Defensible10

ISAUnited
INSTITUTE OF SECURITY ARCHITECTURE UNITED

Defensible 10

Annex D (Normative): D04- Application Security Architecture & Secure Development

Technical Standard

Standards Committee
12-8-2025

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 the integration of 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 (“Shall,” “Should,” “Must”)

- Must / Shall: A mandatory requirement for conformance to the standard.
- Must Not / Shall 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

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.

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 D (Normative): D04- Application Security Architecture & Secure Development

ISAUnited's Defensible 10 Standards
Parent Standard: D04-Application Security Architecture & Secure Development
Document: ISAU-DS-AS-1000
Last Revision Date: December 2025
Peer-Reviewed By: ISAUnited Technical Fellow Society
Approved By: ISAUnited Standards Committee

Section 1. Standard Introduction

The Application Security Architecture & Secure Development Parent Standard (ISAU-DS-AS-1000) establishes the engineering baseline for securing the application layer end-to-end, including web and mobile applications, public and private APIs, microservices, serverless functions, and event-driven backends. As a Parent Standard, it defines shared terminology, scope, requirements (inputs), technical specifications (outputs), and verification and validation expectations that subordinate sub-standards will inherit. It is vendor-neutral and implementation agnostic, aligning with recognized foundational frameworks (NIST, ISO/IEC) while extending them with normative, testable specifications. The intent is to deliver a defensible, measurable, and auditable approach to application-layer security across on-premises, cloud, and hybrid environments.

Objective

This standard establishes foundational principles for Application Security Architecture & Secure Development (ISAU-DS-AS-1000), engineered to protect business logic, interfaces, and data flows across modern application styles. It provides cybersecurity architects, engineers, and software developers with a structured, defensible methodology for building and operating applications that enforce security at trust boundaries, within code paths, and at the first boundary (gateway/edge).

Emphasis is placed on:

- Defining trust boundaries and enforcing contract-true interfaces, including strict request validation and response schema alignment (SRA) with OpenAPI/JSON Schema/Proto, strict mode, and unknown-field rejection.
- Implementing explicit authorization at object, field, and function scope, and enforcing least privilege across all mutating handlers and sensitive reads.
- Hardening token and session design (OAuth2/OIDC, PKCE, scoped claims, rotation, and revocation) and validating tokens on every request.
- Eliminating injection and unsafe deserialization through canonicalization, schema validation, context-correct output encoding, and safe serializers.
- Protecting data in code paths through classification, minimization, masking or tokenization, and correct in-code cryptographic use via approved boundaries.
- Hardening client-facing surfaces (CSP with nonces or hashes, strict CORS, CSRF defenses, clickjacking, and MIME protections) and resisting abuse (rate limits, backpressure, SSRF egress controls).
- Generating structured, defender-useful telemetry and deterministic error semantics with correlation identifiers to support monitoring, forensics, and automated response.

- Producing evidence—design artifacts, contract and abuse-case tests, authorization proofs, and operational logs—that make application security measurable and auditable.

By integrating these engineering-focused capabilities, this standard provides a measurable and defensible framework for securing application architectures and code across monoliths, microservices, serverless, and event-driven systems.

Justification

Enterprise applications span distributed APIs, microservices, clients, and serverless or event-driven components. While this improves agility, it expands the attack surface in ways that perimeter or pipeline controls alone cannot address. Persistent failure classes include broken object, field, and function level authorization (BOLA/BFLA/BOPLA), injection and unsafe deserialization, weak token and session semantics, SSRF via server-initiated outbound requests, and leaky errors or logs that aid enumeration. High delivery velocity further exposes gaps when security is bolted on rather than engineered.

Security must be embedded as application requirements, enforced in code and at the first boundary, and proven through tests and evidence tied to explicit acceptance thresholds. This standard addresses these realities by unifying contract enforcement (including SRA), authorization correctness, input and serialization safety, token and session hardening, client-surface defenses, abuse resistance, telemetry, and evidence into a cohesive application-layer architecture. It aligns with NIST and ISO/IEC at the Parent level. At the same time, detailed control mappings to CSA CCM, CIS, and OWASP are provided in Section 9 (Security Controls) and in associated sub-standards. Through structured requirements, measurable outputs, and rigorous verification and validation, teams can proactively secure applications—reducing the risk of exploits, unauthorized access, and fragile error or telemetry semantics across hybrid, cloud-native, and on-premises deployments.

Section 2. Definitions

Abuse Case — An adversarial user story that exercises edge cases or business logic to confirm the application fails securely (e.g., object/field/function-level authorization).

API Contract — The authoritative interface specification (OpenAPI/JSON Schema/Proto) used to generate validators and tests; enforced in strict mode with unknown-field rejection and bounds checks.

API Gateway (First Boundary) — The gateway/edge where authentication, authorization, and contract/schema validation are enforced before requests reach application code.

ASR-ID (Application Security Requirement ID) — A uniquely identified, testable requirement (e.g., API-AUTH-xx, DATA-VAL-xx, TOKEN-xx) traced to code and tests.

Authorization Models — RBAC/ABAC/ReBAC decisions applied at object, field, and function scope to enforce least privilege.

Bounds Checks — Numeric/range/enum validation applied to request and response fields to prevent out-of-contract values.

BOLA / BFLA / BOPLA — Broken Object-, Function-, and Object-Property-Level Authorization classes where identity is valid but access decisions are incorrect.

Business-Logic Abuse Suite — A named set of tests that exercise workflow and sequencing abuse (e.g., excessive actions, bypass of step order) to validate invariants beyond simple input validation.

Canonicalization — Normalizing inputs (paths, encodings, Unicode) prior to validation and authorization to prevent alternate-representation bypass.

Client Interaction Hardening — Controls at the application boundary (CSP, strict CORS, CSRF defenses, clickjacking/MIME protections, pagination/size/time limits, cache controls, SRI where used).

Content Security Policy (CSP) — A response header restricting browser resource loading/execution; uses nonces or hashes for dynamic content.

Correlation Identifier (trace_id) — A stable identifier propagated across requests/responses and logs to link events and evidence.

CORS (Cross-Origin Resource Sharing) — Explicit, minimal cross-origin policy for allowed origins, methods, and headers; validated at the first boundary.

CSRF Defense — Protections for state-changing endpoints (same-site cookies, anti-CSRF tokens, per-request nonces).

Data Classification & Minimization — Assigning sensitivity levels; limiting collection, use, and retention to what is strictly required.

Deterministic Error Template — User-facing error format that omits sensitive detail while returning a correlation ID; full diagnostics appear only in logs.

DFD (Data Flow Diagram) — Diagram of components, data stores, and flows used to locate trust boundaries and derive requirements/tests.

Defense in Depth — Layered controls at boundaries and in code (canonicalize → validate → authorize → encode → log) so no single failure compromises the system.

Encoder-at-sink — The practice of applying context-appropriate output encoding immediately before a rendering/execution sink (HTML/JS/CSS/URL/SQL) to prevent injection.

Evidence Pack (EP-04) — The versioned bundle of artifacts for this annex (and child packs EP-04.x) containing DFDs, contracts, ASR-IDs, tests, logs, and V&V evidence proving conformance.

Evidence SLOs — Quantified acceptance limits used for validation (e.g., contract pass rate = 100 %, CSP violation rate ≤ 0.1 % over 7 days).

File/Media Handling — Safe processing of uploads (type/size/extension checks, content sniffing disabled, storage outside webroot, sandbox/AV where justified).

First Boundary (Gateway/Edge) — The application’s ingress control point where contract enforcement, authentication, and authorization are applied before code execution.

HSTS — HTTP Strict Transport Security; enforces HTTPS for defined max-age and prevents downgrade/stripping.

Idempotency-Key — A unique, time-bounded token on mutating HTTP routes ensuring once-only effects under retries.

IdP (Identity Provider) — The trusted issuer of identities and tokens (OAuth2/OIDC).
Ingest Schema Conformance — The requirement that events at telemetry ingest include all required fields (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code) with a 100 % pass rate.

Input Validation — Whitelist-oriented checks (type/range/length/format/schema) performed at every trust boundary before parsing or interpretation.

JWT — JSON Web Token used to carry claims; validated for issuer, audience, expiry, algorithm, and signature on every use.

mTLS — Mutual TLS providing authenticated, encrypted service-to-service communications.

Non-Functional Security SLOs — Measurable acceptance thresholds for security behavior (e.g., log-redaction error rate, throttle/block rates).

OAuth2 / OIDC — Standards for delegated authorization and identity; include scopes/claims, PKCE for public clients, and token introspection/revocation.

Output Encoding — Context-appropriate encoding (HTML/JS/CSS/URL/SQL parameters) applied immediately before a sink to prevent injection.

Permissions-Policy — A response header that restricts access to browser features (e.g., camera, microphone, geolocation) to the minimum necessary.

PKCE — Proof Key for Code Exchange; protects public clients in OAuth2/OIDC authorization code flow.

RASP (Runtime Application Self-Protection) — In-process instrumentation that detects/blocks exploitation attempts within application execution paths.

Rate Limiting / Backpressure — Per-principal and per-route controls (token/leaky buckets, circuit breakers) to bound resource consumption and resist abuse.

Referrer-Policy — A response header that controls how much referrer information is sent with requests to other origins.

Response Schema Alignment (SRA) — Enforcement that responses conform to declared schemas with type/enum/bounds checks; reject or normalize on mismatch.

Serialization/Deserialization Safety — Use of safe formats/libraries; allowlisted types; size/time limits; gadget resolution disabled for untrusted data.

Session Management — Establishment, rotation, expiry, and revocation of authenticated state; cookies set with Secure/HttpOnly/SameSite as applicable.

SIEM — Security Information and Event Management platform that ingests application events for detection and investigation.

Sinks — Code paths where data is executed or rendered (DB queries, templates, eval/exec, shell); require validation/encoding before use.

SLO (Service Level Objective) — Target threshold indicating acceptable security behavior and validation success criteria.

SRI (Subresource Integrity) — A browser mechanism that verifies third-party script/style assets against a cryptographic hash to prevent tampering.

SSRF Protections — Outbound-request safeguards (egress allowlists, metadata/localhost blocks, protocol/port restrictions, DNS pinning where supported).

STRIDE — Threat modeling method (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) used to derive requirements/tests.

Taint Tracking — Propagation of “untrusted” labels through code to ensure validation/encoding occurs before sensitive sinks.

Template/Server-Side Injection Safety — Use of auto-escaping template engines and parameterized DB/APIs; prohibition of dynamic evaluation/concatenation reaching execution sinks.

Test-ID — Stable identifier for a verification/validation test case mapped to specific §6 outputs and ASR-IDs, with Owner and Frequency.

Threat-Model Delta — A concise record of how the threat model changes when routes, contracts, tokens, or boundaries change, including impacted tests and evidence.

Token Design — Construction/handling of tokens (claims, scopes, audiences, expiry, rotation, revocation, nonce/jti) and on-request validation.

Trust Boundary — A crossing between principals/processes/contexts with different trust; requires authentication, authorization, validation, and observability.

Validate Sequence (Canonicalize → Validate → Authorize → Encode) — The required order at boundaries to ensure safe handling before any sink.

Work Queue / State Store Hardening — Integrity (HMAC/AEAD) for messages/sessions across boundaries; TTL/de-duplication/idempotency; schema enforcement.

Zero Trust (Application Layer) — No implicit trust between components; every request is authenticated, authorized, validated, and logged at each hop.

Section 3. Scope

Modern applications span distributed web and mobile front ends, public and private APIs, microservices, serverless functions, and event-driven backends. This creates frequent trust-boundary crossings that cannot be secured by perimeter or pipeline controls alone. The scope of ISAU-DS-AS-1000 covers the application layer end to end: interface contracts, authorization decisions, input and serialization safety, token and session semantics, client-interaction defenses, abuse resistance, data protection in code paths, application-layer telemetry, and evidence production across monoliths,

microservices, serverless, and event-driven systems deployed on-premises, in the cloud, and in hybrid environments.

This Parent Standard defines the architectural expectations, engineering methods, and technical guardrails required to achieve measurable, defensible application behavior. It helps practitioners model trust boundaries, enforce contract-true interfaces (including strict request validation and response schema alignment, SRA), verify authorization correctness, eliminate injection and unsafe deserialization, harden tokens and sessions, resist abuse and SSRF, and produce auditable evidence—while remaining language-, framework-, and vendor-neutral.

Applicability

- Application Types and Styles: Web and mobile applications, public and private APIs, microservices, service consumers, serverless functions, and event processors.
- Environments: On-premises, single-cloud, multi-cloud, and hybrid deployments.
- Roles: Application architects, software engineers, API designers, AppSec engineers, and reviewers accountable for application-layer security decisions and evidence.

Key Focus Areas

- Trust Boundaries and Contracts: Define DFDs; enforce OpenAPI/JSON Schema/Proto in strict mode with unknown-field rejection and bounds checks, and require SRA for response schemas.
- Authorization Models and Enforcement: RBAC/ABAC/ReBAC decisions at object, field, and function scope; deny-by-default on protected resources.
- Input, Serialization, and Encoding Safety: Canonicalize → validate → authorize → encode at every boundary; safe deserialization with allowlists and size/time limits.
- Session and Token Security: OAuth2/OIDC with PKCE as applicable; issuer/audience/scope validation per request; rotation, revocation, and replay resistance.
- Data Protection in Code Paths: Classification and minimization; masking/tokenization; correct in-code cryptographic use through approved boundaries.
- Client Interaction Hardening: CSP with nonces/hashes, strict CORS, CSRF defenses, clickjacking/MIME protections, pagination/size/time limits.
- Abuse Resistance and SSRF Controls: Per-principal/route throttles and backpressure; egress allowlists, metadata/localhost blocks, protocol/port

constraints.

- Application-Layer Telemetry and Errors: Structured events with required fields (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code); schema-conformant ingest and upstream immutability.
- Evidence Production: Versioned Application Evidence Pack (ID) containing architecture artifacts, contracts, tests (unit/contract/abuse), header/policy scans, token/session drills, abuse/SSRF logs, and telemetry samples.

Exclusions & Interfaces (authoritative)

- CI/CD mechanics, artifact signing/attestation, SBOM/provenance, environment parity, and promotion/rollback controls are governed by Annex J (DevSecOps & Secure SDLC Engineering).
- Cryptographic module selection, key lifecycles, certificate issuance, and transport policy profiles are governed by Annex I (CEK); Annex D specifies correct cryptographic use in application code and at application boundaries.
- Detection engineering, SOC workflows, and incident playbooks are governed by Annex H (MDIR); Annex D defines the application-layer events and semantics that those functions consume.

Outcomes

By defining this scope, ISAU-DS-AS-1000 ensures that application security is:

- **Defensible:** Explicit trust boundaries, contract-true interfaces (request and response), explicit authorization decisions, and auditable, test-proven behavior.
- **Measurable:** Acceptance thresholds tied to leading indicators—authorization coverage on mutating handlers, contract pass rate, CSP violation rate, and abuse throttle/block rate.
- **Adaptive:** Patterns apply consistently across monoliths, microservices, and serverless with minimal redesign as systems evolve.
- **Aligned:** Clean interfaces to CEK (Annex I) and DevSecOps (Annex J), consistent with organizational policy and foundational standards.

This scope provides the foundation for engineering applications that withstand modern attack techniques at the code and interface layers while supporting product agility and operational integrity.

Section 4. Use Case

Achieving resilient application security requires deliberate practice in real-world systems—not just theoretical patterns. The following consolidated use case illustrates a complex enterprise that operates distributed web applications and APIs. It surfaces common application-layer weaknesses, maps them to targeted application controls, and defines measurable outcomes. This links architecture decisions directly to defensible results in production behavior.

Table D-1:

Field	**Details**
Use Case Name	Securing Web Applications and APIs Against Injection, Abuse, and Authorization Flaws
Objective	Apply Zero-Trust Application Security to eliminate injection/serialization risks, prevent BOLA/BFLA/BOPLA, and harden sessions/tokens—while producing application-layer evidence that requirements are met.
Scenario	A fintech platform exposes multi-tenant web apps and public/private APIs. Reviews show object/field/function-level authorization gaps, inconsistent input/contract validation, fragile token/session handling, SSRF-prone server-initiated outbound requests, and leaky error semantics that aid enumeration.
Actors	Application Security Architect; Product/API Owner; Lead Software Engineer; AppSec Engineer; QA/Automation Engineer; Privacy Engineer; SOC Analyst
Challenges Identified	• Authorization gaps (BOLA/BFLA/BOPLA) • Inconsistent canonicalization/validation; unsafe deserialization • Token/session TTLs too long; weak replay/fixation defenses; missing per-request ISS/AUD/Scope checks • Permissive CORS; missing CSP nonces/hashes; incomplete CSRF • No per-principal throttles; missing SSRF egress controls • Unstructured logs; revealing errors; missing trace_id/control_id
Technical Solution	

Field	Details
	Trust Boundaries & Contracts: Define DFDs and trust zones; enforce OpenAPI/JSON Schema/Proto in strict mode with unknown-field reject and bounds checks at the gateway and in code; require response schema alignment (SRA). Authorization: RBAC/ABAC/ReBAC; explicit object/field/function decisions on 100 % mutating handlers; deny-by-default; require Idempotency-Key on POST/PUT/PATCH. Input/Serialization: Canonicalize → validate → authorize → encode; disable unsafe deserialization; allowlist types with size/time limits. Session/Token: OAuth2/OIDC (+PKCE for public clients); validate issuer/audience/scope on every request; access-token TTL ≤ 60 minutes; privileged inactivity ≤ 15 minutes; rotation/revocation ≤ 5 minutes; block replay/fixation. Client Surface: CSP with nonces/hashes; strict CORS; CSRF; HSTS where applicable; clickjacking/MIME defenses; pagination/size/time bounds. Abuse/SSRF: Per-principal/route rate limits and backpressure; SSRF egress allowlists, metadata/localhost blocks, protocol/port constraints; DNS pinning where supported. Telemetry & Errors: Structured events (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code); deterministic user errors with correlation IDs; schema-conformant ingest = 100 %; upstream immutability. RASP (where justified): Instrument critical paths to detect/block exploitation in-process with FP ≤ 1 % and ≤ 5 ms p50 overhead.
Expected Outcome (acceptance thresholds)	• Authorization: 100 % mutating handlers and sensitive reads execute explicit authorization; BOLA/BFLA/BOPLA suite = 100 % pass. • Contracts: Contract/negative tests = 100 % pass on external routes; SRA response checks pass for targeted services; Idempotency-Key enforced on all mutating routes. • Injection/serialization: Critical injection/unsafe deserialization = 0. • Tokens/Sessions: 100 % tokens validated per request (ISS/AUD/Scope); TTL/rotation/revocation targets met; replay/fixation blocked with evidence. • Client surface: CSP violation rate ≤ 0.1 % over 7 days; CSRF tests pass; HSTS enabled where applicable. • Abuse/SSRF: ≥ 95 % automated abuse throttled/blocked; SSRF attempts blocked and logged. • Telemetry: 100 % events include required fields; ingest schema conformance = 100 %; audit logs append-only/tamper-evident.
Evidence (Application)	DFDs/trust-boundary maps; contract/negative and SRA test reports; BOLA/BFLA/BOPLA suite results; idempotency duplicate-request tests; token/session drill logs; header scans (CSP/CORS/CSRF/HSTS); abuse/SSRF simulation logs; structured event samples with trace_id/control_id and ingest conformance report; “SLO met / not met” sheet - all artifacts stored under the Application Evidence Pack ID.

Key Takeaways:

- The direct mapping of application risks to engineering controls guarantees actionable, measurable security improvements.
- Zero Trust and SSDLC drive consistent enforcement, testing, and monitoring across complex application environments.
- Metrics and operational drill-down validate outcomes, supporting continuous improvement and regulatory alignment.

Section 5. Requirements (Inputs)

A defensible application security architecture is grounded in clearly defined, actionable inputs. These requirements establish the technical and procedural preconditions that must be present before design and implementation proceed. Meeting these inputs ensures engineering teams can produce measurable, auditable application-layer outcomes.

5.1 Threat Modeling Practice & Artifacts

A documented method (e.g., STRIDE) must exist to produce DFDs, trust-boundary maps, and an abuse-case catalog per application/service. Artifacts are version-controlled and referenced in change reviews.
Proof: Link to current DFDs, trust-boundary map, and abuse-case list (EP-04:/architecture/ or EP-04.1:/architecture/).

5.2 Application Security Requirement Catalog (ASR-IDs)

A maintained, testable requirement set must exist (e.g., API-AUTH-xx, DATA-VAL-xx, TOKEN-xx, ERR-xx), each mapped to enforcement points and tests.
Proof: ASR-ID register (CSV/MD) showing spec → code location(s) → Test-IDs (EP-04:/requirements/ or EP-04.1:/requirements/).

5.3 API Inventory & Contract Repository

All externally reachable and inter-service endpoints must be inventoried and linked to authoritative contracts/schemas (OpenAPI/JSON Schema/Proto) in strict mode with unknown-field rejection and bounds checks.
Proof: API inventory + contract repo URLs; CI job output showing strict-mode enabled (EP-04:/contracts/).

5.4 Authentication & Authorization Baseline

The application must declare the authentication pattern (e.g., OAuth2/OIDC, mTLS for service calls) and the authorization model (RBAC/ABAC/ReBAC) down to object/field/function scope, with decision points and enforcement locations identified.

Proof: Authn/z decision map per route/handler; policy snippets or guards in code (EP-04:/authorization/).

5.5 Input/Serialization & Output-Encoding Standards
Language/framework-specific standards must exist for canonicalization, validation, safe deserialization (allowlists, size/time limits), and context-correct output encoding (HTML/JS/CSS/URL/SQL parameters). Approved libraries/utilities are listed.
Proof: Coding standard + linter/static-rule config referencing approved libraries (EP-04:/coding-standards/).

5.6 Data Classification & Privacy-by-Design
Data elements processed by the app must be classified; minimization and masking/tokenization rules must be defined; and privacy requirements must be captured as ASR-IDs and linked to code paths.
Proof: Data map with classification/retention; masking/tokenization rules (EP-04:/data/).

5.7 Session & Token Lifecycle Policy
Rules must exist for token/session lifetimes, rotation/revocation, replay protection (nonce/jti), cookie flags (Secure/HttpOnly/SameSite), and audience/scope semantics; validation per request is specified.
Proof: Token/session policy; automated tests for TTL/rotation/revocation; cookie-flag scanner results (EP-04:/tokens/).

5.8 Application-Layer Telemetry & Error Semantics
A structured logging schema is defined for security events (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code). User-facing error templates avoid sensitive detail and include correlation IDs; upstream log immutability is specified.
Proof: Logging schema, sample events, error templates, and immutability settings (EP-04:/telemetry/).

5.9 Abuse-Resistance & Egress Safety Hooks
Per-principal/per-route throttles/backpressure and pagination/size/time bounds are defined. Server-initiated outbound requests implement SSRF guards (egress allowlists, metadata/localhost blocks, protocol/port constraints; DNS pinning where available).
Proof: Gateway/app rate-limit policy; SSRF egress allowlist and enforcement config (EP-04:/abuse-ssrf/).

5.10 Dependency & Component Provenance (Application View)
An inventory of in-app libraries/frameworks/engines (serializers, template engines, JSON/XML libs, crypto calls) exists with version constraints and approved usage notes. (Annex J handles SBOM/provenance; Annex D requires the app-side inventory and policy.)

Proof: Dependency inventory with allow/deny and minimum versions; unsafe API ban list (EP-04:/dependencies/).

Practitioner Guidance:

- Traceability first: For each ASR-ID (5.2), point to its contract location (5.3), code enforcement, and named Test-ID. If you cannot draw this chain, the input is not ready.
- Single sources of truth: Keep DFDs, contracts, coding standards, and policies under EP-04 (or child packs EP-04.n).
- Pre-flight before §6: Contracts strict-mode enabled; explicit authorization map for 100 % mutating handlers; token/session policy pinned; logging schema and error templates finalized; abuse/SSRF hooks declared; dependency inventory current.
- Fail-closed stance: Block merges/deploys when 5.3/5.4 are not satisfied; when 5.5 standards are violated; or when mandatory telemetry/error schema (5.8) is absent.
- Ownership: Assign an owner for each 5.x item inside EP-04; re-validate at major releases and at least quarterly.

Section 6. Technical Specifications (Outputs)

These specifications translate the application security policy into measurable, testable application behavior. Outputs are enforced in code, contracts, gateways under the app team's control, and the application-layer runtime. (Annex J. governs pipeline gates, SBOM/provenance, and promotion controls).

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 Identity & Authorization in Application

- Explicit authorization decisions Must execute on 100% of mutating handlers (create/update/delete) and sensitive reads; decisions are object/field/function-level per the chosen model (RBAC/ABAC/ReBAC).
- Authorization test suite (BOLA/BFLA/BOPLA) Must pass 100% on every external route and protected internal route.
- Sensitive functions MFA (where user-interactive) Must be required for account/credential changes or high-impact actions.

- Evidence: unit/integration/contract tests; handler inventory showing authorization coverage.

6.2 API Boundary & Contract Enforcement

- Contract strictness: External and inter-service APIs Must enforce OpenAPI/JSON-Schema/Proto contracts in strict mode; unknown fields rejected; numeric/enum bounds checked. Contract pass rate = 100%.
- Idempotency for mutating routes: Idempotency keys Must be required for client-initiated retries on create/update operations exposed over HTTP.
- Gateway policy: Authentication, authorization, and schema validation Must be enforced at the first application boundary (gateway/edge).
- Response alignment (SRA): Responses Must conform to declared schemas with type/enum/bounds checks; reject or normalize on mismatch.
- Evidence: contract test reports; gateway policy export; idempotency test logs.
- External webhooks/callbacks: Inbound webhooks Must be authenticated (HMAC signature or mTLS), timestamped, and replay-protected; validate source (domain or IP allowlist) and contract schema; reject on signature, freshness, or schema failure.
- Evidence: webhook contract + signature verification tests; replay/clock-skew negative tests (EP-04:/webhooks/).

6.3 Input, Serialization, and Output Encoding

- Canonicalize→Validate→Authorize→Encode Must be followed at every trust boundary.
- Unsafe deserialization Must be prohibited; only allowlisted types with size/time limits; deserialization gadget resolution = disabled.
- Context-correct encoding Must be applied at sinks (HTML/JS/CSS/URL/SQL params).
- Injection SLO: Critical injection findings (code/templating/SQL/NoSQL/LDAP/XPath) = 0.
- Evidence: negative tests; fuzz/grammar tests for parsers; code audit of serializers/templaters.
- Unsafe API bans: Enforce an application-side allow/deny list for risky patterns (e.g., dynamic eval/exec, unsafe reflection, raw SQL concatenation, unsafe deserializers); block on violation in CI and fail at runtime guards where feasible.
- Evidence: linter/static-rule config + violation reports; targeted unit tests for banned patterns (EP-04:/serialization-safety/).

6.4 Data Protection in Code Paths

- Classification & minimization: Data elements Must be classified; collection and retention minimized; masking/tokenization applied where specified.
- Cryptographic use in code: Only CEK-approved primitives and parameters; keys never embedded in code or images; crypto operations performed through approved service boundaries where feasible.

- Logging redaction: Sensitive classes Must be redacted or tokenized in logs; accidental PII in logs ≤ 0.1% over 7 days.
- Evidence: data map, masking rules, crypto call sites, redaction tests.

6.5 Session & Token Security

- OAuth2/OIDC flows Must validate audience, issuer, and scope/claims on every request; PKCE required for public clients.
- Token/Session lifetime: Access token TTL ≤ 60 minutes; inactivity timeout for privileged sessions ≤ 15 minutes; refresh rotation on use; revocation honored within ≤ 5 minutes.
- Replay/nonce: Nonce/jti and anti-replay checks required for state-changing flows.
- Evidence: token inspection tests, revocation/rotation drill logs, cookie flag verification (Secure/HttpOnly/SameSite).

6.6 Client Interaction Hardening

- CSP: Content-Security-Policy with nonces/hashes enabled on pages rendering dynamic content; CSP violation rate ≤ 0.1% over 7 days.
- HSTS: HTTP Strict Transport Security Must be enabled where applicable to prevent downgrade/stripping.
- CORS: Explicit, minimal origins/methods/headers; preflight validation aligned to the trust model.
- CSRF: Required protections (same-site cookies + anti-CSRF tokens) for state-changing endpoints.
- Clickjacking/MIME: X-Frame-Options/Frame-Ancestors and X-Content-Type-Options/NO-SNIFF enabled where applicable.
- Evidence: header scanner output, CSP report samples, CSRF test cases.
- Request bounds: Enforce maximum request size and execution time on relevant endpoints; list/paginated routes Must enforce explicit page size and upper bounds.
- Referrer-Policy: Set an explicit, minimal Referrer-Policy (e.g., no-referrer, strict-origin-when-cross-origin) aligned to the threat model.
- Permissions-Policy: Set Permissions-Policy to disable unused browser features (camera, microphone, geolocation, etc.) by default.

6.7 Abuse Resistance & SSRF Controls

- Rate limits/backpressure: Per-principal and per-route limits for authentication and sensitive operations; ≥ 95% automated abuse throttled or blocked at the app boundary.
- SSRF mitigation: Server-initiated fetches Must use egress allowlists, metadata/localhost blocks, protocol/port constraints, and DNS pinning where available.
- Evidence: throttle/abuse test logs, SSRF block logs, allowlist policy export.

6.8 Application Telemetry & Errors

- Structured events: Security-relevant events Must include ts, actor, action, resource, result, trace_id, control_id, data_class, error_code.
- Error semantics: User-facing errors Must avoid sensitive detail yet include correlation IDs; defender-useful diagnostics Must be present in logs.
- Ingest conformance: Event schema conformance at telemetry ingest Must be 100% for required fields (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code).

- Immutability: Application audit logs Must be append-only/tamper-evident upstream.
- Evidence: log samples, schema validators at ingest, immutability configuration.

6.9 State Stores, Queues, and Caches

- Integrity & isolation: Messages and session state Must carry integrity checks (e.g., HMAC/AEAD) where trust boundaries exist; isolate by tenant/environment.
- TTL & replay: Enforce TTL, unique keys, and de-duplication/idempotency for at-least-once flows.
- Schema: Message schema validation Must be enforced; schema pass rate = 100%.
- Evidence: queue/cache config, replay tests, schema validator results.

6.10 Risk-Based RASP / In-App Controls (where justified)

- For high-risk code paths, RASP or equivalent in-process checks May be deployed in block or report mode with a false positive rate ≤ 1% and latency impact ≤ 5 ms p50.
- Evidence: RASP policy, block/report event samples, latency measurements.

Practitioner Guidance:

- Trace from ASR to Test: For every ASR-ID, point to the code location(s), the contract/gateway policy (if any), and the named tests proving the behavior (unit/contract/abuse). Record Test-ID, Owner, Frequency, and link artifacts to EP-04.
- Prefer contracts over code paths: Enforce schema/contract checks at the first boundary; retain code-level validation/authorization for depth.
- Measure what matters: Track four indicators per app—authorization coverage (mutating handlers), contract pass rate, CSP violation rate, abuse throttle/block rate. Store weekly snapshots under EP-04.
- Defer to CEK/MDIR/J where appropriate: Use CEK for crypto parameters and key handling, MDIR for downstream detection workflows, and Annex J for pipeline gates/promotion. Annex D owns application semantics and tests.

Quick Win Playbook

Title: Contract-Strict + Idempotency Gate for a High-Value API

Objective: Prevent malformed or replayed requests from reaching application code by enforcing strict request/response contracts and once-only effects on mutating routes.

Target: Enforce strict contract validation and idempotency on one high-value API service (§6.2, §6.3).

Component/System: API gateway (first boundary) + application service (OpenAPI/JSON Schema/Proto).

Protects: Application from injection/serialization flaws, malformed requests, and duplicate side effects.

Stops/Detects: Unknown fields, out-of-bounds values, response schema (SRA) mismatches, and client retries without Idempotency-Key.

Action:

- Enable strict mode at the gateway (unknown-field reject, numeric/enum bounds checks); deploy generated validators in the service.
- Require Idempotency-Key on POST/PUT/PATCH; define expected behavior for missing/duplicate keys (for example, 400/409; no second side effect).
- Add SRA checks for all success/error responses; reject or normalize on mismatch.
- Run a smoke test: (1) valid request → allow; (2) unknown field/out-of-bounds → 4xx; (3) duplicate POST with same Idempotency-Key → no second side effect; (4) intentionally malformed response → blocked/normalized with logged event.

Proof: Contract manifest; gateway policy export; contract/negative and SRA test results (valid/invalid/idempotent cases); deny/validation logs with trace_id/control_id → EP-04.1.

Metric: Contract/negative tests pass rate = 100 % on targeted service; 100 % mutating routes enforce Idempotency-Key; 100 % of responses pass SRA; unknown-field/bounds/SRA violations produce 4xx with trace_id/control_id in logs.

Rollback: Toggle gateway strict mode and the Idempotency-Key requirement off for the service; revert the validator middleware commit; record the exception with owner and expiry in EP-04.1.

Section 7: Cybersecurity Core Principles

The following ISAUnited Cybersecurity Core Principles provide the engineering foundation for a defensible application security architecture. Each principle Must be applied to the design, coding, and operation of applications and APIs—so that trust boundaries, authorization, data handling, and telemetry are provably correct at the application layer.

Table D-2:

Principle Name	Code	Applicability to Application Security Architecture & Secure Development
Least Privilege	ISAU-RP-01	Enforce minimum necessary permissions in code and at interfaces; apply RBAC/ABAC/ReBAC decisions at object, field, and function scope on 100% of mutating handlers.
Zero Trust	ISAU-RP-02	Require explicit authentication and authorization on every boundary crossing (user→app, service→service); never rely on network location for trust.
Complete Mediation	ISAU-RP-03	Validate, authorize, and log every access to protected resources after canonicalization and before execution; no cached authorizations without re-checks on sensitive actions.
Defense in Depth	ISAU-RP-04	Layer controls at boundaries (contract/schema validation, authorization, output encoding, rate limits, SSRF egress allowlists) so that a single control failure does not compromise the system.
Secure by Design	ISAU-RP-05	Derive requirements from DFDs and threat models; encode them as ASR-IDs and tests (unit/contract/abuse) that prove application behavior.
Minimize Attack Surface	ISAU-RP-06	Remove unused routes/features; restrict parsers, serializers, and MIME types; apply strict CORS/CSP; disable verbose errors and debug endpoints in production.

Principle Name	Code	Applicability to Application Security Architecture & Secure Development
Secure Defaults	ISAU-RP-10	Deny-by-default routes, strict schema mode, safe deserialization off by default, cookies with Secure/HttpOnly/SameSite; explicit action required to relax protections.
Resilience & Recovery	ISAU-RP-14	Design for graceful failure: idempotency keys, circuit breakers, backpressure; ensure error semantics do not leak state while enabling rapid recovery.
Evidence Production	ISAU-RP-15	Emit structured security events with trace_id and control_id; maintain tamper-evident audit trails that support forensics and verification of requirements.
Cryptographic Agility	ISAU-RP-17	Use CEK-approved crypto via abstractions; prevent algorithm lock-in so app code can adopt new suites without redesign.
Protect Confidentiality	ISAU-RP-18	Classify data; minimize collection; apply in-code protections (AEAD/HMAC via approved services); redact sensitive fields in logs and errors.
Protect Integrity	ISAU-RP-19	Validate contracts and signatures; apply integrity checks (HMAC/AEAD) for state stores, queues, and messages crossing trust boundaries.
Protect Availability	ISAU-RP-20	Bound work and memory per request; throttle abusive patterns; ensure business operations hold under adverse inputs without breaking invariants.
Make Compromise Detection Easier	ISAU-RP-16	Design events, error codes, and correlation IDs so defenders can quickly detect abnormal flows (e.g., failed object-level authorization, contract violations, SSRF blocks).

Practitioner Guidance:

These cybersecurity core principles must be systematically integrated into every facet of application security engineering and software development—from architectural blueprints to automated pipelines and operational runbooks.

	• Trace principle → spec → test: For each principle in Table D-2, map it to one or more §6 outputs and a named test in §12. Example: ISAU-RP-03 → 6.2 Contract Enforcement → "Contract-Strict-All-External" test. Record the artifact path in the Application Evidence Pack ID. • Make it measurable: Convert principle intent to an acceptance threshold (e.g., RP-01: "authorization present on 100% mutating handlers"; RP-06: "unknown fields rejected on 100% external routes"). Fail builds or block releases when not met (Annex J will consume these tests). • Design first, then code: Derive ASR-IDs from DFDs and abuse cases; place the enforcement point in code and at the first boundary (gateway/edge) for depth. • Review exceptions: Any temporary relaxation (e.g., broader CORS for a pilot) Must include scope, compensating control, owner, and sunset date; keep the exception record with the Application Evidence Pack.

Section 8: Foundation Standards Alignment

This section identifies the internationally recognized foundational frameworks adopted by ISAUnited for Application Security Architecture & Secure Development. ISAUnited adopts NIST and ISO/IEC as its foundational baselines. These provide the life-cycle, governance, and control context that this Parent Standard refines into application-layer, testable specifications.

Purpose and Function

- Demonstrate alignment with NIST/ISO requirements and guidance.
- Bridge compliance baselines to application-layer engineering specifications in this annex.
- Provide a stable reference for sub-standards to map requirements and evidence at the clause level.

Table D-3:

Framework	Standard / Reference	Applicability to Application Security Architecture & Secure Development
NIST	SP 800-218 (SSDF)	Secure Software Development Framework — lifecycle tasks for requirements, design, coding, verification, and release that underpin app-layer engineering.

Framework	Standard / Reference	Applicability to Application Security Architecture & Secure Development
NIST	SP 800-53 Rev.5	Security & Privacy Controls — SA, AC, AU, IA, SC, SI families most relevant to application requirements (authn/z, input validation, logging, data protection).
NIST	NIST SP 800-160 Vol. 1	Systems Security Engineering — gives the engineering/V&V discipline we invoke in §§10 and 12 (requirements→evidence→assurance) yet stays technology-agnostic and application-layer compatible.
NIST	SP 800-204 Series	Microservices/API/Kubernetes security strategies — guidance for API trust boundaries, service identity, and app-centric zero-trust patterns.
NIST	SP 800-207	Zero Trust Architecture — continuous verification and policy enforcement at application boundaries (user→app, service→service).
NIST	SP 800-63 Series *(pin 63B)*	Digital Identity Guidelines — 63B informs authentication/session lifecycle used in §6.5 (AALs, session binding, verifier requirements).
ISO/IEC	27001:2022 / 27002:2022	ISMS requirements and control catalog — organizational and technical controls operationalized at the application layer (access control, logging, secure development).
ISO/IEC	27034-1	Application Security — process framework for designing, implementing, and evaluating application security throughout the SDLC.
ISO/IEC	ISO/IEC 27034-2	Organization Normative Framework for Application Security — complements 27034-1 by specifying how an organization operationalizes app-sec processes that map cleanly to our Evidence Pack and §12 V&V.
ISO/IEC	27034-6 *(optional but helpful)*	Organizational application security processes — supports §12 V&V practices and Evidence Pack discipline.
ISO/IEC	27036-1	

Framework	Standard / Reference	Applicability to Application Security Architecture & Secure Development
		Information Security for Supplier Relationships — Supplier/Software Assurance Interfacing with Application Dependency Risk.
ISO/IEC	29147 *(optional per ISO)*	Vulnerability Disclosure — coordinated intake/communication for issues surfaced by §12 testing.
ISO/IEC	30111 *(optional per ISO)*	Vulnerability Handling — internal processing, triage, and remediation workflows aligned to §12 corrective actions.

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 to ISO/IEC and NIST for architectural grounding and auditability, and this alignment flows down to Sub-Standards as invariants and 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:

- Cite specific clauses from Table D-3 (e.g., NIST SP 800-218 Task PS.1; ISO/IEC 27034-1 clause X.Y) for every normative output they extend.
- Convert those clauses into testable application behaviors (requirements → code enforcement points → named tests in §12) with explicit acceptance thresholds.
- Document any divergence with compensating controls, a rationale, and a sunset date; include passing verification artifacts in the Application Evidence Pack.
- Provide a short mapping table inside the sub-standard: *ASR-ID / Spec → Framework → Clause → Evidence Pack ID*.

Practitioner Guidance:

Practitioners should always map technical controls, process documentation, and audit artifacts directly to these frameworks in design, delivery, and verification phases. When designing sub-standards or organizational supplements:

- Foundations scope only: Use this mapping for NIST and ISO/IEC foundations in §8. Controls frameworks (OWASP/CIS/CSA) map in §9.
- Map at clause level with ownership: For each §6 output (e.g., 6.2 Contract Enforcement, 6.5 Session & Token Security), add a row to your mapping sheet with columns: ASR-ID / Spec → Framework → Clause/Task ID → Standard version/date → Enforcement point (code or gateway) → Test-ID → Owner → Frequency → Evidence (EP-04 path).
- Use SSDF as the backbone: Ensure every relevant NIST SP 800-218 task is backed by a concrete application behavior and a passing §12 test (unit/contract/abuse), with Test-ID recorded.
- Pin specifics where required: When referencing the Digital Identity suite, pin SP 800-63B for authentication/session items in §6.5. *(API Top 10 version pinning belongs in §9, not here.)*
- Keep mappings current in PRs: When a requirement or enforcement point changes, update the clause/task reference in the same PR, include the diff, and store artifacts under EP-04.
- Trace to Table D-6: For each row, record the corresponding Table D-6 entry (Requirement ID and Related §6 Outputs) so auditors can follow clause → spec → enforcement → test → evidence.
- Divergence discipline: If a clause cannot be met verbatim, document the compensating control, the rationale, and a sunset date, then include verification proving equal or stronger application-layer effect (evidence in EP-04).

Section 9: Security Controls

This section specifies the technical control families and control references enforced by the Application Security Architecture & Secure Development Parent Standard. These mappings ensure traceability between application-layer requirements and recognized industry frameworks—providing explicit, actionable guidance for engineers, reviewers, and auditors.

Purpose and Function

Security controls bridge architectural objectives and actionable safeguards at the application layer—protecting confidentiality, integrity, availability, authentication, authorization, and auditability in code and interfaces.

By mapping to CSA CCM, CIS Controls v8, and OWASP ASVS/API Top 10, ISAUnited ensures:

- Alignment with widely adopted best practices,
- Interoperability across stacks, languages, and patterns,
- Audit-ready traceability into sub-standards and project implementations.

Implementation Guidance

Sub-Standard Authors and practitioners Must:

- Reference controls from CSA CCM, CIS Controls v8, and OWASP (ASVS/API Top 10) that are directly enforced in the application.
- Provide framework acronym, control family/ID, and a concise, implementation-oriented description.
- Map each control to one or more §6 outputs and to named tests in §12 (Verification & Validation).
- Favor enforceable controls (contract checks, authorization decisions, header/policy settings, validation/encoding) over policy-only statements.

Table D-4. Control Mappings for Application Security Architecture & Secure Development:

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
CIS v8	3.x (Data Protection)	Application-level data classification, minimization, masking/tokenization, and log redaction.	6.4 Data Protection, 6.8 Telemetry & Errors
CIS v8	6.x (Access Control Management)	Authorization policy definition and enforcement at application boundaries and in code (least privilege, deny-by-default).	6.1 Identity & Authorization
CIS v8	16.x (Application Software Security)	Define app security requirements; enforce input validation, authn/z,	6.1 Identity & Authorization; 6.2 Contract Enforcement; 6.3

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
		error handling, and secure coding across the SDLC (app scope).	Input/Serialization/Encoding; 6.6 Client Hardening
CIS v8	8.x (Audit Log Management)	Emit, protect, and retain application security logs with required fields; support forensics and correlation.	6.8 Telemetry & Errors
CIS v8	18.x (Penetration Testing)	Perform application-layer testing (authorization abuse, injection, deserialization) and track remediation results.	6.1; 6.2; 6.3; 6.7

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
CSA CCM	AIS (Application & Interface Security)	Apply secure application design, input/contract validation, and interface protections at exposed and inter-service APIs.	6.2 Contract Enforcement; 6.3 Input/Serialization/Encoding
CSA CCM	IAM (Identity & Access Management)	Enforce strong authentication and granular authorization at app boundaries and in code paths.	6.1 Identity & Authorization; 6.5 Session & Token
CSA CCM	DCS / DSI (Data Security)	Protect sensitive data in application flows: classification, minimization, masking/tokenization, and in-code crypto use per CEK.	6.4 Data Protection in Code Paths
CSA CCM	TVM (Threat & Vulnerability Management)	Application-layer testing and remediation tracking (contract/negative suites, authorization/abuse testing).	6.1; 6.2; 6.3; 6.7

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
OWASP ASVS	V1 (Architecture & Design)	Threat modeling, trust boundaries, and attack-surface minimization proven by architecture artifacts and tests.	6.2 Contract Enforcement; 6.9 State Stores
OWASP ASVS	V2 (Authentication)	Standards-based auth flows; token/session lifecycle, replay resistance, cookie flags.	6.5 Session & Token
OWASP ASVS	V3 (Session Management)	Session lifecycle controls (establish/rotate/revoke), cookie flags, fixation/replay resistance.	6.5 Session & Token

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
OWASP ASVS	V4 (Access Control)	Object/field/function-level authorization with deny-by-default and explicit decisions on 100 % mutating handlers.	6.1 Identity & Authorization
OWASP ASVS	V5 (Validation, Sanitization, Encoding)	Canonicalize → validate → authorize → encode at each boundary; safe deserialization; upstream/downstream schema checks.	6.3 Input/Serialization/Encoding; 6.2 Contract Enforcement
OWASP ASVS	V7 (Error Handling & Logging)	Deterministic error semantics and structured logging suitable for detection and forensics.	6.8 Telemetry & Errors
OWASP ASVS	V9 (Communications)	TLS everywhere; HSTS where applicable; mTLS for service-to-service as required.	6.6 Client Hardening
OWASP ASVS	V12 (Files & Resources)	Safe file/media handling: type/size checks, storage outside webroot, sandbox/AV as justified.	6.6a File & Media Handling
OWASP ASVS	V14 (Config & Operations)	Secure headers (CSP, HSTS, Referrer-Policy, Permissions-Policy), deterministic error semantics, operational checks.	6.6 Client Hardening; 6.8 Telemetry & Errors

Framework	Control ID	Control name/description (application-layer)	Primary linkage to §6 outputs
OWASP API Top 10	API1, API5	Broken Object/Function Level Authorization—prevent and test BOLA/BFLA/BOPLA across APIs.	6.1 Identity & Authorization
OWASP API Top 10	API2	Broken Authentication—harden flows and tokens; validate issuer/audience/scope.	6.5 Session & Token
OWASP API Top 10	Injection & SSRF *(e.g., API8/2019 Injection; API7/2023 SSRF)*	Enforce schema validation/SRA, safe deserialization, encoder-at-sink, and SSRF egress controls.	6.2; 6.3; 6.7

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 application-layer threats and frameworks evolve, sub-standards may incorporate additional OWASP controls (e.g., ASVS sections beyond those listed) where they are directly enforced by application behavior. Foundational NIST/ISO references remain limited to §8.

Sub-Standard Expectations

Sub-standards under this Parent Standard Must:

- Select and enforce explicit application-layer controls relevant to their scope (e.g., authorization design, input/contract safety, token/session, data-in-code protection).
- Provide detailed mappings from each control to §6 outputs, §12 tests, and an Application Evidence Pack ID.
- Document any deviation from control families with compensating controls and a sunset date; include passing verification artifacts.

Practitioner Guidance:

- Maintain a Controls → Outputs → Tests sheet: For every row in Table D-4, record: Control (CSA/CIS/ASVS/API10) → §6 output(s) → Test-ID(s) (§12) → Enforcement point *(code or first boundary)* → Owner → Frequency → Evidence (EP-04 path).
- Map at clause/task precision: Cite the exact ASVS clause (e.g., V4.1.2), CIS sub-control (e.g., 16.12), or CSA CCM ID (e.g., AIS-xx). Convert each into an enforceable behavior in code or gateway policy.
- Pin OWASP API Top 10 version: In sub-standards and project sheets, explicitly pin 2019 or 2023 and cite the exact category (e.g., API7/2023 — SSRF).
- Express controls as code or boundary policy: Prefer contracts/headers/rules at the first boundary (gateway/edge) and keep code-level checks for depth; avoid policy-only statements.
- Change control in PRs: When a route, contract, authorization rule, or header policy changes, update the mapping row in the same PR, attach proof (test results, policy diffs, header scans), and store artifacts under EP-04.
- Acceptance criteria: A control is "implemented" only when its Test-ID passes in §12 V&V and evidence is present in EP-04 (logs, scans, reports).

	• Divergence discipline: If a framework clause cannot be met verbatim, document the compensating control, rationale, and sunset date; include verification that proves equal or stronger application-layer effect (evidence in EP-04).

Section 10: Engineering Discipline

This section defines the architectural thinking, rigorous engineering processes, and disciplined operational behaviors required to implement the Application Security Architecture & Secure Development (ISAU-DS-AS-1000). ISAUnited's Defensible Standards are not compliance checklists; they are engineered systems—grounded in systems thinking, critical reasoning, and Verification & Validation (V&V)—that produce measurable, auditable, defensible outcomes across applications, APIs, and client interactions.

10.1 Purpose & Function

Purpose. Establish a repeatable, auditable way of working that integrates systems thinking, lifecycle controls, adversary-aware design, and measurable outcomes for application security.

Function in D10S. Parent Standards set expectations and invariants. Sub-Standards convert them into controls-as-code, test specifications, and evidence artifacts embedded in delivery and operations.

10.2 Systems Thinking

Goal: Make the application system end-to-end legible—comprising components, interfaces, dependencies, and failure modes—so controls are positioned where risk manifests.

10.2.1 System Definition & Boundaries

- Declare system purpose, scope, stakeholders, and in-/out-of-scope assets (web/mobile front ends, API gateway, services/microservices, serverless functions, message queues, data stores, IdP, secrets service, logging/SIEM).
- Model trust zones and boundary crossings (user→app, app→service, service→service, app→data store, service→external API).

10.2.2 Interfaces & Contracts

- Maintain Interface Control Documents (ICDs) for application connections (HTTP/gRPC/async endpoints, queue topics, data store access patterns, IdP token flows).
- For each interface, specify: authentication/authorization model, identity type (human/service), contract/schema (OpenAPI/JSON Schema/Proto), data classification, rate/flow limits, error semantics, telemetry fields, and security invariants (e.g., “unknown fields rejected,” “idempotency required on mutating routes”).

10.2.3 Dependencies & Emergent Behavior

- Map shared services (IdP, secrets, time sync, logging, config) and blast radius per dependency.
- Identify emergent risks from composition (e.g., permissive CORS + verbose errors → account enumeration; client retries + no idempotency → duplicate writes; outbound fetch + no egress allowlist → SSRF).

10.2.4 Failure Modes & Safeguards

- For critical paths, document failure modes (broken object/field/function authorization, injection/unsafe deserialization, token replay/fixation, SSRF, telemetry loss) and safeguards (deny-by-default, strict contracts, idempotency keys, CSP nonces, egress allowlists, structured logging with correlation IDs).
- Treat security invariants as non-negotiable requirements (for example: explicit authorization on 100 % mutating handlers, strict contracts at the first boundary (gateway/edge) with bounds checks and response schema alignment (SRA), no secrets in code, tokens validated per request, schema-conformant telemetry ingest = 100 %, and egress allowlists on server-initiated outbound requests).

10.3 Critical Thinking

Goal: Replace assumptions with explicit reasoning that survives review, attack, and audit.

10.3.1 Decision Discipline

- Use Architecture Decision Records (ADRs): problem → options → constraints/assumptions → trade-offs → decision → invariants → Threat-Model Delta → test/evidence plan with Test-ID, Owner, Frequency, and EP-04 path (who/when/how measured).

10.3.2 Engineering Prompts

- **Boundaries**: What is the application system? Where are the trust boundaries and why?
- **Interfaces**: What must always be true at each interface (invariants)? How do we test it?

- **Adversary**: Which attack techniques are credible here (e.g., BOLA/BFLA/BOPLA, injection/XXE, token replay/nonce bypass, SSRF)? What is the shortest attack path?
- **Evidence**: What objective signals prove this control works today and after the change?
- **Failure**: When this fails, does it fail safe? What is the operator's next action?

Required Artifacts (min): ADRs; assumptions & constraints log; evidence plan per decision.

10.4 Domain-Wide Engineering Expectations

Secure System Design

- Define application boundaries (front ends, gateway, services, queues, data stores, IdP, secrets, telemetry sinks).
- Validate boundaries and trust relationships via structured reviews using §10.2 artifacts.

Implementation Philosophy — "Built-in, not bolted-on."

- Integrate controls at design time and the first boundary; avoid post-hoc patching.
- Express controls as contracts/policies-as-code bound to invariants in §10.2.4 (e.g., strict contracts, explicit authorization, idempotency keys, CSP nonces, SSRF egress allowlists).

Lifecycle Integration

- Embed application controls into design reviews, code reviews, and release processes; keep semantics here (Annex D) and delivery mechanics in Annex J.
- Enforce version-controlled reviews with required ADRs and evidence updates on every change.

Verification Rigor (V&V)

- Combine automated checks (contract/negative tests, authorization suites, token/session drills, header scans, SSRF/abuse simulations) with manual probes (adversary-informed business-logic testing).
- Require continuous validation in pipelines and runtime monitoring tied to invariants (e.g., unknown-field reject; deny injection; throttle abuse; block SSRF).

Operational Discipline

- Monitor for drift and unauthorized change; auto-remediate where safe with time-bounded exceptions.
- Maintain playbooks for token revocation/rotation drills, contract rollback, header policy regressions, and SSRF containment.
- Contracts include request strictness and SRA; telemetry includes required fields and 100 % ingest conformance; both are verified by named Test-IDs.

10.5 Engineering Implementation Expectations

- Contracts/Policies as Code. Manage OpenAPI/JSON Schema/Proto, gateway policies, CSP/CORS/CSRF headers, and SSRF egress allowlists as code under version control with peer review and provenance.
- Structured Enforcement Path. Build → contract/negative tests → authorization suite → token/session drills → canary → promote/rollback (execution in Annex J; semantics and acceptance here).
- Explicit Security Boundaries. Maintain diagrams and ICDs; continuously validate posture (strict contracts, explicit authorization, CSP nonces, idempotency, SSRF guards) with targeted audits and smoke tests.
- Automated Security Testing. Integrate static pattern checks for unsafe serializers/encoders, contract test generation, BOLA/BFLA/BOPLA suites, abuse/SSRF simulations before production.
- Traceable Architecture Decisions. Link ADRs to controls, tests, and evidence; include a change-impact checklist that enumerates the affected §6 outputs and the §12 Test-IDs to re-run; update ADRs and evidence in EP-04 on each change request.

10.6 Sub-Standard Alignment (inheritance rules)

Sub-Standards must operationalize this discipline with application-specific detail:

- API Authorization & Contract Enforcement (e.g., ISAU-DS-AS-1010). Explicit authorization on 100% mutating handlers; strict contracts at gateway and in code; idempotency keys on mutating routes; mapped Test-IDs for BOLA/BFLA/BOPLA and contract suites.
- Input/Serialization Safety (e.g., ISAU-DS-AS-1020). Canonicalize → validate → authorize → encode; safe deserialization (allowlists, size/time limits); encoder-at-sink tests.
- Session & Token Design (e.g., ISAU-DS-AS-1040). OAuth2/OIDC with PKCE as applicable; per-request token validation; TTL/rotation/revocation targets; replay/fixation drills.
- Client-Side Protections (e.g., ISAU-DS-AS-1050). CSP with nonces/hashes; strict CORS; CSRF; clickjacking/MIME defenses; header policy scans and CSP report analysis.
- Abuse Resistance & SSRF (e.g., ISAU-DS-AS-1060). Per-principal throttles/backpressure; SSRF egress allowlists/metadata blocks/protocol constraints; simulation logs.

10.7 Evidence & V&V (what proves it works)

Establish an Application Evidence Pack per system containing:

- **Design Evidence**: trust-boundary diagrams, interface/route map with ICDs, invariants register, ADRs (with Threat-Model Delta).
- **Build Evidence**: contracts/policies-as-code, test results (contract/negative, authorization, token/session), header scans, SSRF/abuse simulations.
- **Operate Evidence**: runtime policy/deny logs (unknown fields, authorization denials, CSP/CORS/CSRF reports), token

revocation/rotation drill logs, telemetry samples with required fields and ingest conformance = 100 %, incident and rollback records.
- **Challenge Evidence**: red-team/business-logic test reports, bug-bounty findings, adversary-emulation outcomes, remediation closure with re-test.

10.8 Example: Sub-Standard Discipline Alignment (API Authorization & Contract Enforcement)

Scope: ISAU-DS-AS-1010 API Authorization & Contract Enforcement
Design: Define trust boundaries and invariants (e.g., “explicit authorization on 100% mutating handlers,” “strict contracts at the first boundary,” “idempotency keys on mutating routes”): document decision points and enforcement locations per route.
Implement: Express contracts and gateway policies as code; generate validators in services; enforce deny-by-default for protected resources; require per-request token validation and idempotency keys.
V&V: Run contract/negative suites (100% pass); execute BOLA/BFLA/BOPLA tests; measure authorization coverage; simulate duplicate POST with identical Idempotency-Key; verify logs include trace_id/control_id.
Operate: Evidence Pack includes contract/policy repo history, gate results, authorization coverage reports, deny/validation logs, CSP/CORS/CSRF scans where applicable, incidents, and closed-loop remediation.

Each control requires objective pass/fail criteria, a specified test frequency, a designated responsible owner, and a defined retention policy. Map EP-04 IDs into §12 traceability and keep Test-ID, Owner, Frequency fields with every artifact.

Practitioner Guidance:

- Maintain a Controls → Outputs → Tests sheet: each Table D-4 control maps to §6 output(s), §12 Test-ID(s), enforcement point (code or first boundary), Owner, Frequency, and EP-04 evidence path.
- Update the sheet in the same PR that changes a route, contract, authorization rule, token policy, or header policy; attach proofs and store artifacts under EP-04.
- A change is “done” only when the impacted §12 Test-IDs have passed, and the new evidence is present in EP-04.

Section 11. Associate Sub-Standards Mapping

Purpose of Sub-Standards

ISAUnited Defensible Sub-Standards under Application Security Architecture & Secure Development are tightly scoped, engineering-driven extensions that:

- Define granular, application-layer requirements (ASR-IDs) for specialized domains.
- Translate architectural intent into enforceable behaviors in code and at app boundaries (contracts/gateways).
- Specify verification/validation methods that yield test artifacts (unit/contract/abuse) referenced in §12.
- Align directly to the Parent Standard's §6 outputs and §7 principles, with traceable evidence.

Interface notes (non-normative):

- Annex D produces app-layer requirements, enforcement points, and tests.
- Annex J ensures those tests run in CI/CD and at promotion; SBOM/provenance and gates live there.
- Annex I (CEK) governs crypto parameters and key lifecycles; Annex D governs correct use in code.

Scope and Focus of Application Security Sub-Standards

Sub-Standards developed under this Parent Standard will address specialized areas, including but not limited to:

Secure API Authorization & Contract Safety
Example Sub-Standard: ISAU-DS-AS-1010 – API Authorization & Gateway Contract Enforcement

- Object/field/function-level authorization (RBAC/ABAC/ReBAC) for 100% mutating handlers.
- Strict contract/schema validation (OpenAPI/JSON-Schema/Proto) with unknown-field reject and bounds checks.
- Idempotency keys for mutating HTTP routes; gateway/edge policy must mirror code semantics.
- Maps to §6: 6.1, 6.2, 6.5
- Tests: authorization (BOLA/BFLA/BOPLA), contract, idempotency.

Secure Coding & Code Review Standard
Example Sub-Standard: ISAU-DS-AS-1020 – Secure Coding & Review (Validation/Encoding/Deserialization)

- Canonicalize→Validate→Authorize→Encode sequence at every boundary.
- Safe serialization/deserialization (type allowlists, size/time limits; gadget resolution disabled).
- Context-correct output encoding (HTML/JS/CSS/URL/SQL params).

- Maps to §6: 6.3
- Tests: negative/fuzz/grammar tests; sink-focused unit tests.

Application Dependency Governance & Component Safety
Example Sub-Standard: ISAU-DS-AS-1030 – Library/Framework Usage & Unsafe Pattern Elimination

- Approved library lists and version constraints for parsers, template engines, JSON/XML libs, and crypto calls.
- Prohibit insecure APIs/patterns (eval/exec, unsafe reflection, raw SQL concatenation).
- Application-side inventory and policy (Annex J handles SBOM/provenance enforcement).
- Maps to §6: 6.3, 6.4
- Tests: static pattern checks; targeted unit tests for risky call sites.

Application Data Protection & Privacy Engineering
Example Sub-Standard: ISAU-DS-AS-1040 – Data Classification, Minimization, Masking/Tokenization in Code

- Data element classification; collection/retention minimization; masking/tokenization rules.
- In-code cryptographic use via CEK-approved primitives; logging redaction with error semantics.
- Maps to §6: 6.4, 6.8
- Tests: data-path unit tests; redaction/format assertions.

Client Interaction & Browser Surface Hardening
Example Sub-Standard: ISAU-DS-AS-1050 – CSP/CORS/CSRF & Clickjacking Defenses

- CSP with nonces/hashes for dynamic content; strict CORS; CSRF protections on state-changing endpoints.
- X-Frame-Options/Frame-Ancestors; X-Content-Type-Options; explicit MIME expectations.
- Maps to §6: 6.6, 6.8
- Tests: header scan, CSP report analysis, CSRF/iframe harness tests.

Abuse Resistance & SSRF Controls
Example Sub-Standard: ISAU-DS-AS-1060 – Rate Limits/Backpressure & Egress Safety for SSRF

- Throttle/block automated abuse ≥ 95% at app boundary; pagination/size/time bounds.
- SSRF mitigations: egress allowlists, metadata/localhost block, protocol/port constraints, DNS pinning.
- Maps to §6: 6.7
- Tests: abuse/credential-stuffing simulations; SSRF block tests.

State Stores, Queues, and Caches Integrity

Example Sub-Standard: ISAU-DS-AS-1070 – Message/Session Integrity, TTL, Replay Control

- Integrity (HMAC/AEAD) for cross-boundary messages; tenant/env isolation; idempotency/de-dupe.
- Schema enforcement with 100% pass rate for queue payloads.
- Maps to §6: 6.9
- Tests: schema validators; replay/idempotency tests.

Table D-5. Example Future Sub-Standards:

Sub-Standard ID	Sub-Standard Name	Focus Area
ISAU-DS-AS-1010	API Authorization & Gateway Contract Enforcement	API authorization & contracts
ISAU-DS-AS-1020	Secure Coding & Review (Validation/Encoding/Deserialization)	Input/serialization safety
ISAU-DS-AS-1030	Library/Framework Usage & Unsafe Pattern Elimination	Dependency governance in code
ISAU-DS-AS-1040	Data Classification, Minimization & Masking/Tokenization	Data protection & privacy in code
ISAU-DS-AS-1050	CSP/CORS/CSRF & Clickjacking Defenses	Client/boundary hardening
ISAU-DS-AS-1060	Rate Limits/Backpressure & Egress Safety (SSRF)	Abuse resistance
ISAU-DS-AS-1070	Message/Session Integrity, TTL & Replay Control	State stores & queues integrity
ISAU-DS-AS-1080	(Optional) RASP / In-App Controls	In-process detection/defense

Sub-Standard ID	Sub-Standard Name	Focus Area

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 Annex D §6 outputs and §7 principles they extend, plus NIST/ISO clauses from §8.
- Technical Peer Review — Evaluate engineering rigor, testability, and clarity of enforcement points.
- Approval & Publication — Assigned identifier, version, and publication as an actionable extension.

Sub-Standard Deliverables (normative)

Each sub-standard Must include:

- **Inputs (Requirements):** Preconditions from Annex D §5, it depends on.
- **Outputs (Specifications):** Concrete application behaviors with thresholds (SLOs).
- **Verification/Validation:** Named tests (unit/contract/abuse) and acceptance criteria tied to §12; Test-IDs, Owner, and Frequency must be declared.
- **Evidence:** Artifact list and storage location (EP-04 or child packs EP-04.x).
- **Standards Mapping:** ASR-ID/Spec → NIST/ISO clause (from §8) → Controls (from §9) → Test-ID → Evidence (EP-04 path).
- **Interfaces:** What is enforced in code/boundary (Annex D) vs. what is executed in delivery (Annex J) and crypto parameters (Annex I).
- **Version pins (when applicable):** If referencing OWASP API Top 10, pin the version year (2019 or 2023) and cite the exact category (e.g., API7/2023 — SSRF).

Practitioner Guidance:

- Start with your app's ASR-IDs and choose the sub-standards that apply; for each ASR-ID, identify the enforcement point in code and at the first boundary (gateway/edge), then assign a Test-ID, Owner, and Frequency, and store artifacts under EP-04 (or EP-04.x).
- Keep a one-page Sub-Standard Readiness Sheet per app: inputs satisfied, outputs targeted, Test-IDs named, artifact paths (EP-04), and clause mappings (from §8) to controls (from §9).

	• If a requirement spills into CI/CD (e.g., “run contract tests on every merge”), reference Annex J rather than duplicating mechanics here. • When citing OWASP API Top 10, pin the version year (2019 or 2023) and category.

Section 12: Verification and Validation

The effectiveness and defensibility of an application security architecture must be continuously verified and validated using structured, engineering-grade assessments. While detailed test requirements for specific stacks will live in Application sub-standards, this Parent establishes the gold-standard expectations below.

> **Verification** confirms the application has been implemented according to this standard’s Requirements (Inputs, §5) and Technical Specifications (Outputs, §6).
>
> **Validation** proves the application performs under real operating conditions and withstands adversarial testing.

Core Verification Activities

- Confirm §6 controls exist and are enforced at the first boundary (gateway/edge) and in code: strict contracts (OpenAPI/JSON Schema/Proto) with unknown-field reject and bounds checks; explicit authorization on 100 % mutating handlers; OAuth2/OIDC token validation per request; idempotency on mutating routes; CSP nonces/hashes; strict CORS; CSRF protections; SSRF egress allowlists for server-initiated outbound requests; structured logging with trace_id/control_id; immutable audit storage.
- Review application coding standards and libraries against approved lists: safe serializers, encoder-at-sink usage, banned APIs/patterns; confirm unsafe deserialization disabled with allowlists and size/time limits.
- Verify integration points and contracts: IdP ↔ app token flows, gateway ↔ service validators, secrets service ↔ application code, telemetry pipeline ↔ SIEM, ensure controls do not break business-critical paths.
- Peer-review architecture diagrams, trust-boundary maps, interface/route ICDs, header policies, gateway policies, authorization decision maps, and control mappings for completeness and accuracy.

Core Validation Activities

- Perform adversarial testing, application/API penetration testing, business-logic abuse probes, and BAS/emulation, focused on: BOLA/BFLA/BOPLA,

injection/XXE, unsafe deserialization, token replay/fixation, SSRF, and abuse throttling.
- Validate runtime resilience with automated and manual methods aligned to credible attack paths (for example, deny unknown fields at gateway; block SSRF to metadata/localhost; throttle credential stuffing; reject duplicate POSTs without Idempotency-Key).
- Test operational resilience: contract rollback to last-known-good, header policy regression detection (CSP/CORS/CSRF), token revocation/rotation drills, and error-template checks for deterministic non-leaking responses.
- Measure performance against targets such as contract pass rate, authorization coverage on mutating handlers, CSP violation rate, abuse throttle/block rate, token revocation latency, and schema-conformance at ingest (100 %).

Required Deliverables

All Verification & Validation efforts must produce documented outputs that include:

1. Test Plans and Procedures — Scope, tooling, and methods for verification and validation phases, including Test-IDs, Owner, and Frequency.
2. Validation Reports — Pass/fail results, residual risk, and prioritized remediation actions tied to §6 outputs and ASR-IDs.
3. Evidence Artifacts — Contract/negative test reports, BOLA/BFLA/BOPLA suite results, token/session drill logs, header scans (CSP/CORS/CSRF), SSRF/abuse simulation logs, structured event samples with trace_id/control_id, and immutability settings—each labeled and stored under EP-04 (or child packs EP-04.x) and referenced in Table D-6.
4. Corrective Action Plans — Time-bound remediation for findings that must be closed before acceptance.

Common Pitfalls to Avoid

- Treating “pen test” as a box-check instead of adversary-aware validation of Annex D invariants (for example, strict contracts, explicit authorization, encoder at sink, SSRF egress control).
- Missing evidence: tests run, but artifacts are not versioned, immutable, or linked to Table D-6/EP-04.
- Skipping continuous validation in dynamic areas (new routes/interfaces, policy/header changes, library/serializer upgrades).

Table D-6. Traceability Matrix: Requirements (§5) to Verification/Validation (§12) and Technical Specifications (§6):

Requirement ID	Requirement (summary)	Verification (build-correct)	Validation (works-right)	Related §6 Outputs
5.1	Threat modeling practice & artifacts	DFDs/trust-boundaries/abuse-case catalog present and versioned; change includes Threat-Model Delta.	Abuse-case scenarios execute; invariants hold at boundaries.	6.1, 6.2, 6.3, 6.6, 6.7
5.2	ASR-ID catalog (application requirements)	ASR-IDs map to enforcement points (code + first boundary) and named Test-IDs.	Tests for each ASR-ID pass; traces show enforcement firing.	6.1–6.10 (as applicable)
5.3	API inventory & contract repository	Authoritative OpenAPI/Schema/Proto in strict mode; unknown-field reject and **SRA** (response schema) configured.	Contract/negative + SRA suite 100 % pass for external routes.	6.2
5.4	AuthN/Authorization baseline	Authn pattern + authorization model defined; decision points identified.	BOLA/BFLA/BOPLA suite 100 % pass; 100 % mutating handlers have explicit authorization.	6.1, 6.5
5.5	Input/serialization & encoding standards	Approved libraries; unsafe APIs banned; sequence defined.	Injection & unsafe deserialization blocked; encoder tests pass.	6.3
5.6	Data classification & privacy-by-design	Data map; minimization/masking/tokenization rules documented.	Redaction tests pass; accidental PII in logs ≤ 0.1 % (7-day window).	6.4, 6.8
5.7	Session & token lifecycle policy	Token/session rules; cookie flags set (Secure/HttpOnly/SameSite).	TTL ≤ 60 min; revocation/rotation ≤ 5 min; replay/fixation blocked.	6.5

Requirement ID	Requirement (summary)	Verification (build-correct)	Validation (works-right)	Related §6 Outputs
5.8	Telemetry & error semantics	Structured event schema enforced; error templates defined.	Event samples include trace_id/control_id; user errors have a correlation ID; upstream immutability verified; ingest conformance = 100%.	6.8
5.9	Abuse-resistance & egress safety	Rate limits/backpressure; SSRF allowlists + metadata/localhost blocks.	Abuse simulations show ≥ 95 % block/throttle; SSRF blocked with evidence.	6.7
5.10	Dependency/component policy (app view)	App-side inventory and version constraints for critical libs.	Static/pattern checks clean; no dangerous API usage.	6.3, 6.4

Evidence guidance

Attach: DFDs; contracts/schemas and SRA test reports; ASR-ID catalog; unit/contract/abuse test results; serializer/encoder audits; token/session drill logs; header scans (CSP/CORS/CSRF); abuse/SSRF simulation logs; structured event samples; immutability settings. Store under EP-04 (or EP-04.x).

How to use this matrix

- Plan: For each §5 requirement, define ≥ 1 verification and ≥ 1 validation tied to §6 outputs; assign Owner and Frequency and record EP-04 locations.
- Execute: Run tests; record SLO met / not met with direct artifact links in EP-04.
- Maintain: On any requirement/enforcement change, update the row and re-run impacted tests.

Practitioner Guidance:

- Start from boundaries: Confirm contracts and explicit authorization at every boundary before deeper tests; then validate code-level checks for depth.
- Name your tests: Give each verification/validation a stable Test-ID and keep it in EP-04 alongside the ASR-ID it proves; include Owner and Frequency.
- Measure four indicators: authorization coverage on mutating handlers, contract pass rate, CSP violation rate, abuse block/throttle rate—review weekly.
- Interface cleanly: If a requirement needs CI/CD execution (for example, “run contract suite on each merge”), reference Annex J for delivery mechanics; keep the semantic requirement and tests here.

Quick Win Playbook:

Title: “Contract + Authorization” V&V Smoke Suite with Fail-Closed Gates

Objective: Prove that strict contracts and explicit authorization are enforced before promotion, and block releases when they are not.

Target: Stand up a “contract + authorization” V&V smoke suite with fail-closed gates for one high-value service (§6.1, §6.2, §12).

Component/System: API gateway (first boundary), CI pipeline stage, application service (OpenAPI/JSON Schema/Proto), test harness.

Protects: The application from BOLA/BFLA/BOPLA, malformed/hostile inputs, and duplicate side-effects on mutating routes.

Stops/Detects: Missing explicit authorization on mutating handlers; unknown fields/out-of-bounds values; unsafe deserialization paths; duplicate POSTs without Idempotency-Key.

Action:

- Enable strict contracts at the gateway (unknown-field reject, numeric/enum bounds); generate and wire validators in the service.
- Implement explicit authorization checks at object/field/function scope on all mutating handlers; require Idempotency-Key on POST/PUT/PATCH.
- Add a CI “V&V smoke” stage that runs: (1) contract/negative tests, (2) BOLA/BFLA/BOPLA suite, (3) idempotency duplicate-request test.
- Execute a staging run: valid request → allow; unknown field/out-of-bounds → 4xx; duplicate POST (same Idempotency-Key) → no second side-effect; unauthorized object access → deny.
- Set gates to fail-closed on any test failure; capture logs with trace_id/control_id.

	Proof: Contract manifest; gateway policy export; CI job log with Test-IDs; deny/validation logs; authorization coverage report → EP-04.3. **Metric:** Contract/negative tests pass rate = 100 % for targeted routes; 100 % mutating handlers covered by explicit authorization tests; duplicate-request test prevents second side-effect; all denials logged with trace_id/control_id. **Rollback:** Temporarily set the CI gate to warn-only for the service and revert the validator/authorization commit; record the exception with the owner and expiry in EP-04.3.

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 must translate the Parent's intent into operational behaviors that are testable, automatable, and auditable for Application Security Architecture & Secure Development (Annex D). Delivery mechanics (pipeline orchestration, SBOM/provenance, promotion/rollback) are governed by Annex J.

Purpose of This Section in Sub-Standards

Sub-standards must use Implementation Guidelines to:

- Translate architectural expectations from the Parent into enforceable run-time and first-boundary (gateway/edge) behaviors (e.g., strict contracts, explicit authorization, CSRF/CSP/CORS, SSRF egress allowlists).
- Provide stack-agnostic practices that improve adoption, reduce 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 (as code) that enforce controls, trust models, and engineering discipline across front ends, API gateways, services/microservices, serverless functions, queues, data stores, IdP, secrets, and telemetry.

Open Season Guidance for Contributors

Contributors developing sub-standards Must:

- Align all guidance with this Parent's strategic posture and §6 outputs.

- Avoid vendor/product terms; express controls as requirements, tests, and evidence.
- Include lessons learned (what fails, why, and how the test proves it).
- Focus on repeatable engineering patterns (contracts/policies-as-code), not one-offs.
- Provide a minimal Standards Mapping (Spec/Control → NIST/ISO clause from §8 → Evidence Pack ID).

Technical Guidance

A. Organizing Principles (normative)

1. Everything as code — Contracts (OpenAPI/JSON Schema/Proto), gateway policies, header policies (CSP/CORS), CSRF protections, SSRF egress allowlists, logging schemas, and runbooks Must be version-controlled, peer-reviewed, and promoted on protected branches.
2. Gated change — Every merge/release Must pass non-bypassable security gates tied to §6 and §12 acceptance criteria (for example, contract/negative tests = 100 %, explicit authorization coverage = 100 % on mutating handlers, CSP/CORS/CSRF scans clean, SSRF allowlist tests pass, SRA checks pass).
3. Immutable, reproducible releases — No manual policy or code changes post-build; releases Must be reproducible and verified at the first boundary and in code.
4. Least privilege & JIT (application context) — Service identities, automation accounts, and admin functions Must be scoped; step-up/MFA for high-impact actions; error templates and logs Must preserve confidentiality while remaining diagnostically useful.
5. Environment parity — Staging Must mirror production controls (contracts, authorization, headers, SSRF, logging schema) so test results are predictive; drift Must be monitored and reconciled; telemetry ingest must meet schema-conformance = 100 % in staging.

B. Guardrails by Pipeline Stage (normative)

1. **Pre-commit / local**
   - Secrets scanning and signed commits required.
   - Pre-commit hooks Should generate/validate API contracts and run encoder/serializer linters; block unsafe APIs/patterns.
2. **Pull request (PR) / code review**
   - CODEOWNERS approval required; record a Threat-Model Delta for significant boundary or contract changes.
   - Contract/negative gate for all changed routes; Critical findings = 0.
   - Response schema alignment (SRA) tests for changed routes must pass
   - Authorization coverage check for changed mutating handlers; encoder-at-sink checks; evidence pointers in PR (planned §12 Test-IDs and Evidence Pack ID stub).

3. **Build & package**
   - Deterministic artifacts (pinned libraries; no ad-hoc fetch at deploy); integrity checks for contracts/policies-as-code.
   - Generate validators from contracts; package header policies and SSRF allowlists as deployable config.
4. **Pre-deploy / release**
   - Config drift detection against approved contracts/policies; approvals “as code.”
   - Progressive rollout (staged/canary) for gateway/headers/SSRF rules with health SLOs and automatic rollback; include SRA tests for changed routes.
   - Positive/negative traffic-contract tests for external and inter-service flows; idempotency tests on mutating routes.
5. **Deploy & runtime**
   - Enforce strict contracts at the first boundary (unknown-field reject; numeric/enum bounds) and uphold in code.
   - Per-request token validation (issuer/audience/scope); CSRF protections on state-changing endpoints; CSP with nonces/hashes; explicit, minimal CORS.
   - SSRF controls: egress allowlists, metadata/localhost blocks, protocol/port constraints.
   - Unified logging schema (ts, actor, action, resource, result, trace_id, control_id, data_class, error_code); logs to immutable storage with authenticated time sync.
6. **Post-deploy validation & operations**
   - Continuous validation: contract/negative suites, BOLA/BFLA/BOPLA tests, header policy scans, abuse/SSRF simulations on a schedule.
   - Track Security SLOs: contract pass rate, authorization coverage, CSP violation rate (≤ 0.1% over 7 days), abuse throttle/block rate (≥ 95%), token revocation latency, schema conformance at ingest (100%).
   - Auto-generate child Evidence Pack(s) per release (**EP-04.x**) with policy/contract/SRA diffs, validation results, deny logs, token/session drill logs, header scan outputs, SSRF/abuse simulation logs, and ADR links.

C. Identity, Tokens, and Secrets (normative alignment to §6.4–§6.6, §6.8)

- Validate OAuth2/OIDC tokens per request; require PKCE for public clients; define rotation/revocation drills and record latency.
- Secrets never in repos or images; inject at runtime via approved services with audit trails; redact in logs.
- Error templates Must be deterministic, non-leaking, and include correlation IDs; telemetry Must meet the required schema.

D. Application Supply-Chain Integrity (normative; mechanics in Annex J)

- Only deploy artifacts whose contracts/policies and code passed gates; restrict sources and namespaces.

- Quarantine and verify third-party packages; enforce license and integrity checks.
- Separate build and deploy identities; forbid production writes from build jobs; treat contract/policy tamper as a release-blocking event.

E. Measurement & Acceptance (aligned to §6 and §12)

- Contracts & Boundary: strict-mode enforcement; SRA (response) schemas enforced; contract/negative pass on external routes = 100 %; idempotency on mutating routes.
- Authorization: explicit decisions on 100 % mutating handlers; BOLA/BFLA/BOPLA suite = 100 % pass.
- Client Surface: CSP nonces/hashes enabled; CSP violation rate ≤ 0.1 % over a 7-day window; CORS minimal/explicit; CSRF protections verified.
- Abuse/SSRF: ≥ 95 % throttled/blocked at boundary; SSRF egress controls verified by tests.
- Logging & Evidence: schema-conformant events at ingest = 100 %; immutable retention; every change linked to EP-04 (trace §5 → §6 → §12).

Common Pitfalls (and the engineered countermeasure)

1. Pipelines as suggestions → Enforce non-bypassable gates; block merges/releases on fails; keep failing artifacts as proof.
2. One-time scanning → Treat checks as recurring gates; require coverage for changed items and boundary enforcement events.
3. Manual hot-fixes/drift → Detect & reconcile drift; forbid out-of-band edits; require ADRs and rollback plans.
4. Open egress / unvetted outbound calls → Enforce SSRF allowlists and protocol/port constraints; test them.
5. Weak headers and leaky errors → Enforce CSP/CORS/CSRF and deterministic error templates with correlation IDs.
6. Unsafe serialization/encoding → Ban unsafe serializers; require encoder-at-sink checks.
7. No evidence → Every release Must have an Application Evidence Pack ID with linked tests and results.

Practitioner Guidance:

- Start from ASR-IDs: for each requirement, identify the enforcement point (gateway policy, handler, serializer) and the named tests (unit/contract/abuse) proving it; record **Test-ID, Owner, and Frequency**, and link artifacts to **EP-04** (or **EP-04.x**).

	• Prefer first-boundary controls (contracts/headers) and retain code checks for depth; ensure strict-mode contracts at the gateway and encoder-at-sink in code; keep all artifacts under EP-04. • Track four indicators weekly: authorization coverage (mutating handlers), contract pass rate, CSP violation rate, and abuse throttle/block rate. *(Optional secondary: token revocation latency; schema conformance at ingest.)* • When routes/contracts/trust boundaries change, include a Threat-Model Delta in the PR and update affected tests and evidence links. • If enforcement needs CI/CD or runtime promotion mechanics, reference Annex J; for cryptographic parameters and key lifecycles, reference Annex I (CEK).

Quick Win Playbook:

Title: PR and Pre-Deploy Security Gates for Route/Contract Changes

Objective: Stop unsafe route or contract changes at review time by enforcing non-bypassable PR and pre-deploy gates.

Target: Wire non-bypassable PR and pre-deploy gates for one high-value service (§13.A.2, §13.B.2–4; §6.1, §6.2; §12).

Component/System: Repo (contracts/policies-as-code), CI checks, API gateway (first boundary), application service (OpenAPI/JSON Schema/Proto), test harness.

Protects: Prevents schema drift, missing authorization on mutating handlers, and unsafe changes from reaching production.

Stops/Detects: Unknown-field/bounds violations; mutating routes without explicit authorization tests; header regressions (CSP/CORS/CSRF); missing Test-IDs/EP links.

Action:

- Add CODEOWNERS and a Threat-Model Delta template to the repo; require it on PRs that change routes/contracts.
- Enable strict mode at the gateway (unknown-field reject, numeric/enum bounds checks); generate and wire validators in the service.
- Require Idempotency-Key on POST/PUT/PATCH; implement explicit object/field/function authorization on all mutating handlers.
- Add PR gates: (1) contract/negative tests = 100 % pass, (2) authorization coverage report = 100 % on mutating handlers, (3) header scan (CSP/CORS/CSRF) clean for changed paths, (4) SRA tests pass, (5) Test-ID + EP-04 link present in the PR.
- Add pre-deploy canary: replay contract/negative + SRA suites and header scan against the canary; block promotion on failures.

	Proof: CODEOWNERS + PR template diffs; gateway policy export; validator commit; CI job logs with Test-IDs; authorization coverage and header/SRA scan results → EP-04.4. **Metric:** 100 % PRs touching routes/contracts pass gates; contract/negative/SRA = 100 %; 100 % mutating handlers covered by explicit authorization tests; CSP violation rate ≤ 0.1 % over a 7-day window for changed paths; zero promotions with failed gates. **Rollback:** Temporarily set the service gates to warn-only and revert the validator/authorization/header-policy commit; record an exception with the owner and expiry in EP-04.4.

Appendices

Appendix A: Engineering Traceability Matrix (ETM)

Req ID	Requirement (Inputs) (§5)	Technical Specifications (Outputs) (§6)	Core Principles (§7)	Control Mappings (§9)	Verification – Build Correct (§12)	Validation – Works Right (§12)	Evidence Pack ID
5.1	Threat modeling practice & artifacts	§6.1 Identity & Authorization; §6.2 Contract Enforcement; §6.3 Input/Serialization; §6.9 State Stores	RP-05 Secure by Design; RP-03 Complete Mediation	OWASP ASVS V1 Architecture; CSA CCM AIS	DFDs, trust-boundary maps, and abuse-case catalog present and versioned	Abuse-case scenarios execute, and invariants hold at boundaries	EP-04
5.2	ASR-ID catalog (application requirements)	§6.1–§6.10 (all relevant outputs)	RP-05 Secure by Design; RP-15 Evidence Production	CIS 16.x; OWASP ASVS V4 & V5	ASR-IDs map to code + first boundary enforcement + Test-IDs	Tests for each ASR-ID pass; traces show enforcement firing	EP-04
5.3	API inventory & contract repository	§6.2 API Boundary & Contract Enforcement; §6.9 State Stores	RP-03 Complete Mediation; RP-06 Minimize Attack Surface	OWASP ASVS V5; CSA CCM AIS	Strict-mode schema validation enabled; unknown fields rejected	Contract/negative suite = 100% pass on external routes	EP-04.1
5.4	Authentication & Authorization baseline	§6.1 Identity & Authorization; §6.5 Session & Token	RP-01 Least Privilege; RP-02 Zero Trust; RP-03 Complete Mediation	OWASP API1/API5/API2; CIS 6.x	Authn model + decision map validated; explicit object/field/function auth present.	BOLA/BFLA/BOPLA suite = 100% pass; 100% mutating handlers have explicit authorization	EP-04.1
5.5	Input/serialization & encoding standards	§6.3 Input, Serialization & Output Encoding	RP-04 Defense in Depth; RP-06 Minimize Attack Surface	OWASP ASVS V5; ASVS V12 (files/resources)	Canonicalization + validation pipeline present; unsafe deserialization disabled; banned API list enforced	Injection & unsafe-deserialization tests clean; sink-focused encoder tests pass.	EP-04.2
5.6	Data classification & privacy-by-design	§6.4 Data Protection in Code Paths; §6.8 Telemetry & Errors	RP-18 Confidentiality; RP-19 Integrity	CSA CCM DSI; CIS 3.x	Data map, minimization & masking rules validated	Redaction tests pass; accidental PII ≤ 0.1% over 7 days	EP-04.4

Req ID	Requirement (Inputs) (§5)	Technical Specifications (Outputs) (§6)	Core Principles (§7)	Control Mappings (§9)	Verification – Build Correct (§12)	Validation – Works Right (§12)	Evidence Pack ID
5.7	Session & token lifecycle policy	§6.5 Session & Token Security	RP-02 Zero Trust; RP-10 Secure Defaults	OWASP ASVS V2, V3; CIS 6.x	Token/session rules validated; cookie flags set; TTL/rotation tests present	Token replay/fixation blocked; revocation honored ≤5 minutes	EP-04.1
5.8	Application-layer telemetry & error semantics	§6.8 Telemetry & Errors	RP-15 Evidence Production; RP-16 Make Detection Easier	CIS 8.x; CSA CCM DCS	Structured schema validated; error templates defined; immutability configured	100% ingest schema conformance; correlation IDs appear in logs; upstream immutability confirmed	EP-04
5.9	Abuse-resistance & SSRF controls	§6.7 Abuse Resistance & SSRF Controls	RP-06 Minimize Attack Surface; RP-04 Defense in Depth	OWASP API8; ASVS V9	Rate-limit/backpressure and SSRF allowlists validated	≥95% automated abuse throttled/blocked; SSRF blocked with evidence	EP-04.6
5.10	Dependency & component governance	§6.3 Input/Serialization; §6.4 Data Protection; §6.10 RASP (optional)	RP-06 Minimize Attack Surface; RP-10 Secure Defaults	CIS 16.x; OWASP ASVS V5	Dependency inventory enforced; unsafe patterns blocked; static analysis clean.	Library misuse tests pass; no dangerous API calls reachable	EP-04.3

Appendix B: EP-04 Summary Matrix – Evidence Pack Overview

Layer	EP Identifier	Purpose	Evidence Categories Included
Parent EP	EP-04	Stores architecture-wide application-layer evidence supporting §§5, 6, 10, 12.	• DFDs, trust-boundary maps, interface diagrams • ASR-ID catalog • Contract repo references • Invariants register • Logging schema & error templates • Quick Win: Contract Strict Mode Smoke Test (Section 6/12) — first-boundary strict-mode enforcement w/ pass/fail logs • Quick Win: Gateway policy export proofs
Sub-EP	EP-04.1	API Authorization & Contract Enforcement (AS-1010).	• Authorization decision maps • Contract strict-mode validation logs • BOLA/BFLA/BOPLA suite results • Idempotency-Key test results • Gateway policy exports • Quick Win (6.2): Schema reject tests • Quick Win (12): Contract + Authorization V&V Smoke Suite
Sub-EP	EP-04.2	Secure Coding, Input/Serialization Safety (AS-1020).	• Canonicalization tests • Encoder-at-sink validations • Safe deserialization enforcement • Static/semantic analysis evidence • Quick Win: Injection-negative suite • Quick Win: Serializer safety fuzz tests
Sub-EP	EP-04.3	Dependency & Component Governance (AS-1030).	• Dependency inventory • Unsafe pattern detection logs • Allowed/denied API list • Quick Win: Supply-chain vulnerability block report • Quick Win: Library misuse detection tests
Sub-EP	EP-04.4	Data Protection & Privacy Engineering (AS-1040).	• Data classification map • Masking/tokenization evidence • Log redaction output • Crypto usage proof • Quick Win: PII leakage scan (target ≤0.1% over 7 days)
Sub-EP	EP-04.5	Client Interaction & Browser Security (AS-1050).	• CSP/CORS/CSRF evidence • Header scans (CSP violation rate ≤0.1%) • XFO/MIME header logs • Quick Win: CSP report-only → enforced transition test • Quick Win: Header regression smoke test
Sub-EP	EP-04.6	Abuse Resistance & SSRF Controls (AS-1060).	• Throttle/backpressure logs • SSRF block logs • Egress allowlist enforcement • Quick Win: SSRF smoke-test with allowed/blocked cases • Quick Win: Abuse simulation (≥95% blocked)

Layer	EP Identifier	Purpose	Evidence Categories Included
Sub-EP	EP-04.7	State Stores, Queues & Cache Integrity (AS-1070).	• HMAC/AEAD evidence • TTL/replay/idempotency validations • Schema validation logs • Quick Win: Duplicate-request suppression test
Sub-EP	EP-04.8	Optional: RASP / In-App Runtime Controls (AS-1080).	• RASP block/report samples • Latency impact measurements • Quick Win: RASP rule simulation
Future Sub-EPs	EP-04.9+	Reserved for future sub-standards.	• Will inherit the same EP structure, including Quick Win mapping.

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.**
(Parent → 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 shall 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
December 2025	Standards Revision	Standards Committee	Publication	Pending
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.