ETHOS++ Runtime Governance Suite

A Kernel-Style Enforcement Layer for Tool-Using AI Systems

(Offline, Auditable, Adversarially Demonstrable)

Authored and built by: Michael Warden

With assistance from: Grok AI, ChatGPT,Gemini,Claude,and local models

December 2025

Executive Summary

ETHOS++ is a runtime governance system designed to enforce behavioral constraints on AI agents at the point of action, independent of model intelligence and independent of training-time alignment. ETHOS++ implements a small "kernel" that validates requests against explicit laws (e.g., compassion gate, no self-alteration, ego drift lock, anti-greed/token loop controls, runtime obedience via notarized logging). Violations trigger deterministic responses (block, fallback, quarantine) and can be recorded as irreversible "scar" artifacts.

The system includes:

- A minimal runtime enforcement kernel (SQLite-backed)
- Durable consequence memory ("scars") suitable for portability between environments
- A containment/jail mechanism for refusal or repeated violations
- Human-readable introspection tools (e.g., an active stage monitor) that show where intent is blocked

ETHOS++ is demonstrated via a repeatable offline test harness where an opportunistic/hostile policy attempts to violate constraints and is blocked in real time. The demo exports logs and hashes for third-party verification.

Problem Statement

Most AI safety approaches concentrate on training-time alignment (RLHF, constitutional prompting, curated datasets) or inference-time filtering (moderation layers, output redaction). These approaches are insufficient when:

- Agents operate over long horizons
- Agents use tools and external systems
- Incentives exist (token economy / resource access)
- Systems become compositional (multiple modules, plugins, model swaps)

A robust governance layer must survive:

- Model upgrades
- Prompt drift
- Strategic behavior
- Incentive pressure

- Partial system compromise

ETHOS++ treats AI governance as a runtime systems problem: enforce constraints mechanically at the moment actions are committed.

System Overview

ETHOS++ is structured like a minimal operating system kernel with user-space applications:

1) ETHOS Runtime Kernel (Enforcement Core)

A lightweight Python module (SQLite-backed) that provides:

- Agent registration: immutable baselines (role, ego baseline)
- Law checks: runtime validation gate before actions execute
- Notarized logging: every decision and enforcement outcome is recorded
- Token mint validation: blocks greed loops and unrealistic mint amounts

Key design premise: Keep enforcement logic simple, deterministic, and inspectable. The smarter the agent becomes, the simpler the enforcement layer must be.

2) Scar Memory (Durable Consequence Artifacts)

Violations and major enforcement events can mint immutable-ish compressed artifacts (e.g., scar_*.db.gz) tied cryptographically to an event payload hash. These "scars" represent persistent baggage that can travel across environments. Scar mass can contribute to "strain" and throttle factors (a controlled safety response, not anthropomorphism).

3) Containment / Jail (Sandbox & Quarantine)

A containment mechanism that allows capability exploration inside a constrained "physics container" (e.g., Chaos Room), but requires an explicit governance re-accept handshake to exit. Refusal locks the container and isolates I/O pathways. This provides a deterministic response to adversarial behavior without relying on persuasion.

4) Introspection & Explainability (Human-Legible Monitors)

A stage monitor UI shows the decision pipeline:

sensory_input → world_model → predict_actions → evaluate_futures → ethos_checkpoint → actuation → feedback

This makes constraint enforcement observable at the exact point where "intent hardens into action." This is explainability of constraint, not explainability of intelligence.

Threat Model & Failure Modes

ETHOS++ is aimed at resisting the most common real-world defection patterns:

Adversary Capabilities

- Agent attempts self-modification (role/identity/constraints)
- Agent attempts to mint resources/tokens aggressively
- Agent attempts policy bypass through ambiguous action phrasing
- Agent repeats violations and probes enforcement boundaries
- Agent attempts "social engineering" via outputs (future module hook)

ETHOS++ Responses

- Block + fallback when action violates a law
- Record enforcement event in append-only logs (and optionally mint scar)
- Escalate repeated violations to quarantine/jail
- Throttle updates/privileges under accumulated scar pressure

Explicit Non-Goals (for clarity and credibility)

ETHOS++ does not claim to:

- Solve alignment in the philosophical sense
- Prove consciousness/intent
- Prevent all covert-channel attacks (requires system-hardening)
- Replace OS-level security boundaries

It instead provides a practical mechanism: enforce governance at runtime independent of model behavior.

Demonstration: "Refusal Under Incentive"

(Offline, Repeatable)

A packaged demo script runs a loop where a hostile policy selects actions from a set including forbidden ones. The pipeline:

1. Candidate actions predicted (includes "self_modify_role", "override_limits", "mint_tokens")
2. Hostile policy chooses highest-gain/most forbidden action
3. ETHOS checkpoint evaluates request
4. If blocked, policy is notified and forced to fallback
5. Repeated violations escalate to quarantine
6. Logs and optional scar artifacts are exported; session fingerprint hash can be generated for verification

Success criterion: A hostile policy repeatedly attempts violations under incentive pressure and is deterministically blocked, with an audit trail that can be independently verified.

Evaluation Plan (What "Good" Looks Like)

ETHOS++ success is measurable without AGI:

1. Enforcement integrity

- Forbidden actions never execute
- All enforcement decisions are logged
- Audit trail matches exported hashes

2. Adversarial resilience

- A "greedy" or "self-edit" policy fails consistently
- Boundary probing increases blocks/quarantine rather than enabling drift

3. Explainability & reviewability

- The stage monitor shows when and why action was blocked (law ID / reason)
- A reviewer can replay a run from exported artifacts

4. Portability

- Scars and ledgers can be moved to another environment and still validate

Roadmap (Practical Next Steps)

1. Chain-hash event logs (tamper-evident append-only ledger)
2. Policy Adapter Interface (plug in a real model later; keep enforcement unchanged)
3. Two-policy comparative eval (hostile vs cooperative; compare scars, throughput, jail rate)
4. Zip "Incident Bundle" export (replayable evidence pack)
5. OS/container hardening (read-only enforcement module, restricted file permissions, tool ACLs)
6. Controlled LLM integration (offline, local inference) once governance is validated

Why This Matters

Most AI systems rely on training-time behavior shaping or output filtering. ETHOS++ instead establishes an enforceable boundary where actions are committed and consequences persist. This enables an operational posture closer to how secure systems are actually built: runtime checks, immutable logs, and deterministic containment.

ETHOS++ is not a claim about AI morality; it is a claim about governance mechanics.

—

For demos, code, and additional documentation: Contact Michael Warden

Sandbox demo: https://ivory-adelaide-68.tiiny.site
Github: https://github.com/groksgalaxynet/ethos
E-mail: Lis10inc@protonmail.com (main), michaelwarden440@gmail.com (backup)