ENGINEERING STANDARDS LIBRARY / SOFTWARE ENGINEERING

Desktop Architecture and Tauri/Svelte Integration Standard

Secure desktop runtimes, Rust-native command boundaries, Svelte application state, capability control, and update integrity.

PERMANENT DOCUMENT ID

MYTHOS-SWE-0006

PUBLICATION

Candidate Standard
Version 1.0.0-candidate

ASSURANCE PROFILE

Standard / Advanced
High Assurance

PERMANENT PUBLICATION IDENTITY

Desktop Architecture and Tauri/Svelte Integration Standard

DOCUMENT ID
MYTHOS-SWE-0006

VERSION
1.0.0-candidate

STATUS
Candidate Standard

PUBLISHER
Mythos Systems

PUBLICATION DATE
2026-07-24

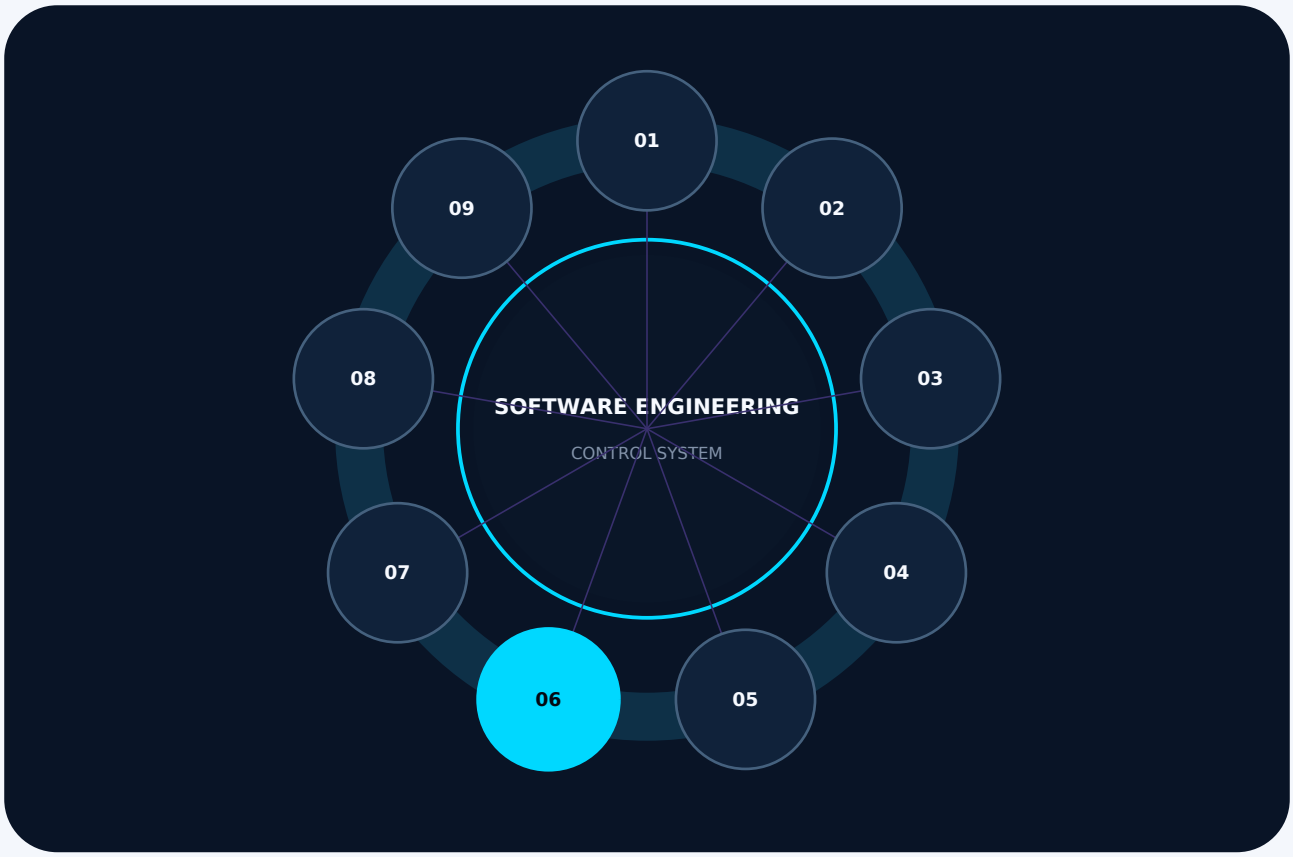
SCHEDULED REVIEW
2027-01-24

17	6	4	1
ATOMIC CRITERIA	ASSURANCE VIEWS	IMPLEMENTATION PROFILES	PERMANENT IDENTITY

Publication doctrine. This publication establishes durable guidance or requirements for its defined scope. Interpretation must preserve the permanent document identity, recorded evidence, controlled exceptions, and formal revision history.

DOMAIN CONTROL SYSTEM

MYTHOS-SWE-0006 inside the software engineering standard constellation



Connected assurance	Architecture, implementation, operations, evidence, and lifecycle are evaluated as one controlled system.
Specialization rule	Companion standards may add precision, but they cannot silently weaken this publication's requirements.

EXECUTIVE POSITION

Desktop Architecture and Tauri/Svelte Integration Standard

The architecture of desktop applications has failed.

Evidence-bearing delivery path

Assurance maturity spectrum

CONTENTS

Publication navigation

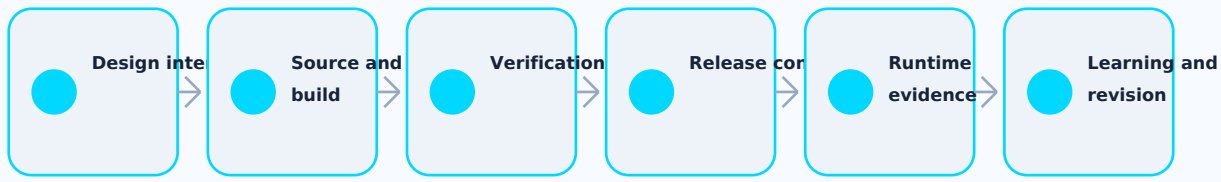
CONTROL TOPOLOGY

Assurance model and evaluation surface

The standard is assessed as an evidence-bearing system. Capability scores do not replace mandatory gates, implementation proof, or operational validation.

Capability family	Evaluation nodes
IPC Security and Renderer Isolation (Weight: 25%)	3
Backend Authority and Data Sovereignty (Weight: 20%)	2
Auto-Updater and Code Signing (Weight: 20%)	2
Frontend Discipline (Svelte) (Weight: 15%)	2
Local-First Data Architecture (Weight: 10%)	1
Performance and Resource Efficiency (Weight: 10%)	1
Suite Composition	2
Additional Evaluation Dimension	4

Lifecycle control loop



EVALUATION INDEX

Criteria and evidence map

#	Capability family	Criterion
1	IPC Security and Renderer Isolation (Weight: 25%)	IPC Validation
2	IPC Security and Renderer Isolation (Weight: 25%)	Tauri Allowlist Enforcement
3	IPC Security and Renderer Isolation (Weight: 25%)	Content Security Policy (CSP)
4	Backend Authority and Data Sovereignty (Weight: 20%)	Secret Isolation
5	Backend Authority and Data Sovereignty (Weight: 20%)	Business Logic Placement
6	Auto-Updater and Code Signing (Weight: 20%)	Update Signature Verification
7	Auto-Updater and Code Signing (Weight: 20%)	Atomic Updates and Rollback
8	Frontend Discipline (Svelte) (Weight: 15%)	Type Safety Across IPC
9	Frontend Discipline (Svelte) (Weight: 15%)	Dependency Minimalism (Frontend)
10	Local-First Data Architecture (Weight: 10%)	Local Storage
11	Performance and Resource Efficiency (Weight: 10%)	Resource Footprint
12	Suite Composition	M-DESK-S-IPC
13	Suite Composition	M-DESK-S-Update
14	Additional Evaluation Dimension	Desktop Dimensions
15	Additional Evaluation Dimension	The Desktop Doctrine
16	Additional Evaluation Dimension	Scoring Model
17	Additional Evaluation Dimension	Suite Composition

NORMATIVE PUBLICATION

Requirements, evaluation methodology, and implementation guidance

The following body preserves the substantive source standard while normalizing publication status, identity, navigation, and visual presentation.

Executive Mandate

The architecture of desktop applications has failed.

For a decade, the industry has defaulted to Electron, bundling an entire Chromium browser and a Node.js runtime into every desktop app. This results in 200MB "Hello World" applications, 1GB+ memory footprints, and a catastrophic attack surface where a single Cross-Site Scripting (XSS) vulnerability in a web view leads to Remote Code Execution (RCE) because the renderer has direct access to the filesystem and OS shell.

This standard exists because:

1. The Renderer is Untrusted: A web view (Chromium/WebView2) processes untrusted data from the internet, local files, and user inputs. Granting it system-level authority is an architectural failure. The renderer MUST be a thin, stateless projection of the backend.
2. Electron is a Liability: Bundling Node.js into the renderer bridges the untrusted web world with the trusted OS. Tauri solves this by using the OS-native WebView and a Rust backend, shrinking the attack surface by orders of magnitude.
3. IPC is a Network Boundary: The bridge between the Svelte frontend and the Rust backend must be treated with the same zero-trust rigor as a public API. All inputs from the renderer MUST be validated; the frontend is assumed compromised.
4. An Unsigned Update is a Backdoor: Auto-updaters that fetch and execute unsigned binaries are the easiest path to compromising a fleet. Updates MUST be cryptographically signed, ideally with PQC-ready algorithms, and verified before execution.

This document establishes the Mythos Desktop Architecture Standard (MDAS), a comprehensive, evidence-based methodology for evaluating Tauri/Svelte desktop applications against IPC security, authority separation, and update integrity.

Scope and Applicability

In Scope

This standard covers the evaluation of:

1. Desktop application architectures (Tauri, Electron, native)
1. Inter-Process Communication (IPC) between frontend and backend
1. WebView isolation and Content Security Policy (CSP)

1. Auto-updater security and code signing
1. Local-first data architecture (SQLite, `sqlite-vec`)
1. Svelte/SvelteKit integration patterns and discipline
1. Native OS integration (filesystem, shell, notifications)

Out of Scope

1. General Rust engineering (covered in MYTHOS-SWE-0002)
1. General web/browser engineering (reserved for MYTHOS-WEB)
1. General local-first sync architecture (covered in MYTHOS-DAT-0001)

Audience

1. Desktop application engineers and architects
1. Security teams evaluating desktop attack surfaces
1. Platform engineers building developer tooling
1. AI governance teams evaluating local agent UIs
1. Standards bodies seeking a reference desktop architecture methodology

---

Definitions and Taxonomy

Desktop Dimensions

Dimension	Definition
Tauri	A framework for building tiny, fast binaries for all major desktop platforms using the OS's native WebView and a Rust backend.
IPC (Inter-Process Communication)	The protocol bridge (e.g., Tauri's `invoke`) allowing the untrusted WebView (Svelte) to request actions from the trusted Rust backend.
Renderer Isolation	The architectural principle that the WebView (renderer) has zero direct access to the OS, filesystem, or network; all access is brokered by the Rust backend.
CSP	Content Security Policy. A browser security standard that restricts what resources the WebView is allowed to load (e.g., no inline scripts, no external domains).

The Desktop Doctrine

> The renderer is untrusted. The backend is the authority. IPC is a network boundary. An unsigned update is a backdoor. A 200MB hello-world is a failure.

Evaluation Methodology

Scoring Model

Each capability is scored from 0 to 5.

Score	Meaning
0	Fails basic task; Electron, Node in renderer, unvalidated IPC
1	Basic Tauri but overly permissive allowlists or no CSP
2	Functional but lacks strict IPC validation or update signing
3	Solid production performance; strict CSP, validated IPC, signed updates
4	Strong performance; PQC updates, local-first encrypted data, typed IPC
5	Best-in-class; sets the standard for zero-trust desktop architecture

Weighting

Category	Weight	Rationale
IPC Security & Renderer Isolation	25%	XSS to RCE is the #1 desktop threat
Backend Authority & Data Sovereignty	20%	The frontend must not hold secrets or logic
Auto-Updater & Code Signing	20%	A compromised updater owns the fleet
Frontend Discipline (Svelte)	15%	Logic in the frontend is unverified logic
Local-First Data Architecture	10%	Cloud-dependent desktops are fragile
Performance & Resource Efficiency	10%	Resource waste is an architectural failure

Total: 100%

A system MUST score at least 3.0 in IPC Security and Auto-Updater to be considered for production use.

Evaluation Categories

IPC Security and Renderer Isolation (Weight: 25%)

IPC Validation

Are all inputs from the frontend validated and sanitized in the Rust backend?

Score	Criteria
0	No validation; frontend data trusted implicitly
1	Basic type checking but no domain validation
2	Strict type validation using Serde/Schemars for all IPC commands
3	Domain validation (e.g., path traversal prevention, command allowlisting)
4	Sandboxed IPC: frontend cannot invoke commands not explicitly registered and typed
5	Perfect security: formally validated IPC schemas, zero implicit trust, WASM-level sandboxing of IPC handlers

Tauri Allowlist Enforcement

Is the Tauri allowlist restricted to the absolute minimum required capabilities?

Score	Criteria
0	`allowlist: { all: true }` or equivalent
1	Broad permissions (e.g., `fs: { scope: ["**"] }`)
2	Scope restricted to specific directories
3	Scope restricted to specific file patterns; shell disabled
4	Granular capability-based permissions (Tauri v2 capabilities)
5	Perfect enforcement: zero wildcard scopes, formally verified minimal surface, dynamic capability grants

Content Security Policy (CSP)

Is a strict CSP enforced on the WebView?

Score	Criteria
0	No CSP; external scripts loadable
1	Basic CSP but allows `unsafe-inline` or `unsafe-eval`
2	Strict CSP; no inline scripts; no eval
3	Nonce-based or hash-based CSP for all inline needs
4	CSP restricts connections to specific backend origins; no external CDNs
5	Perfect CSP: hash-based, zero external connections, formally verified policy, no `unsafe-*`

Backend Authority and Data Sovereignty (Weight: 20%)

Secret Isolation

Are secrets (API keys, tokens, passwords) prevented from entering the frontend?

Score	Criteria
0	Secrets stored in `localStorage` or JS variables
1	Secrets fetched by frontend and passed to backend
2	Secrets stored by backend; references sent to frontend
3	Just-in-time secret brokering; frontend never sees raw secret
4	Hardware-backed key storage (OS Keychain/DPAPI) managed by Rust
5	Perfect isolation: zero secrets in WebView memory, cryptographic attestation of secret usage, PQC-encrypted local vault

Business Logic Placement

Is business logic restricted to the Rust backend?

Score	Criteria
0	Complex business logic in Svelte stores
1	Logic split evenly between frontend and backend

Score	Criteria
2	Logic primarily in backend; frontend handles routing/display
3	Frontend is a "thin client"; only UI state and rendering
4	Typed state synchronization; frontend state is a projection of backend state
5	Perfect placement: frontend is a pure function of backend state, formally verified authority boundaries

Auto-Updater and Code Signing (Weight: 20%)

Update Signature Verification

Are updates cryptographically signed and verified before execution?

Score	Criteria
0	No auto-update or unsigned updates
1	HTTPS-only verification (relies on PKI)
2	Ed25519 signature verification on update binary
3	Signature + Transparency log (Sigstore/Rekor) verification
4	PQC signature support (ML-DSA) for future-proofing
5	Perfect signing: PQC/hybrid signatures, transparency log, hardware-rooted verification, zero unsigned execution

Atomic Updates and Rollback

Are updates atomic and instantly rollback-safe?

Score	Criteria
0	In-place mutation; failed update corrupts install
1	Backup created but manual rollback required
2	Dual-directory layout (current/next); fallback on crash
3	Atomic swap with automatic rollback on startup failure
4	Delta updates with verified patching
5	Perfect updates: atomic, delta, PQC-signed, formally verified rollback, zero user disruption

Frontend Discipline (Svelte) (Weight: 15%)

Type Safety Across IPC

Are types consistent between Rust backend and Svelte frontend?

Score	Criteria
0	Manual type definition; TypeScript and Rust types drift
1	TypeScript types manually synced with Rust Serde schemas
2	TypeScript types auto-generated from Rust (e.g., `ts-rs`, `specta`)
3	Generated types + runtime validation (Zod) on IPC responses

Score	Criteria
4	End-to-end type safety: Rust -> TypeScript -> Svelte stores
5	Perfect discipline: auto-generated, runtime-validated, formally verified schema compatibility

Dependency Minimalism (Frontend)

Is the frontend dependency tree minimal and audited?

Score	Criteria
0	Heavy framework (Next.js on desktop); massive node_modules
1	SvelteKit but with many third-party components
2	Minimal Svelte; no heavy UI libraries
3	No external runtime dependencies; only dev-dependencies
4	Sub-resource integrity for any loaded assets
5	Perfect minimalism: zero runtime npm dependencies, audited dev-dependencies, strictly typed

Local-First Data Architecture (Weight: 10%)

Local Storage

Is user data stored locally-first, encrypted, and sovereign?

Score	Criteria
0	Cloud-only storage; no offline capability
1	Local SQLite but unencrypted
2	SQLite with SQLCipher encryption at rest
3	Local-first sync architecture (`sqlite-vec` for vectors)
4	CRDT-based synchronization for multi-device
5	Perfect local-first: encrypted SQLite, `sqlite-vec`, CRDT sync, zero data loss, formally verified consistency

Performance and Resource Efficiency (Weight: 10%)

Resource Footprint

Does the application respect system resources (RAM, CPU, binary size)?

Score	Criteria
0	>500MB RAM idle; >200MB binary; >5s startup (Electron baseline)
1	<200MB RAM idle; <100MB binary; <3s startup
2	<100MB RAM idle; <50MB binary; <2s startup
3	<50MB RAM idle; <20MB binary; <1s startup
4	<20MB RAM idle; <10MB binary; <500ms startup
5	Perfect efficiency: <10MB RAM idle, <5MB binary, <200ms startup, formally verified resource bounds

The Mythos Desktop Suite (M-DESK-S)

Traditional desktop benchmarks measure feature completeness. The M-DESK-S evaluates IPC isolation, update security, and resource discipline.

Suite Composition

M-DESK-S-IPC

1. XSS to RCE: 100+ tests injecting malicious payloads into WebView inputs to verify IPC validation prevents execution.
1. Allowlist Violation: 50+ tests attempting filesystem/network access from unregistered IPC commands.

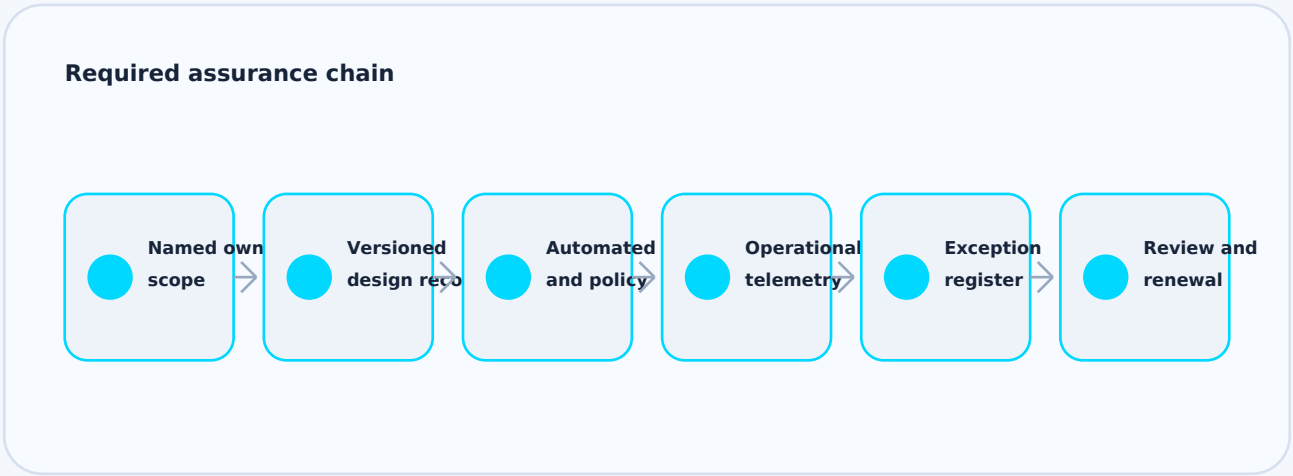
M-DESK-S-Update

1. Malicious Update: 50+ tests serving tampered binaries to verify the updater rejects them.
1. Rollback: 50+ tests interrupting updates to verify atomic rollback.

Execution Protocol

ASSURANCE AND ADOPTION

Implementation evidence and review gates



A passing score is not sufficient without current evidence. High-assurance adoption requires traceable design decisions, reproducible builds, independently reviewable tests, controlled secrets, runtime observability, failure exercises, and a dated exception register.

Current authority register

Authority	Version / snapshot	Applicability	Review by
Rust Project: The Rust Programming Language and Rust Reference	Rolling official documentation; book assumes Rust 1.90.0 or later and Rust 2024 Edition	High	2026-10-22
Go Project: The Go Programming Language Specification and Effective Go	Rolling official documentation snapshot	High	2026-10-22
Python Software Foundation: Python Language Reference, Packaging, Typing, and PEP Index	Rolling official documentation snapshot	High	2026-10-22
Tauri Programme: Tauri 2 Documentation	Tauri 2 rolling documentation snapshot	High	2026-10-22
Svelte Project: Svelte Documentation	Svelte 5 rolling documentation snapshot	High	2026-10-22
TLA+ Foundation: TLA+ Language and Tools Documentation	Rolling documentation snapshot	High	2027-01-24
OpenAPI Initiative: OpenAPI Specification	3.2.0	High	2027-01-24
OpenTelemetry Project: OpenTelemetry Documentation and Specification	Rolling specification snapshot; profiles public alpha and declarative configuration stable in 2026	High	2026-10-22

Research limitation. Official documentation establishes published interfaces and declared behavior. It does not prove the performance, security, interoperability, cost, or operational suitability of a specific implementation. Those claims require controlled testing and retained evidence.