TINL — Threat Model and Safeguards (public cut)

Author: Dennis DZ Zweigle, Founder & Chief Architect, A3E9 **Date:** 17 August 2026
Document type: Threat model **Product:** A3E9 TINL — Transaction Intent Normalization Layer

---

The claim this document exists to prevent

A person who installs a security extension concludes: *I installed a security extension, therefore I am safe on bad sites.*

For TINL today that conclusion is **false**, and the gap between it and reality is the most important thing in this document. TINL hard-stops one class of decision and gives advice on the rest. Advice can be ignored, and on a hostile page some of it can be undermined.

TINL does not intercept every transaction, and no percentage of coverage is claimed. Browser script ordering can leave a wallet connection unwrapped — a dApp that cached its provider before TINL loaded, or a Solana Wallet Standard race — and TINL never sees those requests at all. A safeguard that is sometimes not on the path is not a guarantee, and this document does not present it as one.

---

1. What TINL is

TINL decodes what a transaction will actually do — the raw ABI calldata or Solana instruction — and evaluates it against a tiered policy profile **before** the wallet ever prompts for a signature. The decision is one of four:

Decision	Meaning
BLOCK	Refuse. The request does not proceed through TINL's path.
WARN	Proceed only after the user consents to a stated risk.
NORMALIZE	Offer a safer substitute (for example, a capped approval instead of an unlimited one).
ALLOW	No policy objection.

The policy core is written in Rust and compiled to WASM, so the evaluation path is not reimplemented ad hoc in page JavaScript.

2. What TINL is not

System	Relationship to TINL
HSM `a3e9-attestation` (daemon integrity, Merkle re-verify, runtime watchdog)	**Not used by TINL.** It applies to the Linux signing daemon. TINL does not run it, at any frequency. Porting it into a browser is an explicit non-goal — wrong runtime, false assurance.
Chrome Web Store package signing	Google's package authenticity and update channel. Complements TINL; does not enforce policy.
TINL's HMAC audit chain	Tamper-evident decision history *after* evaluation. Evidence, not a substitute for a hard BLOCK.

Price Impact is not a fourth product. It is a rule surface inside the same engine — constant-product AMM reserve maths feeding the same four decisions.

Chrome and Android are separate tracks with separate architectures, separate test runs, and separate limits. Completing one does not complete the other, and their results must not be pooled.

---

3. Decision strength — the load-bearing table

Against a **hostile page**, the four decisions are not equally strong. This is the distinction a reader should carry away:

Decision	Strength against a hostile dApp
BLOCK	**Enforced.** It throws without waiting on any page message, and the unwrapped provider is reachable only through a closure the page cannot access. A blocked request's signature does not reach the wallet through TINL's hold path.
WARN	**Advisory.** Treat user consent as capable of being undermined by a hostile page.
NORMALIZE	**Advisory**, on the same basis.
ALLOW	Depends on the path; bytes must be re-evaluated rather than trusted from a prior decision.

A change landed on 2026-08-08 moved WARN/NORMALIZE consent *resolution* off page-forgeable `window.postMessage` and onto a MessagePort held only by the extension's own scripts, which closes the specific forgery route on that channel. This document deliberately does **not** upgrade WARN and NORMALIZE to "enforced" on the strength of it: the internal threat model still carries that hardening as an open roadmap item in its own summary, the

provider-wrap races in section 5 are unaffected by it, and the honest public position while those two statements disagree is the conservative one. **The wallet's own UI remains the backstop.**

---

4. Attacker goals and mitigations

Vector	Example	Mitigation today
Hostile dApp	Unlimited `approve`; forged consent	BLOCK for catastrophic rules; advisory disclosure elsewhere
Payload swap	Calldata changed after the decision	Digest binding of the exact bytes is a **target**, not shipped
Fake or sideloaded extension	Lookalike package	Retail install via Web Store only; unpacked builds are dev/UAT
Tampered package or WASM	Modified policy engine	Store signing today; load-time WASM hash pinning is **roadmap**
Poisoned dependencies	Compromised npm/cargo crate	Lockfiles, CI, audit discipline
RPC or oracle lies	Fake reserves producing a wrong price impact	Fail closed on decode/RPC errors; AMM rules rely on RPC honesty
Click fatigue / UX spoof	Fake wallet UI	Distinct TINL UI; rate-limited warnings; escalation to BLOCK
Audit wipe	Storage cleared after a drain	Institutional Mode B audit; chain break fails closed

---

5. Safeguards in place today

Stated without overclaiming:

1. Policy core in Rust compiled to WASM — one evaluation path.
2. **BLOCK** is a hard deny against a hostile page, and is tested as such.
3. HMAC-chained audit in both modes, appending fail-closed where implemented.
4. A missing or malformed profile fails closed — it never silently allows.
5. MV3 isolation: service worker plus an isolated relay, with extension CSP.
6. An end-to-end test that documents the known forge path, so it cannot regress silently.

The residual that matters most: provider-wrap races. TINL wraps `window.ethereum` from the page's main world because that is the only place it can be done, and a dApp that grabbed its provider first — or won a Wallet Standard race — is never seen by TINL. No policy engine can decide on a request that never reaches it.

---

6. Roadmap

ID	User-visible outcome	Status
R1	A hostile site cannot fake "user clicked Proceed"	Consent resolution moved off page-forgeable messaging 2026-08-08; digest-bound proceed still open
R2	Fewer "wallet signed without TINL noticing" cases	Partial — traps and EIP-6963 today; residual races remain
R3	Harder to replace the engine with a tampered build	Planned — load-time WASM hash pin
R4	Brand-new sites treated more carefully	Planned — origin-strict defaults, optional WARN→BLOCK
R5	A phone path with no page-script races	Scaffolded and demoable on Android

On R5, precisely. The Android build's consent UI is native and in-process: there is no main world to race and no page to forge a consent from. That property comes from *where the consent UI lives*, not from where the engine runs. But evaluation in that build is **Mode B — a local TINL service over REST**; on-device Mode A is not wired, and the app says so itself. It must not be described as embedded or offline evaluation. Live WalletConnect relay pairing is stubbed in v0.

---

7. The one-paragraph version

> *TINL hard-enforces **BLOCK** decisions on the Chrome extension path. **WARN** and **NORMALIZE** should be treated as **advisory** against a hostile page, with the wallet's own UI as the backstop. Some wallet connections are never wrapped at all because of browser script ordering, so TINL is not a complete intercept and no coverage percentage is claimed. Continuous binary attestation applies to the A3E9 **HSM signing service**, not to this extension. Retail package integrity is intended through Chrome Web*

> *Store distribution, with WASM load-time pinning on the roadmap. The Android track is architecturally stronger on consent and weaker on evaluation mode, and is a separate track with separate results.*

The full acceptance procedures for both tracks — every scenario and the decision expected at each step — are published, without a login, at https://a3e9.com/tinl/uat-guide and https://a3e9.com/tinl/mobile-uat-guide.
