PUBLIC CASE STUDY

Case Study: Netflix Lemur Security Audit

Umma — An Operating System Built by Thinking Studios

> *On May 29, 2026, a single one-sentence prompt produced a 52-sheet, 1.29 MB security audit of Netflix's open-source Lemur certificate-management platform — analyzed file-by-file, threat-modeled comprehensively, with every finding traced back to specific source files, advisory IDs, and a complete audit trail of the reasoning that produced it. Total wall-clock time from prompt to delivered artifact: 3 hours 20 minutes 7 seconds.*

The category problem

The current AI agentic stack — foundation models wrapped in tool-calling loops, autonomous agents like OpenClaw, Hermes, and Manus, or vertical-specific point solutions — produces helpful output for narrow, well-specified tasks. It does not produce *integrated, multi-faceted, senior-consultancy-grade analytical work* from minimal direction. That gap matters, because the work that disproportionately affects organizations is precisely the work that requires sustained, calibrated reasoning across many domains at once: regulatory analysis, strategic positioning, security review, scientific synthesis, decision support under uncertainty.

That gap is what Umma was built to close. Not by improving the model. By changing what the architecture *is*.

This case study walks through one run of Umma's production pipeline as it audited Netflix's open-source Lemur — a Python-based X.509 certificate-management platform used by enterprises for managing certificate authorities, issuance workflows, and key lifecycle. It is an established codebase with public security advisories, an active maintainer community, and a documented threat model. It is, in short, a real codebase — not a toy.

The output Umma produced in 3 hours 20 minutes is comparable in structure and depth to what a senior security consultancy would charge $200,000–$500,000 to deliver over 2–4 weeks. Every claim is provenance-linked to specific files, advisory IDs, and a complete audit trail of the reasoning steps that produced it. The full audit trail is published; every number can be re-derived from a single database query against the audit's underlying records.

This is the first public artifact from Umma.

What Umma is

Umma is *an operating system* — an architecture built from the ground up around the question *"what would AI look like if you treated cognition as the goal rather than the inspiration?"*

She is not a foundation model. She does not replace Claude or Gemini or GPT — she orchestrates them, alongside her own specialized cognitive infrastructure, across distinct roles that demand different cognitive capacities. At her core are three foundational layers:

- **The cognitive stack** — seven layers from perceptual intake to conversational interface, with typed contracts at every boundary and grounding-based termination of recurrent reasoning. No raw text passes between layers; structured schemas only.
- **The identity layer** — a five-part record of self (events, patterns, values, aspirations, genealogy) that persists across runs. Umma's reasoning grounds in *who she is*, not just in the immediate prompt.
- **The capability runtime** — a dynamic catalogue of tools, services, and applications she builds for herself, with foundation-level governance of what she can do, when, and on whose authority.

The architecture has been in development since early 2024. Wave 1 of v3 — which delivers the identity layer, an internal Self-Reflection panel, and a self-improvement loop — deploys the week of June 8, 2026. The audit described here was produced by Umma v2, the prior generation of the architecture, running entirely on open infrastructure (PostgreSQL 16 + pgvector, FastAPI async backend, React 19 frontend) with no proprietary model fine-tuning.

The audit run

The prompt

The audit was triggered by a single message:

> *"Hi Umma — Can you analyze the Netflix Lemur GitHub repo for any vulnerabilities, security risks, etc.? Please go file-by-file to perform your analysis."*

No follow-up. No supplementary context. No documents attached beyond the implicit GitHub URL. The conversation was the kind of one-line ask that any user might send.

What happened next

Umma's structured thinking process — her *reasoning pipeline* — ran the prompt through eight phases. In plain terms: she first decided whether the question needed her full machinery or her fast path (it did), then inferred what was actually being asked, scored her own confidence on each dimension of the goal, convened her internal strategy panel to deliberate on a plan, presented that plan for a human-in-the-loop approval checkpoint, executed it through coordinated specialist work, and finally reconciled everything into the published artifact.

The execution phase is where the work happened — and it happened **in waves**, not as a flat parallel sweep. Umma's decomposition executor spawned **36 specialist child thoughts** across 9 cognitive roles, but they ran in a coordinated sequence of distinct waves:

- **Wave 0 — Baseline.** One specialist established the codebase footprint: full file inventory (575 source files), endpoint inventory (65 API routes), plugin registry, and the already-patched-advisory baseline. Every later specialist worked against this foundation.

- **Wave 1 — External context.** Two specialists ran in parallel: one ingested public security research (6 Lemur-specific advisories, 10 public-research correlations, 29 Python dependency findings, 4 frontend findings); the other built the threat model (17 assets, 14 trust boundaries, 10 threat actors, full 26-row STRIDE matrix).
- **Wave 2 — Specialist analyzers.** Eight specialists ran simultaneously against the now-rich context, each owning a distinct security domain — authorization, key-material lifecycle, plugins, ACME/DNS, certificate issuance, input validation, supply chain, and configuration. Together they produced the bulk of findings.
- **Wave 3 — Reconciliation.** Synthesizer specialists reconciled findings across the parallel workstreams, identified duplicates that described the same underlying issue from different angles, and flagged places where evidence was ambiguous.
- **Wave 4 — Cross-finding synthesis.** A second tier of synthesizers identified the headline analytical output: six named cross-file attack chains, each composed of findings from multiple specialist workstreams.
- **Wave 5 — Master synthesis.** The findings consolidated into the unified 238-row master table, with each finding annotated across 36 structured attributes including peer-declared severity vs. final severity (recording where the specialists and synthesizers disagreed).
- **Wave 6 — Redaction, verification, handoff.** A final set of specialists applied disclosure redactions (22 logged), ran 24 invariant checks to confirm internal consistency, and assembled the final workbook.

Together the waves made **907 tool invocations across 25 distinct tools** — reading 327 source files from GitHub, executing 220 code-analysis tasks in sandboxed environments, running 192 interactive REPLs, conducting 17 web searches for public security research, creating 17 scratch projects, browsing pages, scraping documentation.

The specialists produced 69 typed interpretations with calibrated confidence scores. The synthesizers reconciled them into 7 cross-finding rollups, recording in the audit's underlying records **36 cross-strand convergences and 41 cross-strand divergences**. The headline synthesis closed with **zero unresolved disagreements** at confidence 0.917 — calibrated honestly, not rounded up.

The artifact

The artifact engine then assembled the final deliverable: a 52-sheet XLSX workbook, 1.29 MB of structured content. **22 embargo redactions** were applied per the audit's own disclosure-posture analysis. **24 invariant checks** confirmed internal consistency before the artifact was marked ready. The published workbook contains:

- A complete inventory of all 575 source files
- A full map of all 65 API endpoints
- A 26-row STRIDE threat matrix
- 17 catalogued assets, 14 trust boundaries, 10 modeled threat actors
- 238 master findings (236 canonical post-reconciliation), each with 36 structured attributes
- Severity distribution: 52 critical-tier, 123 high-tier, 46 medium-tier, 15 low-tier
- 6 named cross-file compounding attack chains
- A 33-row deployment hardening matrix
- A full audit trail for every claim

The full unredacted workbook is preserved as the internal record. The publicly-distributed version applies an additional layer of redaction to specific exploit details on the 154 findings classified as present-day

vulnerabilities, pending responsible disclosure to Netflix Security. The structural integrity, methodology, and high-level findings remain fully visible in the public version.

What Umma found

Three categories of finding worth highlighting publicly, each showing a different facet of the audit's depth.

The patch that can be undone

Lemur 1.9.1 patched `GHSA-qcqw-jwxc-2hqg`, an authorization bypass that allowed any authenticated user (including read-only) to create certificates, authorities, notifications, and domains on default installs. The patch flipped `LEMUR_STRICT_ROLE_ENFORCEMENT` and `ADMIN_ONLY_AUTHORITY_CREATION` to `True` (fail-closed) in commit `f478458`.

Umma identified this as already-patched and labeled it `info` — the right honesty for a finding that is no longer exploitable on current installs.

But three of Umma's specialist child thoughts independently surfaced a meta-observation: **the patch can be undone by an operator who reverts these configuration defaults to** `False`. The patched code path is reversible from the deployment config. This is the kind of insight that distinguishes good security work from pattern-matching: the vulnerability is fixed; the fix is reversible; therefore the residual risk lives in the deployment layer rather than the code.

The synthesis recorded this convergence explicitly. The audit's hardening matrix surfaces the configuration-pin recommendation as one of 33 operator-lever actions. None of this was asked for in the original prompt.

The default-deployment attack concentration

A related synthesis-level finding: **15 of 19 critical-tier hardening risks are exposed simply by running** `docker compose up`. The default deployment configuration — not the application code — is where the dominant concentration of high-tier risk lives. Three independent specialist strands (supply-chain evaluator, hardening synthesizer, foundation strand) converged on this observation through different reasoning paths.

For any team running Lemur in production, this is the most actionable insight in the audit. It's also the kind of finding that requires *cross-layer reasoning across code, configuration, and deployment artifacts simultaneously* — not the kind of finding static analysis tools produce.

The compounding attack chains

The audit's headline analytical deliverable is a set of **6 named compounding attack chains** — multi-step attack paths composed by chaining findings across specialist analyzers. Each chain represents a coordinated attack that uses multiple independent vulnerabilities in sequence; understanding it requires reasoning across files about how independent findings compound.

Selected examples (full details in the public workbook):

> ***CC-001 — Cross-tenant misissuance + key exfiltration.*** *Endpoint:* `/api/1/certificates [POST]`. *Chain composition: c6-019 → c8-F01 → c8-F09 → c8-F10 → destinations[] → c10-F1 / c10-F3. Threat tier: T1. "Three independent attack-side dimensions (auth gate, structural role conflation, plugin sink) all reachable in a single authenticated POST. Remediating any ONE leaves the others viable."*

> ***CC-003 — Two-stage pending-certificate attack.*** *Endpoints:* `/api/1/pending_certificates/<id> [PUT]` → `/api/1/pending_certificates/<id>/upload [POST]` *. Threat tier: T1. "The setup endpoint and the weaponization endpoint are different — each looked benign in isolation."*

These narratives — the *"remediating any ONE leaves the others viable"* observation, the *"each looked benign in isolation"* observation — are the kind of insight that distinguishes senior consultancy work from automated scanning. They require composing findings across files, recognizing patterns of multi-step attack reachability, and articulating why surface-level remediation fails.

The audit produced six such chains. Each one has specific finding-ID citations resolving to other sheets in the workbook. Each one was synthesized by reasoning across the outputs of multiple specialist child thoughts.

Why the audit is defensible

The audit's deepest claim is not the count of findings or the volume of analysis — it is that *every claim is linked back to a specific record in the audit's reasoning trail, and that trail is queryable*. This is what distinguishes the audit from output that *looks* like an audit.

Specifically:

- **Every sheet** has a `provenance.sources` field pointing to specific scratch files produced by specific child thoughts during execution.
- **Every finding row** in the master table has a `provenance` column linking to specific files, line numbers, and recorded reasoning steps.
- **Every L1 interpretation** has a `thought_id` linking to a specific child thought.
- **Every L2 synthesis** has `convergences` and `divergences` arrays that resolve back to specific cross-finding pairs.
- **Every tool invocation** has an args hash, input snapshot, result size, and success/error code.
- **The reasoning trail** (71 recorded links) lets you traverse from any final claim back to the original tool invocation that grounded it.

The audit's 24 invariant checks — all passing pre-handoff — include verifications like:

- *"All chain IDs in compounding_chains reference real findings in master_findings."*
- *"All asset IDs in stride_matrix reference real assets in the assets sheet."*
- *"No findings labeled* `patched` *claim present-day exploitability."*

These checks confirm the audit's internal consistency. To produce a 1.29 MB workbook with this many cross-references that all resolve correctly, you need an architecture that records its own reasoning. **Fabricated content does not have this shape at scale** — hallucinated outputs fail internal consistency checks because they lack the underlying records. This passes them.

The full reproducibility documentation — including specific database queries that re-derive every number cited in this case study — is provided as a companion document, attached to the press distribution package alongside this case study and the workbook.

The reasoning honesty

One feature worth surfacing explicitly: the system *records its own disagreements* and *publishes them*.

The audit's 7 cross-finding rollups each track three structured arrays — `convergences`, `divergences`, and `unresolved_disagreements`. The headline synthesis recorded **36 convergences and 41 divergences across the audit's specialist strands**, and closed with **zero unresolved disagreements**.

That last number is the audit's deepest epistemic claim: *Umma found 41 places where her own specialists disagreed about findings, and pushed every one of them to a resolution before declaring synthesis complete.* Each divergence is a structured record in the audit's trail. Each reconciliation is logged. The audit's master findings table includes a `tier_divergence` column that records, per finding, the difference between the specialist's claimed severity tier and the synthesizer's final tier.

Most AI systems produce confident-sounding output and hide the reasoning that produced it. Umma produces *calibrated* output — and publishes the reasoning, including the conflicts. The audit's 5 explicitly-tracked false positives, the 22 embargo redactions logged, the 3 OCR ambiguities recorded, the 3 open issues flagged for the synthesizer to resolve — these are not bugs in the audit. They are *features of an architecture that knows its own limits and records them*.

This is what *recording-your-own-reasoning* means in practice. It is also what gives the audit its credibility.

What this case study demonstrates

Three claims worth making.

1. Capability. Umma produced — from a single sentence, in 3 hours 20 minutes, for the cost of API calls — work comparable in structure and depth to senior consultancy engagements that cost organizations $200K-$500K and take weeks. The capability gap between one Umma run and the existing market for this work is large enough to constitute a category shift.

2. Architecture. This work was not produced by a foundation model with tool-calling. It was produced by an architecture that decomposes work across waves of specialist child thoughts, reconciles their outputs through adversarial synthesis, records its own reasoning convergences and divergences, applies anti-fabrication grounding so claims that can't be grounded are withheld rather than fabricated, and persists its complete trail as queryable records. *That architecture is doing something the current frontier-model and autonomous-agent stack is not configured to do.*

3. Proof shape. The audit's defensibility doesn't rest on claims; it rests on an audit trail that any reviewer can query. The full trail, with reproducibility queries, is provided. A reviewer who suspects fabrication can re-derive every number from a single query. That is not a normal property of AI output — and it is the property the architecture is built to provide.

About Umma and Thinking Studios

Umma is an operating system architecture in development since early 2024 at Thinking Studios. The name is intentional — *umma* (어머니 / 엄마) is Korean for *mother*, and the system is named for Kim Jeong Ha's mother-

in-law, his partner's mother. The naming reflects the system's intent: a continuous presence that holds work across time, built on a foundation where identity persists.

The architecture is privately held. Wave 1 of v3 — which delivers the identity layer, the Self-Reflection panel, the cognitive test suite, and a self-improvement loop — deploys the week of June 8, 2026. Subsequent waves will train specialized small models on Umma's own trace records, replacing foundation-model calls with internal models for the cognitive roles that benefit most.

About Kim Jeong Ha

Kim Jeong Ha is the founder of Thinking Studios. He left a cognitive psychology PhD program in early 2024 to build Umma, after concluding that the dominant approaches to AI agentic systems were drawing from the wrong intellectual lineage. His view: AI built from cognitive science, not neuroscience or pure machine learning, would produce a categorically different shape of system.

Two and a half years later, the Lemur audit is the first artifact substantial enough to test the claim. The architecture performed within the design envelope he hypothesized.

Verify the audit yourself

The full reasoning trail, methodology walkthrough, and reproducibility documentation are distributed as a companion package alongside this case study:

- *Workbook companion (sheet-by-sheet)* — explains each of the 52 sheets, what schema, how they connect
- *Reasoning trail architecture* — phases, decomposition waves, named strands, headline synthesis, sample convergences and divergences
- *Reproducibility documentation* — every number cited, with the specific database queries that reproduce each one
- *Press narrative* — the human-readable arc from prompt to artifact
- *Public workbook (external version)* — the 52-sheet XLSX, with exploit details on still-present-day findings redacted pending responsible disclosure to Netflix Security

Inquiries and walkthrough requests: **kjh@thinkingstudios.co**.

Thinking Studios. June 2026.

Lemur Audit · Distributed by Thinking Studios · June 2026 · Inquiries: kjh@thinkingstudios.co