control Must update the mapping in the same change record and include an updated evidence reference.

**Practitioner Guidance**:

- Build a Controls-to-Outputs map for each service. Each control in Table J-4 should map to the related §6 output, the applicable §12 test reference, and an Evidence Pack reference showing pass or fail.
- Keep mappings current. Update the map in the same pull request that changes a gate, policy, or runtime control, and attach proof artifacts to the Evidence Pack.
- Prefer enforceable controls. If a control cannot be expressed as code or measured, replace it with an equivalent control that can be enforced and verified.

## Section 10. Engineering Discipline

This section outlines the architectural thinking, rigorous engineering processes, and disciplined operational behaviors required to implement the DevSecOps and Secure SDLC Engineering Parent Standard (ISAU-DS-DSS-1000) effectively. ISAUnited's Defensible Standards are not compliance checklists. They are engineered frameworks that emphasize integrity, precision, and sustained operational effectiveness across planning, build and test, release and deploy, and runtime operations.

### 10.1 Purpose and Function

**Purpose**.
Establish a repeatable, evidence-producing engineering system that integrates systems thinking, lifecycle control, supply chain assurance, adversary-aware design, and measurable security outcomes across CI/CD, and runtime promotion.

**Function in D10S**.
Parent Standards define domain-level engineering invariants and expectations. Sub-standards operationalize those invariants through security-as-code, policy-as-code, test specifications, admission controls, and evidence artifacts embedded in software delivery pipelines and runtime promotion paths.

### 10.2 Systems Thinking

**Goal**: Make the DevSecOps system legible end-to-end, including boundaries, data flows, trust relationships, identities, promotion paths, supply chain dependencies, and pipeline invariants, so controls bind where pipeline and artifact integrity risks actually occur.

### 10.2.1 System Definition and Boundaries

- Declare CI/CD, repository, registry, build system, and deployment system boundaries, including VCS, runner fleet, artifact registry, signing and attestation services, policy controller, orchestrator, and runtime environment.
- Define trust zones for source to build to artifact to registry to deploy to runtime pathways, including identity issuance, attestation signing, and verify-on-pull enforcement points.
- Establish boundary invariants, for example: no unsigned commits on protected branches, no unsigned or un-attested artifacts admitted to deploy, no fail-open pipeline gates, short-lived tokens for CI/CD identity surfaces, and immutable audit logs for build and deploy actions.

### 10.2.2 Interfaces and DevSecOps Contracts

Engineering teams Must maintain Interface Control Documents (ICDs) for source control, pipeline orchestration, registry interactions, attestation workflows, admission control checks, and evidence exchange.

**For each interface, specify**: identity model (human, service, workload); privileges and constraints; data schemas (SBOM format, provenance schema, signing metadata, pipeline event logs); latency SLOs; promotion invariants (verify-on-pull must pass); fail-closed behaviors; mandatory audit fields (build_id, commit_id, signer_id, attestation_id, evidence_pack_reference)

### 10.2.3 Dependencies and Emergent Behavior

- Map shared services that influence DevSecOps integrity, including key management, secrets platforms, signing and attestation services, package repositories, orchestrator APIs, runner infrastructure, evidence stores, and logging and telemetry stacks.
- Identify emergent risk from composition, for example: build system compromise paired with unsigned commits produces untrustworthy artifacts, registry tampering paired with missing provenance creates insertion paths, runner reuse paired with long-lived tokens creates identity hijack paths, drift paired with weakened policy enforcement enables bypass of trusted stages, and schema mismatch paired with partial SBOMs creates false perceptions of supply chain integrity.

#### 10.2.4 Failure Modes and Safeguards

- For each critical path from commit to deploy, document likely failures, including missing or invalid signatures, broken attestation chains, disabled pipeline gates, policy controller outages, drift between staging and production, registry unavailability, compromised runners, and expired workload certificates.
- Design safeguards include negative tests for unsigned artifacts or missing SBOMs, fail-closed pipeline behavior, registry read-only fallback, where applicable, policy regression tests, ephemeral runners with teardown guarantees, parity validation across environments, and rollback automation.

**Required Artifacts (minimum):** DevSecOps system diagram with trust boundaries; source to build to artifact to deploy data-flow map; ICD set, pipeline, and registry invariants register

### 10.3 Critical Thinking

**Goal**: Eliminate assumption-based pipeline and supply chain configurations by replacing them with explicit, reviewed, evidence-based reasoning that withstands adversarial pressure, operational constraints, and audit scrutiny.

#### 10.3.1 Decision Discipline

- Maintain Architecture Decision Records (ADRs): problem to options to constraints/assumptions to trade-offs to decision to invariants to test/evidence plan (who / when / how measured).
- Require ADR linkage to relevant ISAU-RPs (01–20), NIST/ISO clauses, and Evidence Pack IDs.

#### 10.3.2 Engineering Prompts

Prompts engineers should answer explicitly:

- **Boundaries**: Which supply chain or identity boundaries exist and why? Where must trust be re-established, including commit signing, attestation signing, and verify-on-pull?
- **Interfaces**: What invariants must always hold, including signature verification, SBOM completeness, and schema adherence? How are these invariants validated during every release?
- **Adversary Pressure**: Which ATT and CK techniques apply to CI/CD, registries, or orchestrators? Where could adversaries insert artifacts or hijack identity?
- **Evidence**: What objective evidence proves integrity, including attestations, SBOMs, build logs, and verify-on-pull outputs? How is parity between staging and production validated?

- **Failure**: Does failure default to deny or permit? When deployed, does rollback execute predictably?

**Required Artifacts (minimum)**: ADRs; assumptions and constraints log; evidence plan per architecture or pipeline decision

### 10.4 Domain-Wide Engineering Expectations

**Secure System Design**
Engineering teams Must define and validate boundaries for source control, build systems, runners, registries, orchestrators, and runtime workloads using the §10.2 artifacts and engineering reviews.

**Implementation Philosophy, Built-in, not Bolted-on**
Engineering teams Must integrate signing, attestation, policy as code, verify-on-pull, and gate enforcement at design time and express enforcement mechanisms as code rather than as post-release compensating patches.

**Lifecycle Integration**
Engineering teams Must integrate DevSecOps checks across merge, pre-deploy, deploy, and post-deploy phases and update ADRs and Evidence Pack references whenever a gating policy, signing policy, or promotion boundary changes.

**Verification Rigor**
Engineering teams Must combine automated checks with targeted probes, including negative tests for unsigned artifacts, revoked key scenarios, and drift injections, and maintain continuous validation of gates, supply chain integrity, and environment parity.

**Operational Discipline**
Engineering teams Must maintain operational runbooks for key compromise, attestation-chain failure, registry tampering, and rollback procedures and record operational actions and outcomes in the Evidence Pack structure.

### 10.5 Engineering Implementation Expectations

- Policies and controls as code. Store policy as code, infrastructure as code, signing policies, admission rules, and provenance validation logic in version control with signed commits and peer review enforcement.
- Structured pipeline promotion. Implement CI/CD pipelines with artifact signing, attestation, SBOM generation, verify-on-pull enforcement, negative tests, and rollback definitions.

- Explicit promotion path mapping. Document commit to build an artifact to the registry to deploy chains and maintain dashboards for SBOM completeness, signature validity, KEV blocking, and runner isolation.
- Automated testing and negative validation. Run supply chain negative tests, including unsigned artifacts, malformed SBOM submissions, or revoked signing keys, on a defined cadence and prior to high-risk promotions.
- Traceable architecture decisions. Link policy changes to ADR, test, and Evidence Pack references.

**Required Artifacts (minimum):** policy and infrastructure repositories; enforcement and test gate definitions; trust-boundary ICDs; signature and attestation validation reports; automated test logs; evidence ledger referenced by §12

### 10.6 Sub-Standard Alignment (Inheritance Rules)

Sub-Standards Must operationalize this engineering discipline with DevSecOps-specific detail and maintain traceability to §6 outputs, §7 principles, §8 foundational standards, §9 controls, and §12 verification and validation activities.

Example Sub-Standard Engineering Applications:

- ISAU-DS-DSS-1010 (CI/CD Runner and Identity Security)
- ISAU-DS-DSS-1020 (Policy as Code and Admission Control)
- ISAU-DS-DSS-1030 (Software Supply Chain Integrity and Provenance)
- ISAU-DS-DSS-1040 (Application and API Security Testing)
- ISAU-DS-DSS-1050 (Environment Parity and Deployment Safety)

### 10.7 Evidence and V&V (What Proves It Works)

Engineering teams Must maintain Evidence Pack references for DevSecOps proof artifacts using the EP-10 structure:

- Design Evidence: architecture diagrams, ICDs, and trust boundaries; invariants register; ADRs; supply chain maps
- Build Evidence: signing logs and attestation bundles; SBOM and provenance artifacts; CI/CD test outputs; pipeline gate enforcement results
- Operate Evidence: environment parity scans; verify-on-pull logs; drift detection results; token lifecycle audit trails; base image lifecycle reports
- Challenge Evidence: negative test runs, including unsigned artifacts, incomplete SBOMs, or revoked keys; drift injection results; rollback drills; adversary simulation scenarios for delivery threats; compromise simulations

Each control defines the objective pass/fail criteria, test frequency, responsible owner, and retention period.

## Section 11. Associate Sub-Standards Mapping

### Purpose of Sub-Standards

ISAUnited Defensible Sub-Standards are detailed, domain-specific extensions of the DevSecOps and Secure SDLC Engineering Parent Standard (ISAU-DS-DSS-1000).

Each Sub-Standard delivers:

- Granular technical guidance tailored to specialized DevSecOps domains.
- Actionable engineering strategies that convert architectural intent into enforceable pipeline and runtime controls.
- Defined verification and validation methodologies ensuring outputs are measurable, testable, and auditable.
- Alignment with the Parent Standard's § 6 technical outputs, § 7 cybersecurity principles, and Table J-3 foundational standards.

Sub-Standards transform high-level DevSecOps direction into the technical precision required for consistent pipeline hardening, continuous validation, secure automation, integrity assurance, and defensible release processes across all delivery environments. This structure enables requirements to flow down from the Parent Standard into Sub-Standard requirements, tests, and Evidence Pack references, ensuring implementations remain consistent, traceable, and auditable.

### Scope and Focus of DSS Sub-Standards

**Secure CI/CD Pipeline Architecture and Runner Isolation**
Example – ISAU-DS-DSS-1010: CI/CD Architecture, Runners, and Execution Controls

- Defines secure runner classes (ephemeral, isolated, scoped) and teardown guarantees.
- Enforces non-bypassable gating functions for SAST, SCA, IaC, and supply chain checks.
- Establishes identity boundaries, token constraints, and required attestation flows.
- Validates runner and pipeline components through negative tests and audit logs.

**Policy-as-Code and IaC Security**
Example – ISAU-DS-DSS-1020: Policy Bundles, Enforcement Engines, and IaC Guardrails

- Defines policy bundles for network, identity, cryptography, logging, and container posture.
- Requires deterministic PaC evaluation in CI/CD, and admission paths.
- Enforces drift detection and automatic remediation.
- Integrates validation of policy changes through regression testing and Evidence Pack entries.

**Automated Security Testing and Release Gates**
Example – ISAU-DS-DSS-1030: Automated Test Gates and Release Quality Controls
- Establishes mandatory SAST, DAST, IAST, SCA, and container-scan thresholds.
- Defines coverage requirements for changed files, endpoints, and service APIs.
- Enforces fail-closed merge and release rules.
- Integrates health SLOs and post-deploy test validation into promotion workflows.

**Software Supply Chain Integrity and Provenance**
Example – ISAU-DS-DSS-1040: SBOM, Signing, and Provenance Validation
- Requires SBOM generation and attestation for 100 percent of artifacts.
- Defines signing policy, attestation formats, transparency requirements, and verify-on-pull behavior.
- Enforces deterministic builds and rebuild-parity checks.
- Integrates key-rotation drills and tampering simulations.

**Secrets Management and Credential Hygiene**
Example – ISAU-DS-DSS-1050: Secrets Governance and Identity-Bound Access Controls
- Prohibits static or embedded secrets in source or images.
- Issues short-lived, scoped pipeline and workload credentials.
- Requires rotation within 15 minutes of compromise.
- Enforces full auditability of secret access and token usage.

**Reproducible Builds and Release Governance**
Example – ISAU-DS-DSS-1060: Deterministic Build Controls and Promotion Path Rigor
- Defines reproducible build requirements and no-latest-tag constraints.
- Requires signed release manifests and structured promotion chains.
- Enforces automated rollback paths and release gating invariants.
- Validates build reproducibility through parity diff tests.

**Pipeline Telemetry, Evidence Production, and Forensic Readiness**
Example – ISAU-DS-DSS-1070: Evidence Generation, Retention, and Forensic Controls
- Standardizes logging schemas for CI/CD, registry, and promotion pathways.
- Requires immutable Evidence Packs per release.
- Enforces retention, retrieval, and auditability of all evidence artifacts.
- Integrates detection and investigation workflows for pipeline-level incidents.

**Continuous Verification of Pipelines (Chaos, Resilience, and Negative Testing)**
Example – ISAU-DS-DSS-1080: Pipeline Chaos Engineering and Resilience Testing

- Introduces negative tests (unsigned artifacts, revoked keys, malformed SBOM).
- Simulates runner compromise, registry failure, gate outages, and policy-controller misconfigurations.
- Validates rollback, fail-closed behavior, and automated recovery.
- Records evidence of resilience tests in the Evidence Pack for audit.

**Table J-5. Example Future Sub-Standards:**

| Sub-Standard ID | Sub-Standard Name | Focus Area |
|---|---|---|
| ISAU-DS-DSS-1010 | Secure CI/CD Pipeline Architecture & Runner Isolation | Runner Isolation & Pipeline Design |
| ISAU-DS-DSS-1020 | Policy-as-Code & IaC Security | Policy Bundles, IaC, Admission Control |
| ISAU-DS-DSS-1030 | Automated Security Testing & Release Gates | Testing Gates & Quality Thresholds |
| ISAU-DS-DSS-1040 | Software Supply Chain Integrity (SBOM, Signing, Provenance) | Signing, Attestations, SBOM |
| ISAU-DS-DSS-1050 | Secrets Management & Credential Hygiene | Secrets, Short-Lived Credentials |
| ISAU-DS-DSS-1060 | Reproducible Builds & Release Governance | Determinism, Manifests, Promotion |
| ISAU-DS-DSS-1070 | Pipeline Telemetry, Evidence Production & Forensic Readiness | Evidence & Retention |
| ISAU-DS-DSS-1080 | Continuous Verification of Pipelines (Chaos/Resilience) | Pipeline Resilience & Drills |

***Note on Traceability:*** *When adopting or extending any Sub-Standard listed in Table J-5, practitioners should maintain a traceability map that links each Sub-Standard requirement to the Parent Standard's §6 technical outputs, the selected ISAUnited Core Principles in §7, the foundational standards in §8, and the control mappings in §9. Store the traceability map in EP-10.4 and cross-link validation artifacts in EP-10.5 so reviewers can verify implementation, validation, and evidence through §12 activities. Maintaining clear traceability reinforces defensibility, supports audit readiness, and ensures alignment with the architectural invariants defined by ISAU-DS-DSS-1000.*

### Development and Approval Process

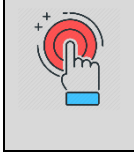
ISAUnited uses an open, peer-driven annual process to propose, review, and publish sub-standards:

- Open Season Submission – Proposals must cite which §6 outputs and §7 principles they extend, plus NIST/ISO clauses from §8 and control mappings from §9.
- Technical Peer Review – Evaluate engineering rigor, testability, scope clarity, and cross-domain consistency (IAM, CEK/CKM, MDR, Cloud/Compute).
- Approval & Publication – Assign identifier, version, and publish as an actionable extension of ISAU-DS-DSS-1000.

### Sub-Standard Deliverables (normative)

Each sub-standard Must include:

- Inputs (Requirements): Preconditions (from §5) depend on.
- Outputs (Specifications): Concrete factory behaviors and thresholds (SLOs) tied to §6.
- Verification/Validation: Named tests and acceptance criteria tied to §12 (e.g., unsigned artifact rejection, deterministic rebuild parity, key-rotation drills).
- Evidence: Artifact list and storage location (Evidence Pack ID), including logs, SBOMs, signatures, attestations, and approvals.
- Standards Mapping: DSSR-ID/Spec to NIST/ISO clause (from §8) to Controls (from §9) to Test-ID to Evidence Pack ID.
- Interfaces: Explicit delineation of what is enforced in delivery (this standard) vs. runtime platforms (Cloud/Compute, MDR) and crypto parameters (CEK/CKM).



**Practitioner Guidance:**

Sub-Standards Must remain vendor-neutral, measurable, and enforceable through code-based controls. Favor organization-wide templates, mandatory pipeline jobs,

| | |
|---|---|
| | signing and attestation policies, and admission-control rules over descriptive or guidance-only text. Negative tests such as unsigned artifact attempts, revoked-key scenarios, malformed SBOM submissions, or runner-reuse detections Must be treated as release blockers unless a time-bounded exception with compensating controls has been explicitly approved and recorded in the Evidence Pack. All Sub-Standards Must define clear inheritance from the Parent Standard, trace their controls to §6 outputs and §12 verification tests, and update Evidence Pack references whenever a policy, gate, or interface changes. Store mappings in EP-10.4 and cross-link test outcomes in EP-10.5. |

# Section 12. Verification and Validation (Tests)

This section defines the structured evaluation methods necessary to ensure that implemented controls, architecture, and engineering decisions align with this Parent Standard. It mandates measurable, repeatable testing to confirm that solutions are technically defensible and adhere to ISAUnited's engineering discipline.

**Verification** confirms that the system has been implemented in accordance with the Requirements (Inputs) in §5 and the Technical Specifications (Outputs) in §6.

**Validation** confirms the system performs effectively under real-world operating and adversarial conditions.

**Core Verification Activities**

- Confirm that all §6 outputs are implemented in the target environment(s).
- Review and validate configuration baselines against engineering and security benchmarks.
- Verify interoperability and integration points so new vulnerabilities are not introduced.
- Conduct peer review of architecture diagrams, ADRs, pipeline definitions, and control mappings.

**Core Validation Activities**

- Perform adversarial testing (e.g., BAS/ATT&CK scenarios, targeted pen testing, red teaming) to measure defensive effectiveness.
- Validate security posture using automated and manual methods against relevant threat models.
- Test operational resilience, including canary/rollback, recovery, and incident response capabilities.

- Measure performance of controls against defined SLOs (e.g., High/Critical at merge = 0, KEV=0 at release, mTLS ≥ 98%, rollback < 5 min).

**Required Deliverables**

1. Test Plans & Procedures – Scope, data sets, tools, and methods for verification and validation.
2. Validation Reports – Results with pass/fail status and residual risk ranking.
3. Evidence Artifacts – Logs, screenshots, signatures/attestations, SBOMs, scan outputs, TLS/mTLS captures, canary/rollback logs.
4. Corrective Action Plans – Remediation steps with owners and target dates before acceptance.

**Common Pitfalls to Avoid**

- Treating verification as a documentation exercise. Verification is not a checklist review. It must produce dated artifacts that prove that gates executed, that enforcement occurred, and that failure conditions block promotion.
- Running validation only after major incidents. Validation must be scheduled and repeatable. Adversary simulation and rollback drills should occur on a defined cadence for in-scope services.
- Allowing tests to pass without proving fail-closed behavior. A passing report is insufficient if it does not prove denial for negative cases. Include negative tests such as unsigned artifacts, revoked signing keys, missing SBOMs, and policy violations.
- Failing to validate deploy-time enforcement. Many organizations validate scanning in CI, but never validate admission enforcement. Verify-on-pull and attestation checks must be exercised during a controlled test deploy.
- Confusing environment parity with functional similarity. Parity must include authorization posture, transport controls, egress restrictions, and alignment of logging schema. If staging lacks these controls, validation results are not predictive.
- Accepting exceptions without time bounds or compensating controls. Any bypass, suppression, or allowlist expansion must include a sunset date, a compensating control, and evidence of review. Track exception outcomes as SLO met or SLO not met.
- Mixing evidence across systems without traceability. Evidence must not be scattered across build logs, ticketing systems, and storage accounts. Store V&V artifacts in EP-10.5 and cross-link to EP-10.2 implementation proof.
- Failing to re-run impacted tests after changes. Any change to gate logic, signing policy, admission rules, or transport profiles must trigger a re-execution of the affected verification and validation activities and an update to the traceability matrix.

- Measuring success without clear pass or fail criteria. Every test must define objective thresholds and record outcomes in binary terms, SLO met or SLO not met, with artifact links.
- Skipping rollback drills. Rollback is a primary safety control in DevSecOps. Validate rollback triggers and completion timing under controlled conditions and retain the drill evidence.

**Table J-6. Traceability Matrix — Requirements (§5) to Verification/Validation (§12) to Related Technical Specs (§6):**

| Req ID | Requirement (summary) | Verification (build-correct) | Validation (works-right) | Related §6 Outputs |
|---|---|---|---|---|
| 5.1 | Version control with protected branches, required reviews, and signed commits | Protected branches configured; CODEOWNERS and required review enforcement present; signed-commit enforcement report available | Sample repositories show signed-commit rate at or above 95 %; unauthorized push or bypass attempts are blocked and logged | 6.1 |
| 5.2 | CI/CD platform with non-bypassable gates and rollback steps | Pipeline definitions show fail-closed gates for SAST, SCA, IaC checks, and image scanning, plus rollback definitions | Seeded policy breach fails merge or release; rollback executes successfully during a controlled test release | 6.2, 6.10 |
| 5.3 | Trusted artifact registry with signature verification, SBOM retention, and provenance or attestations | Registry configuration enforces signature verification at publish and pull; SBOM retention enabled; provenance or attestation recording enabled | Deploy blocks, unsigned or un-attested artifact; SBOM and attestation retrievable for 100 % of released artifacts | 6.2, 6.3, 6.8 |
| 5.4 | Central secrets platform issuing short-lived scoped credentials with full auditability | Secret scanners enabled in pre-commit and CI; token TTL policy configured; audit logging enabled for secret access | Seeded secret commit is blocked; token TTL enforced at 24 h or less; rotation drill meets 15 min objective | 6.4 |
| 5.5 | Application and API test capability available for gating | SAST and dynamic test policies configured; test environments wired; coverage expectations documented | Critical or High findings before production equal zero; API authorization tests prevent BOLA or BOPLA failures in staging validation | 6.5 |
| 5.6 | Policy-as-code guardrails integrated for critical violations | Policy bundles referenced in pipeline; rulesets exist for network, identity, cryptography, logging, and telemetry, and platform hardening | Critical IaC violations are blocked at merge and deploy; drift control shows | 6.2, 6.7 |

| Req ID | Requirement (summary) | Verification (build-correct) | Validation (works-right) | Related §6 Outputs |
|---|---|---|---|---|
| | | | parity checks pass in staging | |
| 5.7 | Dependency and container scanning integrated, KEV synced, base-image lifecycle enforced | SCA and image scan jobs present; KEV feed sync evidence exists; base-image lifecycle policy documented | Release blocks KEV items; base image age meets policy target; container hardening checks pass | 6.3, 6.6 |
| 5.8 | Threat modeling practice with pull request delta for architectural change | Threat model artifact exists; pull request templates include a delta field; mapped mitigations documented | Review confirms mitigations map to tests or enforcement controls; delta reviewed prior to merge for architectural PRs | 6.9 |
| 5.9 | Staging mirrors production control posture for authorization, egress, transport, and logging schemas | Staging TLS, mutual TLS, egress controls, and logging schemas match production; drift monitoring is configured | Mutual TLS coverage meets 98 % target, where in scope, canary or progressive delivery passes parity checks and rollback trigger tests | 6.7, 6.10 |
| 5.10 | Unified logging schema and tamper-evident evidence store | Schema validators configured; evidence store is write-once or tamper-evident; retention configured | Evidence Pack exists for the last release; forensic replay of gate and promotion artifacts succeeds | 6.8 |

**How to use the matrix:**

- **Plan**: Map each §5 input to at least one verification activity and one validation activity tied to §6.
- **Execute**: Attach an Evidence Pack reference and record a clear outcome, SLO met or SLO not met.
- **Maintain**: When a gate, signing policy, or transport profile changes, the mapping Must be updated in the same change record and impacted tests re-executed.

## Evidence Pack

Evidence Must be collected for Section 12 verification and validation activities in EP-10.5 (Verification and Validation). Each verification and validation activity Must produce at least one dated artifact demonstrating execution, result, and the measurement point used for acceptance. Evidence Must be version-controlled and retained according to organizational audit requirements.

Obsolete and withdrawn documents should not be used; please use replacements.

Minimum evidence expectations for EP-10.5 include:

- Test plan and procedure set covering verification and validation scope, frequency, owners, and pass or fail criteria.
- Gate enforcement proof, including blocked merge records, blocked deploy records, and negative test results for unsigned artifacts, revoked keys, or missing SBOMs.
- Provenance and attestation validation results showing acceptance for valid artifacts and denial for invalid artifacts.
- Environment parity validation results, including TLS and mutual TLS checks, egress allowlist enforcement checks, and logging schema parity verification.
- Progressive delivery and rollback drill artifacts, including trigger conditions, execution logs, and measured rollback completion time.
- Adversary simulation results were required by scope, including detection timing and response timeline.

Entries in EP-10.5 Must cross-link back to EP-10.2 for implementation proof and to Table J-6 rows for traceability.

**Practitioner Guidance**:

- Single source of truth. Store every test plan, result, and artifact under EP-10.5 and cross-link implementation artifacts in EP-10.2.
- Binary outcomes. For each row in Table J-6, record SLO met or SLO not met and link the exact artifact used for the determination. Any SLO not met Must generate a corrective action record.
- Adversary parity. For internet-exposed services, run at least one adversary simulation scenario per release and retain the alert timeline proving detection within 10 minutes and rollback within 5 minutes when triggered.
- Change discipline. Any change to a gate, signing policy, admission policy, or transport profile Must include updated verification steps in the next release plan and refreshed artifacts in EP-10.5.

**Quick Win Playbook**:

**Title**: Verification Drill for Supply Chain Admission Controls

| | |
|---|---|
| | **Objective**: Prove that deploy-time admission enforcement blocks unsigned, tampered, or unattested artifacts and that the denial events are captured as verification evidence.<br><br>**Target**: Validate that verify-on-pull, signature verification, and attestation validation function correctly in a controlled test environment (§6.2, §6.3, §6.8). Component/System: Artifact registry, signing and attestation service, CI/CD deployment path, admission controller.<br><br>**Protects**: Prevents unsigned, tampered, or unattested artifacts from entering the deployment chain and confirms enforcement prior to production releases.<br><br>**Stops and Detects**: unsigned image admission; invalid signature; missing or malformed SBOM; revoked or expired signing keys; attestation-chain failures during deploy.<br><br>**Action**: Configure verify-on-pull and attestation validation in the target environment. Introduce a controlled negative test using an unsigned artifact, a signed but modified artifact, a missing SBOM artifact, or an artifact signed with a revoked key. Attempt deployment and confirm admission denial. Capture denial logs and validation errors. Reattempt deploy with a valid artifact to confirm normal promotion behavior.<br><br>**Proof**: Artifacts stored in EP-10.5 and cross-linked to EP-10.2 include signing policy diffs, attestation bundles, denied admission logs, verify-on-pull output, SBOM completeness checks, registry audit trails, and rollback events where triggered.<br><br>**Metric**: 100 % invalid artifacts blocked at deploy; 100 % valid artifacts verified on pull; 0 successful deploys when signatures, attestations, or SBOMs are missing or malformed.<br><br>**Rollback**: Restore prior admission policy only through a time-bounded exception and retain all test artifacts and denial events as superseded evidence in EP-10.5. |

## Section 13. Implementation Guidelines

This section does not prescribe vendor-specific tactics. Parent Standards are stable, long-lived architectural foundations. Here, we define how sub-standards and delivery teams should translate the Parent's intent into operational behaviors that are testable, automatable, and auditable.

**Purpose of This Section in Sub-Standards**

Sub-standards should use Implementation Guidelines to:

- Translate architectural expectations from the Parent Standard into enforceable runtime and pipeline behaviors.
- Provide platform-agnostic practices that improve adoption, avoid failure, and align with ISAUnited's defensible design philosophy.
- Highlight common failure modes and how to prevent them with measurable gates and checks.
- Offer repeatable patterns expressed as code that enforce controls, trust models, and engineering discipline.

**Open Season Guidance for Contributors**

Contributors developing sub-standards should:

- Align all guidance with the strategic posture in this Parent Standard.
- Avoid vendor and product terms and express controls as requirements, tests, and evidence.
- Include lessons learned, including what fails, why it fails, and how the test proves it.
- Focus on repeatable engineering patterns rather than one-off guidance.
- Provide minimal standards mapping that links the specification or control to the NIST or ISO clause from §8 and the Evidence Pack reference.

**Technical Guidance**

**A. Organizing Principles**

1. Everything as code – Policies, configurations, infrastructure, pipelines, runbooks, and tests should be version-controlled, peer-reviewed, and promoted through environments with signed commits on protected branches.
2. Gated change – Every merge and deployment should pass automated, non-bypassable security gates tied to quantitative acceptance criteria (see §6 and §12).
3. Immutable, reproducible builds – Manual changes to artifacts or infrastructure after build should be prohibited, and releases should be reproducible from source with SBOMs, signatures, and attestations.
4. Least privilege and time-bounded elevation – Pipeline identities, runners, and deployers should use scoped permissions with time-bounded elevation. Break-glass paths should be exceptional and fully audited.
5. Environment parity – Staging should mirror production controls for authorization, egress, transport security, and logging schema so test results are predictive. Drift should be monitored and reconciled.

## B. Guardrails by Pipeline Stage

1. **Pre-commit and local**
   - Secrets scanning and commit signing should run locally and in CI where applicable.
   - Pre-commit hooks should run linters, unit tests, and basic SAST and infrastructure as code checks.
2. **Pull request and code review**
   - CODEOWNERS approval should be required, and a threat model delta should be recorded in the pull request template for material change.
   - SAST gates should fail on findings rated High or above, and coverage for changed files should be defined and enforced.
   - Infrastructure policy evaluation should run for network, identity, cryptography, and logging rules, and critical violations should block merge.
   - SBOM generation should occur as part of evaluation, including license and vulnerability policy checks.
3. **Build and package**
   - Builds should use pinned versions and deterministic build definitions and should not rely on latest tags or unverified remote scripts.
   - Images should use multi-stage builds, execute as non-root, drop NET_RAW, and define seccomp or AppArmor profiles.
   - Artifacts should be signed and attested prior to publish, and SBOMs should be stored with artifacts.
   - Transitive dependencies should be evaluated, and builds should fail on KEV items or crypto-policy violations.
4. **Pre-deploy and release**
   - DAST and IAST should execute against pull request or staging environments for internet-exposed services, along with API contract tests using negative and positive cases.
   - Database migrations should include guardrails and an automatic backout plan.
   - Drift detection should be integrated with change approval as code.
   - Progressive delivery should use blue-green or canary patterns with defined health SLOs and automated rollback conditions.
5. **Deploy and runtime**
   - TLS 1.3 should be enforced at edges, and mutual TLS should be used for service and administrative paths where required by architecture. Certificate rotation should align with CEK requirements where applicable.
   - Egress allowlists should be defined per workload, and build and deploy runners should be isolated with restricted outbound paths.
   - A unified logging schema should be enforced for build, deploy, and runtime events, and logs should be stored in append-only or immutable systems where available.

Obsolete and withdrawn documents should not be used; please use replacements.

- Runtime posture should include read-only root filesystem where feasible, CPU and memory limits, and prevention of privilege escalation, including CAP_SYS_ADMIN removal.
- Runtime instrumentation should be applied for critical paths where risk justifies it.

6. **Post-deploy validation and operations**
    - Continuous validation should be scheduled, including adversary simulation scenarios, staging fault injection, and DR restore drills aligned to RTO and RPO targets.
    - Security SLOs should be tracked, including High and Critical findings at merge equal to 0, KEV at release equal to 0, mutual TLS coverage at or above 98 %, and rollback completion time under 5 minutes where required.
    - Release evidence should be generated for each promotion, including configurations, SBOMs, signatures and attestations, scan reports, test results, parity checks, canary and rollback logs, and ADR links.

**C. Identity, Secrets, and Keys (normative alignment to §6)**

- Key storage should use KMS or HSM boundaries where applicable, and rotation should align with CEK requirements where required by scope.
- CI/CD identities should use dynamic, short-lived credentials. Long-lived tokens should be avoided, secrets should be scoped to job and environment, and logs should redact sensitive values.
- Secrets should not be stored in repositories or container layers. Runtime injection and full auditability should be used for access.

**D. Supply Chain Integrity**

- Builds should originate from trusted sources, registries and package repositories should be restricted, and signatures should be verified.
- Third-party artifacts should be quarantined and attested where required, and license policy should be enforced.
- Build and deploy identities should be separated, and production write privileges should not be granted to build jobs.
- Verify-on-pull should be enforced at deploy for deployable artifacts.

**E. Measurement and Acceptance**

- SBOM coverage should be 100 % of deployable artifacts, and promotions should not proceed when SBOMs are missing.
- Container base images should be updated within 30 days, KEV exposure at release should equal 0, and findings rated Critical or High should not remain open for promoted releases.
- Mutual TLS coverage should meet the defined target for in-scope service and administrative paths.

- Pull request gates should enforce SAST High equals 0 and infrastructure policy Critical equals 0, with evidence stored under the release Evidence Pack references.
- Provenance should be present and verified for released artifacts through signatures and attestations, and verify-on-pull should enforce validation at deploy.

**Common Pitfalls and the Engineered Countermeasure**

1. Pipelines treated as advisory – Use non-bypassable gates, block merges and releases on failure, and retain proof artifacts for review.
2. One-time scanning – Treat scans as gating controls with thresholds and enforce coverage for changed files.
3. Unpinned dependencies and latest images – Pin and verify inputs and refuse non-deterministic builds.
4. Containers running as root or with excessive capabilities – Enforce non-root execution and restricted capabilities through policy-as-code.
5. Missing SBOMs and signatures – Block promotion without SBOMs, signatures, attestations, and verify-on-pull enforcement.
6. Secrets in repositories or broad CI variables – Block on detection, use short-lived scoped credentials, and audit access.
7. Open egress and shared runners – Isolate runners, restrict outbound access, and enforce allowlists per workload.
8. Drift and hot fixes outside code – Detect and reconcile drift, avoid manual infrastructure changes, and record ADRs for material changes.
9. Canary without guardrails – Define health SLOs and rollback triggers and validate rollback behavior on a defined cadence.
10. Weak crypto, expired certificates, or missing mutual TLS – Enforce CEK-aligned transport profiles where applicable and measure mutual TLS coverage.
11. Skipping threat model deltas – Require pull request deltas and mapped mitigations tied to tests for material change.
12. Green builds created by suppression – Alert on disabled rules and excessive suppressions and require review with a sunset date.
13. Test data misuse – Mask or tokenize data and avoid live sensitive data in lower environments.
14. License compliance blind spots – Enforce license policy as part of SBOM gates.
15. No rollback plan – Require automated rollback definitions and prove rollback behavior prior to production promotion.
16. Log noise and schema drift – Validate schemas at ingest and alert on missing required fields.
17. Overbroad break-glass – Require dual control, short TTL, full audit, and periodic review of use.
18. Ignoring egress controls – Detect unexpected egress and block at policy enforcement points.

19. Credential mixing across environments – Separate principals and secrets per environment and validate separation through policy.
20. No evidence – Each release should include Evidence Pack references linking §5 prerequisites, §6 outputs, and §12 verification artifacts.

**Practitioner Guidance:**

- Map at the clause level only. For each §6 output, add a mapping row that lists the specification identifier, the NIST or ISO clause, how enforcement is implemented, and the Evidence Pack reference.
- Keep mappings current. When a control or policy changes, update the NIST or ISO citation in the same change record and store the diff under the Evidence Pack references.
- Multi-regime environments. Where multiple clauses could apply, adopt the strictest applicable requirement and record the rationale once in the mapping sheet.
- Scope discipline. Reserve CSA CCM, CIS Controls, and OWASP for Section 9 and do not list them as foundational standards in Section 8.

**Quick Win Playbook**:

**Title**: Pipeline Negative Testing to Enforce Fail-Closed Behavior

**Objective**: Prove that pipeline gates and deploy-time admission checks fail closed for invalid artifacts and that denial evidence is captured for audit.

**Target**: Introduce a controlled negative test to confirm that non-bypassable pipeline gates and verify-on-pull protections function consistently across environments (§6.2, §6.3, §6.8).

**Component/System**: CI pipeline, artifact signing service, registry, admission controller, policy-as-code engine.

**Protects**: Prevents supply chain insertion by ensuring artifacts lacking signatures, attestations, or SBOMs cannot enter promotion paths.

**Stops and Detects**: unsigned artifacts; missing SBOM; tampered signatures; revoked signing keys; bypass of attestation or verify-on-pull policies.

**Action**: Inject an intentionally invalid artifact into a non-production promotion path and execute promotion. Confirm fail-closed behavior at merge and deploy. Capture denial outputs from CI, registry, admission controller, and verify-on-pull. Retest with

| | |
|---|---|
| | a valid artifact to confirm expected promotion behavior.<br><br>**Proof**: Artifacts stored in EP-10.2 and cross-linked to EP-10.5 include pipeline failure logs, verify-on-pull denial outputs, policy-as-code evaluation results, and attestation validation errors.<br><br>**Metric**: 100 % negative-test artifacts blocked at merge and deploy; 0 bypass events; 100 % valid artifacts pass signature and provenance checks.<br><br>**Rollback**: Revert policy changes only through a time-bounded exception and retain negative-test evidence as superseded artifacts under EP-10.5. |

# Appendicies

## Appendix A: Engineering Traceability Matrix (ETM)

This Engineering Traceability Matrix (ETM) links the DevSecOps and Secure SDLC Engineering Parent Standard requirements to measurable technical specifications, cybersecurity core principles, control mappings, and Verification and Validation activities. It provides practitioners with a single view of what must exist, what must be implemented, how it is tested, and how evidence is organized. This ETM also supports flow-downs by showing how Parent Standard requirements translate into enforceable outputs and testable acceptance evidence.

Evidence Pack alignment: Evidence supporting this ETM is organized using the five EP-10 locations. For each row, primary acceptance evidence is captured in EP-10.5 (Verification and Validation results), with supporting artifacts referenced from EP-10.1 (readiness), EP-10.2 (implementation), EP-10.3 (foundational standards mapping), and EP-10.4 (control mappings).

| Req ID | Requirement (Inputs) (§5) | Technical Specifications (Outputs) (§6) | Core Principles (§7) | Control Mappings (§9) | Verification (Build Correct) (§12) | Validation (Works Right) (§12) | EP References |
|---|---|---|---|---|---|---|---|
| 5.1 | Version control and branch protection | 6.1 Everything as code governance | RP-12 Security as Code; RP-03 Complete Mediation; RP-15 Evidence Production | CSA CCM CCC-03; CIS v8 4.x | Protected branches and required reviews configured. Commit signing enforcement enabled. ADR linkage required for material changes. | Unauthorized push or bypass attempts are blocked and logged. Signed-commit rate at or above 95 % validated over the rolling window. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.4 supporting |
| 5.2 | CI/CD platform with non-bypassable security gates | 6.2 Secure pipeline gates; 6.10 Post-deploy validation and rollback | RP-03 Complete Mediation; RP-10 Secure Defaults; RP-14 Resilience & Recovery | CSA CCM TVM-01; CIS v8 16.x; CIS v8 4.x | Pipeline definitions show fail-closed execution for SAST, dependency scanning, infrastructure policy checks, and image scanning. Rollback definition exists as code. | Seeded gate failures block merge or promotion. Rollback executes under controlled conditions and meets defined timing objectives where applicable. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.3 supporting; EP-10.4 supporting |
| 5.3 | Trusted artifact registries and provenance capability | 6.2 Secure pipeline gates; 6.3 Supply chain integrity and build | RP-02 Zero Trust; RP-19 Protect Integrity; | CSA CCM DCS-03; CIS v8 2.x | Registry enforces signature verification at publish and pull. | Deploy rejects unsigned or unattested artifacts. Verify- | EP-10.5 primary; EP-10.1 |

| Req ID | Requirement (Inputs) (§5) | Technical Specifications (Outputs) (§6) | Core Principles (§7) | Control Mappings (§9) | Verification (Build Correct) (§12) | Validation (Works Right) (§12) | EP References |
|---|---|---|---|---|---|---|---|
| | | hygiene; 6.8 Observability and evidence | RP-15 Evidence Production | | SBOM retention and attestation capture are enabled for release artifacts. | on-pull enforcement proves artifact integrity under promotion. SBOMs and attestations are retrievable for 100% of released artifacts. | supporting; EP-10.2 supporting; EP-10.3 supporting; EP-10.4 supporting |
| 5.4 | Secrets and pipeline identity management | 6.4 Secrets and CI/CD identity | RP-01 Least Privilege; RP-02 Zero Trust; RP-15 Evidence Production | CSA CCM IAM-05; CIS v8 5.x | Pre-commit and CI secret scanning enabled. Token TTL policies configured. Audit logging is enabled for secret access. | Seeded secret commit blocked. Token TTL conforms to 24 h or less where required. Rotation drill meets target timing after the compromise trigger. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.4 supporting |
| 5.5 | Application and API test capability available for release enforcement | 6.5 Application security testing as release enforcement | RP-05 Secure by Design; RP-04 Defense in Depth; RP-16 Make Compromise Detection Easier | CIS v8 16.x; OWASP ASVS V2.x; OWASP ASVS V4.x; OWASP API Top 10 API1, API2, API5 | Test environments and policies are configured to run required test suites for in-scope services. Gate wiring established for release decisions. | For in-scope internet-exposed services, findings rated Critical or High do not pass promotion. API authorization tests detect BOLA and BOPLA failure modes prior to deploy where applicable. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.4 supporting |
| 5.6 | Policy-as-code and infrastructure guardrails | 6.2 Secure pipeline gates; 6.7 Environment parity and transport controls | RP-12 Security as Code; RP-03 Complete Mediation; RP-10 Secure Defaults | CIS v8 4.x; CSA CCM CCC-03 | Policy bundles referenced in the pipeline. Critical violations block merge and promotion. Drift detection tooling configured for declared scope. | Drift injections or misconfiguration attempts are detected and handled according to the defined response. Parity checks validate that the staging control posture is predictive of promotion. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.3 supporting; EP-10.4 supporting |
| 5.7 | Dependency and container | 6.3 Supply chain integrity and build hygiene; 6.6 | RP-06 Minimize Attack | CSA CCM TVM-01; CIS v8 4.x | Dependency and image scanning steps are present. | KEV items block promotion. Base image age | EP-10.5 primary; |

| Req ID | Requirement (Inputs) (§5) | Technical Specifications (Outputs) (§6) | Core Principles (§7) | Control Mappings (§9) | Verification (Build Correct) (§12) | Validation (Works Right) (§12) | EP References |
|---|---|---|---|---|---|---|---|
| | security readiness | Dependency and container security | Surface; RP-19 Protect Integrity; RP-10 Secure Defaults | | KEV synchronization evidence exists. Base image lifecycle policy documented and applied. | meets policy target. Container posture checks validate non-root execution and restricted capabilities where in scope. | EP-10.1 supporting; EP-10.2 supporting; EP-10.3 supporting; EP-10.4 supporting |
| 5.8 | Threat modeling practice and pull request delta | 6.9 Threat modeling and pull request delta | RP-05 Secure by Design; RP-13 Plan Security Readiness; RP-15 Evidence Production | CSA CCM CCC-03 | A threat model artifact exists for in-scope services. Pull request templates include a delta requirement for architectural change. | The review confirms that the mitigations map to tests or enforcement controls. Architectural pull requests do not merge without a delta review where required by scope. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.4 supporting |
| 5.9 | Environmental parity and transport controls | 6.7 Environment parity and transport controls; 6.10 Post-deploy validation and rollback | RP-02 Zero Trust; RP-14 Resilience & Recovery; RP-16 Make Compromise Detection Easier | OWASP ASVS V14.x; CIS v8 4.x; CIS v8 8.x | Staging transport, egress controls, and logging schema match production for the declared scope. Drift monitoring configured. | Mutual TLS coverage meets the target where required. Parity checks remain stable over time. Canary or controlled promotion succeeds, and rollback triggers behave as designed. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.3 supporting; EP-10.4 supporting |
| 5.10 | Logging schema and evidence store | 6.8 Observability and evidence | RP-15 Evidence Production; RP-16 Make Compromise Detection Easier; RP-19 Protect Integrity | CSA CCM SEF-01/SEF-02; CIS v8 8.x | Unified logging schema defined and enforced. Evidence store integrity properties configured. Evidence capture paths validated. | Forensic replay succeeds using retained artifacts. Evidence completeness is consistent for releases in scope, supporting audit and incident reconstruction. | EP-10.5 primary; EP-10.1 supporting; EP-10.2 supporting; EP-10.4 supporting |

## Appendix B: Evidence Pack Matrix

This summary matrix provides practitioners with a single, readable view of how the DevSecOps and Secure SDLC Engineering Evidence Pack repository is organized for Parent Standard adoption. Each Evidence Pack location corresponds to a core section of the annex standard, enabling consistent evidence collection and review without creating sub-standard evidence structures.

Evidence Pack alignment: EP-10 is the Evidence Pack repository for D10. Evidence is organized into five section-aligned locations. EP-10.1 captures readiness artifacts for Section 5, EP-10.2 captures implementation artifacts for Section 6, EP-10.3 preserves clause-level foundational standards mappings for Section 8, EP-10.4 maintains external control mappings for Section 9, and EP-10.5 contains Verification and Validation test evidence for Section 12. Together, these five locations provide traceability from prerequisites to implementation to proof.

| Layer | EP Identifier | Purpose | Evidence Categories Included |
|---|---|---|---|
| EP Repository | EP-10 | Evidence Pack repository for D10. Serves as the single entry point for adoption evidence and traceability across Sections 5, 6, 8, 9, and 12. | • Index and file structure overview for EP-10.1 through EP-10.5<br>• Evidence ledger listing section reference, artifact name, date, owner, and review status<br>• Traceability snapshot linking inputs to outputs to tests and Evidence Pack locations<br>• Change log capturing updates to evidence sets and review outcomes |
| Requirements | EP-10.1 | Captures readiness and prerequisite evidence for Section 5 (Inputs). Demonstrates that baseline capability exists before implementation work begins. | • Branch protection settings, required reviews, and commit signing enforcement proof<br>• CI and CD platform readiness artifacts showing gate capability and rollback readiness<br>• Registry configuration baseline supporting signature verification and artifact retention expectations<br>• Secrets platform readiness artifacts supporting short-lived |

| Layer | EP Identifier | Purpose | Evidence Categories Included |
|---|---|---|---|
| | | | credential issuance and audit logging<br>• Policy-as-code readiness artifacts for infrastructure guardrails and drift enforcement<br>• Environment parity readiness artifacts for staging and production posture comparison<br>• Logging schema and evidence store readiness artifacts supporting retention and integrity expectations |
| Technical Specifications | EP-10.2 | Captures implementation evidence for Section 6 (Outputs). Demonstrates controls are built, configured, and enforced as engineered behaviors. | • Gate execution logs and promotion outcomes for fail-closed behavior<br>• SBOM artifacts for deployable releases and retention evidence<br>• Signatures, attestation bundles, and verification outputs<br>• Verify-on-pull enforcement logs and admission denial records for invalid artifacts<br>• Runner isolation and restricted egress validation outputs<br>• Secrets scanning outputs and identity issuance and rotation artifacts<br>• Environment parity and transport validation artifacts<br>• Rollback definitions as code and release decision records |
| Foundational Standards | EP-10.3 | Captures Section 8 alignment to the adopted NIST and ISO baselines. Provides clause-level mapping for design, implementation, and validation reviews. | • Clause-level mapping sheet linking §6 outputs to NIST and ISO references<br>• Standards selection rationale aligned to DevSecOps scope areas such as supply chain assurance and secure delivery practices<br>• Divergence notes and compensating control statements when applicable<br>• Mapping change history with dates and the responsible owner |

| Layer | EP Identifier | Purpose | Evidence Categories Included |
|---|---|---|---|
| | | | • Cross-links to implementation evidence in EP-10.2 and test evidence in EP-10.5 |
| Control Mappings | EP-10.4 | Captures Section 9 mappings to external control frameworks. Shows how DevSecOps outputs align to widely used assurance catalogs without treating them as foundational baselines. | • Control mapping sheet linking each external control to related §6 outputs and §7 principles<br>• Framework version tracking and update history<br>• Equivalence notes to prevent duplicate mappings across frameworks<br>• Exceptions and compensating measures when a control mapping is not applicable in the declared scope<br>• Cross-links to EP-10.2 implementation artifacts and EP-10.5 validation artifacts |
| Verification and Validation | EP-10.5 | Captures Section 12 test evidence and acceptance records. Demonstrates build-correct verification and works-right validation with pass or fail outcomes and remediation linkage. | • Test plans and procedures with scope, prerequisites, and pass or fail criteria<br>• Traceability ledger mapping Table J-6 rows to test references and artifact paths<br>• Verification artifacts such as gate outputs, configuration snapshots, and enforcement proofs<br>• Validation artifacts such as negative tests for unsigned artifacts, revoked keys, and missing SBOMs<br>• Environment parity validation outputs and rollback drill evidence<br>• SLO snapshots supporting acceptance decisions and corrective action plans with re-test results<br>• Change references linking tests to the configuration or policy change that triggered validation |

**Adoption References**

*NOTE: ISAUnited Charter Adoption of External Organizations.*

*ISAUnited formally adopts the work of the International Organization for Standardization / International Electrotechnical Commission (ISO/IEC) and the National Institute of Standards and Technology (NIST) as foundational standards bodies, and the Center for Internet Security (CIS), the Cloud Security Alliance (CSA), and the Open Worldwide Application Security Project (OWASP) as security control–framework organizations. This adoption aligns with each organization's public mission and encourages use by practitioners and institutions. ISAUnited incorporates these organizations into its charter so that every Parent Standard and Sub-Standard is grounded in a common, defensible foundation.*

a) **Foundational Standards (Parent level).**
   ISAUnited adopts *ISO/IEC* and *NIST* as foundational standards organizations. Parent Standards align with these bodies for architectural grounding and auditability, and extend that foundation through ISAUnited's normative, testable specifications. This alignment does not supersede *ISO/IEC* or *NIST*.
b) **Security Control Frameworks (Control level).**
   ISAUnited adopts *CIS, CSA*, and *OWASP* as control framework organizations. Control mappings translate architectural intent into enforceable technical controls within Parent Standards and Sub-Standards. These frameworks provide alignment at the implementation level rather than at the foundational level.
c) **Precedence and scope.**
   Foundational alignment (*ISO/IEC, NIST*) establishes the architectural baseline. Control frameworks (*CIS, CSA, OWASP*) provide enforceable mappings. ISAUnited's security invariants and normative requirements govern implementation details while remaining consistent with the adopted organizations.
d) **Mapping.**
   Each cited control mapping is tied to a defined output, an associated verification and validation activity, and an Evidence Pack ID to maintain end-to-end traceability from requirement to control, test, and evidence.
e) **Attribution.**
   ISAUnited cites organizations by name, respects attribution requirements, and conducts periodic alignment reviews. Updates are recorded in the Change Log with corresponding evidence.
f) **Flow-downs.**

Obsolete and withdrawn documents should not be used; please use replacements.

(Parent to Sub-Standard). Parent alignment to the International *ISO/IEC* and *NIST* flows down as architectural invariants and minimum requirements that Sub-Standards must uphold or tighten. Parent-level mappings to *CIS, CSA*, and *OWASP* flow down as implementation control intents that Sub-Standards must operationalize as controls-as-code, tests, and evidence. Each flow-down MUST reference the Parent clause, the adopted organization name, the Sub-Standard clause that implements it, the associated verification/validation test, and an Evidence Pack ID for traceability. Any variance requires a written rationale, compensating controls, and a time-bounded expiry recorded with an Evidence Pack ID.

## Change Log and Revision History

| Review Date | Changes | Committee | Action | Status |
|---|---|---|---|---|
| January 2026 | Standards Revision | Standards Committee | Publication | Draft v1 published |
| November 2025 | Standards Submitted | Technical Fellow Society | Peer review | Pending |
| October 2025 | Standards Revision | Task Group ISAU-TG39-2024 | Draft submitted | Complete |
| December 2024 | Standards Development (Parent D01) | Task Group ISAU-TG39-2024 | Draft complete | Complete |

End of Document
IO.