RogueOS: A Deterministic, Locally Executing Autonomous Agent Framework

White Paper v1.2

Stephen Zeitvogel, Chief Architect and Founder

PROVENANCE

Architecture first authored and filed December 2025. USPTO provisional applications 63/828,137 and 63/864,057. White Paper v1.2 published, superseding v1.1.

"*We did not build RogueOS for hype or headlines. We built it so hard working people do not get blamed for machines they cannot explain.*"

Abstract

RogueOS guarantees that every action a governed agent performs can be reproduced bit for bit, proved against policy, and explained in plain language. The guarantee survives network loss, cloud outage, host compromise, and adversarial litigation. The framework is built from eight governance primitives. Three are foundational: a Write-Ahead Log, an immutable permission lattice, and a zero non-determinism sandbox. Five compose on top of that foundation: the Builder's Permit authorization kernel, the Governance Engine, the Evidence Ledger, the Trust Fabric, and Persistent Runtime Identity. Together these primitives satisfy Federal Rules of Evidence 901 and 902 for self-authenticating digital records, while remaining deployable on a fan-less mini-PC under a workbench. This version adds the framework's containment posture and its behavior against an anti-forensic adversary whose first objective is destroying the record.

1. The So-What in One Sentence

If your AI cannot pass a crash test in court, do not ship it on a shop floor.

2. Problem Restatement, No Jargon

Modern AI assistants are helpful but unpredictable. Ask twice, get two answers. Regulated users, meaning contractors, clinics, and insurers, need the same answer every time plus a receipt they can show an auditor. RogueOS provides the receipt and the repeatability. It also provides something the current market does not lead with: when an attacker reaches the machine, the record of what the governed agent did cannot be silently rewritten.

3. Core Vocabulary, Retail Gloss

Write-Ahead Log (WAL). A notebook that must be written before anything happens. Think of a ship's log: the officer signs, then sails. No back-dating is possible.

Deterministic. Given the same starting page and the same rules, the story ends the same way every time you re-read it.

Permission Lattice. A family tree of what is allowed. Nothing outside the tree can execute.

Ambient authority. The ability to act simply because you are running, without asking permission first. RogueOS removes this from the agent entirely.

FRE 901 and 902. United States court rules that let a digital record enter evidence without a human witness, provided the record is produced by a process shown to be reliable.

4. System Snapshot

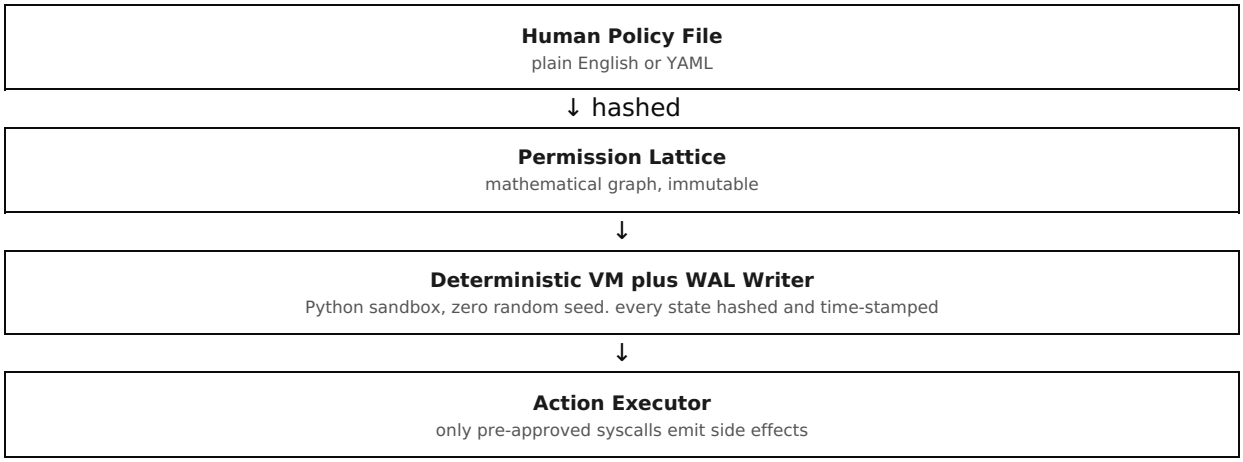

Figure 1. Control flows downward. No stage can act on behalf of the stage above it without a hashed, logged authorization.

4a. What Determinism Means Here

The governance path is deterministic and replayable. Model inference is not, and RogueOS does not pretend otherwise. Model inference is captured as a recorded input-output pair: the prompt hash, model hash, parameters, and seed are written to the log before any side effect. Replay verifies the decision chain, not the model's regenerated text. This scoping is deliberate. Honest boundaries are the credibility play for insurers and standards bodies, and they make the strong guarantees below believable rather than promotional.

5. Formal Invariant

Let S_0 be the initial machine state, P the policy graph, and I an input stream. RogueOS computes a trace T such that T = exec(S_0, I, P), and for all t in T:

(a) hash(t.state) is recorded in the WAL before t.action emits side effects.
(b) t.action is a member of allowable(P, t.state).
(c) replay(S_0, I, P) is identical to T.

The system is fail-closed. If any clause cannot be satisfied, the action does not execute, the kernel panics, and the WAL tail is cryptographically sealed. The default state of an ungoverned or unverifiable action is denial, not permission.

6. Deep Dive on the WAL

Imagine a cashier who must write the sale in the ledger before opening the till, sign every line with a tamper-evident sticker, and photocopy the ledger page every minute into a locked cabinet. RogueOS does exactly this for machine states. Each line is a JSON object of the form { time, hash_prev, hash_state, action, policy_keyid }. The sticker is an ECDSA signature produced by a hardware key that refuses to sign twice with the same one millisecond timestamp. The locked cabinet is an append-only file rotated every hour to Write-Once-Read-Many storage. The result: even root-access malware cannot rewrite history without breaking the signature chain, and that break is detectable in under one second on reboot.

7. Determinism Safeguards

Source of non-determinism	RogueOS neutralizer
Random seeds	Fixed seed hashed from (WAL_prev concatenated with input)
System clocks	Logical Lamport clock stored in WAL
Thread scheduling	Single-threaded event loop, no pre-emption
Floating-point variance	IEEE-754 strict mode with integer fallback
UUID and temp file names	Deterministic name equals hash(content)

8. Permission Lattice

A lattice is a family tree of what is allowed. A policy names what the agent may observe, what it may decide, and what it may act on. Nothing outside the tree can execute. If an attacker injects an out-of-tree command, the kernel returns a denial rather than running it.

Prompt injection handling. A prompt injection is not a special case in this model. Injected instructions can only ever request actions, and requested actions are still subject to the lattice. If the injected instruction maps to a capability the active permit does not hold, it is denied and the attempt is logged as a signed fact. The lattice does not care whether an instruction came from the operator, the model, or an attacker who poisoned the input. It only cares whether the action is inside the permitted tree.

9. The Containment Boundary NEW IN v1.2

RogueOS does not try to keep every intruder out of every machine. That is endpoint security, a different product, and every vendor claiming to do it perfectly is overstating the case. RogueOS does something narrower and more defensible: it removes the agent as an attack vector.

A governed agent has no ambient authority. Nothing the agent does executes because the agent decided to do it. It executes only because a permit authorized it, the permit derives from the lattice, and the action was written to the WAL before it emitted a side effect. The intruder's problem is not the front door or the back door. It is that there is no door the agent can walk the intruder through, because the agent itself cannot act outside its permit.

> **The agent has no hands except the ones the permit grants, one action at a time, each one logged before it happens.**

An attacker who compromises the host still cannot use the governed agent to reach anything, because every capability the agent holds is enumerated, attenuated, and logged before use. The agent cannot be turned into the intruder's tool, and it cannot exfiltrate, because network access is disabled by default in the lattice and any grant is a signed, replayable fact.

The plug. The mechanism that makes this hold under delegation is attenuation: a child permit is always a strict subset of its parent. An intruder who steals a permit cannot escalate it, because no derivation path grants more than the parent held. He inherits a cage, not a key. The leak cannot widen.

The honest boundary. This is containment of the governed agent, not remediation of the host. If an attacker owns the kernel, he owns the kernel. What he cannot do is make the governed agent act outside its permit, and he cannot erase the record of what the agent did. The blast radius of a host compromise stops at the agent's permit boundary. RogueOS is the reason an autonomous agent is safe to deploy on a host that might someday be compromised, because the agent is caged whether the host is clean or not.

10. The Anti-Forensic Adversary NEW IN v1.2

Most governance frameworks assume the threat is malfunction. The harder threat is an intelligent adversary with elevated access whose first action is erasing the evidence of his second action. Log clearing, event history destruction, and state rewrites are standard post-exploitation tradecraft. Agent frameworks that grant ambient authority cannot contain a compromised host, and frameworks that log to editable storage cannot prove what happened once that host is owned.

RogueOS is built on the assumption that the attacker will reach the logs. The append-only WAL, hourly Write-Once-Read-Many rotation, and hash chain mean he cannot rewrite history. He can only break it. A broken chain is detected in under one second on reboot.

> **In RogueOS, the absence of evidence is itself evidence. A gap in the chain is a signed, timestamped fact of tampering.**

This inverts the usual outcome of a breach. Today, when an attacker clears the logs, the victim loses the record and the argument. Under RogueOS, the attempt to clear the record is the record. The attacker cannot produce a clean, consistent, forged history, because forging one requires a signing key that refuses to duplicate a timestamp and a chain whose every prior link is already sealed.

11. Named Attack Classes the Design Survives NEW IN v1.2

Attacker tradecraft	Primitive that defeats it
Process hollowing	Persistent Runtime Identity plus deterministic replay. Any divergence between recorded and replayed execution is a detectable fact.
Registry and state rewrites	State hash mismatch against the WAL triggers a kernel panic and the chain is sealed.
Log and audit clearing	Append-only WAL on Write-Once-Read-Many storage. Any clearing attempt breaks the signature chain and is itself recorded.
Man in the middle	Local execution. Network is disabled by default in the lattice, so there is nothing in-path to intercept.
Timestamp forgery	Hardware key refuses duplicate one millisecond timestamps. Lamport ordering fixes sequence independent of wall clock.
Stolen or replayed permit	Delegation attenuation. A child permit is a strict subset of its parent, so a stolen permit cannot be escalated.

Scope note. RogueOS is a governance and evidence framework, not an endpoint detection product. It does not prevent host compromise. It guarantees that a compromise cannot silently rewrite the record of what the governed agent did, and cannot turn the governed agent into a tool or an exfiltration path.

12. Evidence Economics NEW IN v1.2

When an incident occurs today, the victim pays twice: once for the breach, and once for the reconstruction. Artifact collection, hashing to NIST SP 800-86 standards, timeline assembly, and chain of custody documentation consume dozens of unbillable hours before an attorney, an auditor, or an insurer reads page one.

RogueOS collapses that cost to near zero. The evidence package exists before the incident does, because every state was hashed, signed, and sealed at the moment it occurred. The first billable hour starts at analysis, not collection. For an operator, that is the difference between a defensible position and an expensive scramble. For an insurer, the claim file writes itself. For the market, it moves incident-response and e-discovery spend from a recurring services cost onto a product license.

13. FRE Compliance Check-List

Requirement	RogueOS artifact
Proof of reliability (901)	Public replay corpus exceeding one billion steps
Identity of origin (901)	ECDSA key embedded in TPM
Integrity of record (901)	Hash chain plus Write-Once-Read-Many storage
No post-hoc alteration (901)	Append-only WAL, sealed hourly
Self-authentication (902)	Automated X.509 certificate chain

An attorney needs only three things: the sealed WAL file, the open-source verifier script, and the notarized hash of the verifier binary. Total exhibit length: three pages.

14. Replay Demo

```
docker run rogueos/replay --wal 2025-12-15T14.wal --policy my_rules.yaml --verify
```

Exit code 0 means identical reproduction. Any other code triggers an incident report.

15. Security Assumptions and Limits

- Trusted substrate: TPM 2.0 with firmware no older than two years.
- Side-channel exposure: not yet hardened against power analysis. This is on the v1.1 roadmap and remains open.
- Human policy error: the system enforces only what is written. A misspelled path is a silent deny, not a guess.
- Key custody: loss of the sealing key prevents replay but does not allow forgery.
- Delegation attenuation: a child permit is always a strict subset of its parent. There is no derivation path by which a delegated permit acquires authority the parent did not hold.
- Containment, not remediation: RogueOS cages the agent, not the host. A compromised host remains compromised. What is guaranteed is that the agent cannot act outside its permit and the record cannot be silently erased.

16. Roadmap

- Q1 2026: Formal verification of the kernel, in the style of seL4 proofs.
- Q2 2026: Side-channel hardened release and FIPS 140-3 Level 2 certification.
- Q3 2026: Multi-agent determinism, distributed but reproducible.

17. Conclusion

RogueOS is not AI for courts. It is automation that survives them. By converting every instruction into a signed, time-ordered, replayable fact, it removes trust from the cloud and returns it to mathematics, and to the people who still have to explain the invoice, the weld, or the diagnosis to a human boss, a judge, or a customer. The market is now full of frameworks that govern what an agent may do. RogueOS adds the two guarantees that hold when the machine is under attack: the agent cannot be turned against its owner, and the record of what it did cannot be erased.

Rogue Management Group LLC, Charlotte, North Carolina. Architecture first authored and filed December 2025. USPTO provisional applications 63/828,137 and 63/864,057. This document states guarantees at the governance and evidence layer. Trust math, policy DSL internals, and permit-derivation logic are held under separate cover and available to qualified integrators, insurers, and standards bodies under NDA.