Hanzo AI for Security: Automated, Agentic Security Auditing at Estate Scale

Zach Kelling — Hanzo Team

Version 1.0 2026

Abstract

Security debt does not live in the system you were asked to look at; it lives in the systems you did not know existed. Modern software estates are sprawls of hundreds of microservices, dozens of repositories, and dependency trees no human has ever read end to end—and the vulnerabilities hide precisely in that sprawl: the forgotten service with a wildcard CORS policy, the credential committed to a `.env` four years ago, the auth bypass list that a database write can edit, the package with a published CVE that no audit ever ran against. This paper describes how Hanzo applies an agentic AI stack to the problem: a fleet of specialized review agents that fan out across an entire estate, discover its real extent, find vulnerabilities at a rate no human team can match, scan every dependency tree, and re-audit continuously as code changes—always with a human in the loop for triage and sign-off. We ground the description in a real engagement (anonymized): a FINRA/SEC-regulated securities platform that engaged for a narrow remediation scope and was found to be running more than 200 microservices across 127+ repositories. Across eleven audit rounds the stack surfaced 143 findings (22 Critical), the estate's ∼780 dependency vulnerabilities were driven to zero actionable, and the platform was consolidated from 200+ services into three Go binaries—deleting roughly 3.5 million lines of dead code and, with it, most of the attack surface. The outcome was FINRA/SEC approval and SOC 2. We are explicit throughout about what the machine does versus what a human must still decide.

Executive Summary. You cannot secure what you have not enumerated, and the hard part of real-world security is that nobody has enumerated the estate. A team asks for help hardening a handful of services and the actual footprint turns out to be hundreds of services across scores of repositories, most of them undocumented, several of them forgotten, all of them in scope for the regulator and the attacker even if they were out of scope for the conversation. Hanzo's answer is to point an agentic AI stack at the whole thing. Specialized agents—one reasoning about authentication, one about secrets, one about injection, one about access control, one about dependencies—run in parallel across every repository, build the map nobody had, and report findings far faster than a human auditor can read code. The machine is good at breadth, recall, and tirelessness; it is not the final authority. A human engineer triages every finding, discards the false positives, decides severity and exploitability, and signs off remediation. In the engagement this paper anonymizes, that pairing took a securities platform from 93 source-code findings (22 Critical, 30 High) across 127+ repositories and ∼780 dependency vulnerabilities (18 critical, 350+ high) to zero actionable dependency vulnerabilities and zero open critical or high findings over eleven rounds—while consolidating 200+ microservices into three compiled binaries and deleting ∼3.5 million lines of dead code, a reduction in attack surface that no amount of patching can equal. The same properties that made this possible—owned models, an open and auditable stack, secrets in a KMS, post-quantum identity, and a cryptographically signed, immutable audit trail—are exactly the properties a SOC 2 audit, a FINRA/SEC examination, an ISO 27001 certification, and (for government systems) a continuous-authorization posture all require.

Contents

1 The Problem: Security Debt Hides in the Sprawl

A security review has an implicit assumption baked into it: that someone knows what is being reviewed. In a small, well-documented system that assumption holds. In the systems that actually run regulated businesses and critical infrastructure, it does not. These are estates—accreted over years by rotating teams, spread across many repositories, deployed as dozens or hundreds of microservices, wired together by webhooks and shared databases, and standing on dependency trees that pull in thousands of transitive packages. No single person holds the map. The org chart that built it has turned over. The documentation, if it exists, describes the system someone intended to build, not the one that is running.

This is where vulnerabilities live. Not usually in the service everyone is watching—in the one nobody remembers:

- **The forgotten service.** A proof-of-concept that quietly went to production, an internal tool exposed to the public internet, a duplicate of a service that was "replaced" but never decommissioned. It still has credentials, still talks to the database, and is still in scope for an attacker.
- **The credential in history.** An AWS key, a payment-processor secret, an OAuth client secret, a personal access token—committed to a `.env` or a build layer years ago, never rotated, still valid, sitting in a git history that an attacker can clone.
- **The structural auth flaw.** A bypass-route list loaded from a mutable database table, so that any database write can create an authentication bypass; a JWT verified through an asynchronous callback that can race the next middleware; a tenant identity derived from a client-controllable header.
- **The dependency CVE.** A published, well-known vulnerability in a package like a common HTTP client, a JWT library, an ORM, or a web framework —trivially fixed by a version bump, but invisible because no one ever ran a dependency audit across the estate in years of operation.
- **The cipher that is not encryption.** Sensitive regulated data "protected" by a reversible substitution scheme that a developer mistook for encryption, propagated across many services, and that survived years of operation because no review ever questioned it.

None of these are exotic. Every one of them is a finding a competent auditor would flag on sight—*if* the auditor's eyes ever reached that file. The binding constraint is not auditor skill; it is auditor bandwidth against estate size. A human team can deeply review a handful of services in the time a quarter allows. An estate has hundreds. The math does not close, so coverage is sampled, sampling misses the forgotten service, and the breach—or the regulator—finds what the sample did not. The problem is fundamentally one of *scale*: breadth of coverage, recall across an enormous surface, and the stamina to re-check everything every time the code moves. Those are exactly the properties a human team lacks and an agentic system has.

2 Agentic Security Auditing

Hanzo treats an audit not as a single review but as a swarm of them. The unit of work is one specialized agent assigned to one repository (or one service) with one analytical lens. Many such

agents run concurrently, so the wall-clock time to sweep an estate scales with how many workers you are willing to run, not with how many repositories the estate contains. The model's job is breadth and recall; the human's job is judgment. The two are deliberately kept in separate lanes.

2.1 Specialized agents, run in parallel

A single model prompted to "find security bugs" is a generalist and behaves like one—shallow, distractible, prone to missing a whole class of issue while chasing another. The stack instead runs a set of agents each holding one threat model in working memory, so each pass is deep in one dimension rather than shallow in all of them:

- **Authentication & session.** Token issuance and verification, expiry and rotation, algorithm pinning, bypass routes, rate limiting on auth endpoints, OTP handling, fail-open versus fail-closed middleware.
- **Secrets & credentials.** Plaintext keys in environment files, source, build layers, and manifests; hardcoded tokens and passphrases; secrets reachable in client bundles; secrets that should live in a KMS.
- **Injection & output handling.** SQL/NoSQL built by string concatenation, command injection, and cross-site scripting through unsanitized rendering of user-controlled data.
- **Access control & tenancy.** Missing ownership checks (IDOR), object lookups that resolve across tenant boundaries, authorization enforced in the UI but not the API, header-driven tenant switching.
- **Transport & configuration.** CORS allowlisting (especially wildcard-with-credentials), security headers (CSP, `X-Content-Type-Options`, HSTS), TLS posture, and webhook signature/replay protection.
- **Dependencies & supply chain.** The full transitive tree of every package manifest, matched against published advisories, with a remediation path (override, upgrade, replace, or remove) for each.

Because the agents share a finding schema—identifier, severity, location, category, evidence, suggested fix—their outputs merge into a single estate-wide ledger that deduplicates the same flaw recurring across services (the wildcard CORS that appears in a dozen places, the same vulnerable package in every tree) and makes the systemic pattern legible rather than buried in a hundred separate reports.

2.2 Automated discovery: building the map first

The first thing the stack produces is not a finding but an *inventory*. Before reviewing anything, agents crawl the version-control organizations, enumerate every repository, classify each by language and stack, read whatever deployment configuration exists, and reconstruct the actual service graph—which services are live, which are duplicates, which are abandoned, what talks to what. This is the step a human team usually skips because it is tedious, and it is exactly the step that surfaces the forgotten service. In the engagement this paper describes, this discovery pass is what turned a "narrow remediation" into a true accounting of the estate: it found a footprint of more than 200 microservices across 127+ repositories where the engagement had been scoped to a handful. You cannot audit, certify, or—for a government system—accredit a system whose extent you have not established; automated discovery establishes it.

2.3 Dependency scanning across the whole tree

Dependency risk is the highest-leverage target for automation because it is pure breadth: every repository, every manifest, every transitive edge, matched against a moving database of advisories. A human will not do this across hundreds of repositories; a machine does it exhaustively and repeatably. The stack resolves each tree, identifies every package with a known advisory, and—critically—does not stop at a list. For each vulnerable package it determines the minimal action: a transitive *override* to a patched version, a direct *upgrade*, a *replacement* of a deprecated or unmaintained package with a maintained equivalent, or *removal* of a dependency that was dead weight pulling in a vulnerable transitive package. The residual—advisories with no upstream fix—is documented with an explicit reachability argument (why the affected code path is not exploitable in context) rather than silently ignored. "Zero actionable" means every advisory has been either fixed or formally adjudicated, which is the state an auditor and a regulator will accept.

2.4 Continuous re-audit

An audit is a photograph; security is a film. A finding fixed today regresses tomorrow when someone reintroduces a wildcard CORS in a service the original sweep did not cover, or a dependency bump pulls a fresh CVE, or a refactor reopens a closed injection path. Because each agent run is automated and cheap relative to human review, the same sweep runs continuously—in CI on every change and on a schedule across the whole estate—so regressions are caught as events rather than discovered in the next annual assessment. Every closed finding becomes a standing regression check: the specific bypass that was fixed is asserted to stay fixed, the specific CORS origin is asserted to stay allowlisted, the specific package is asserted to stay above its patched version. This converts "we audited it once" into "it is audited on every commit," which is the substantive content of *continuous authorization*.

2.5 Honest about the human in the loop

The machine is not the auditor of record, and claiming otherwise would be both false and dangerous. Automated static analysis over-reports: it flags string-built queries that are in fact parameterized, "secrets" that are test fixtures, and IDORs guarded by a check it could not see. It under-reports too: business-logic flaws, broken authorization semantics that are individually valid but collectively wrong, and abuse cases that require understanding intent are not reliably found by pattern-matching. So the division of labor is explicit. The agents deliver breadth: they read everything, miss little of the mechanical classes, and never get bored on repository two hundred. A human security engineer delivers judgment: triaging each finding, discarding false positives, confirming exploitability, setting severity, threat-modeling the critical paths by hand, and signing off remediation. The honest claim is not "AI replaces the auditor." It is "AI gives one auditor the reach of fifty, and the auditor still decides." The final sign-off in the case study was human, and the staged penetration re-tests that production money movement demanded were human; the machine made that human able to cover an estate that would otherwise have been impossible to cover.

3 Case Study: From a Narrow Scope to a 200-Service Estate

The following is drawn from a real engagement with a FINRA/SEC-regulated digital securities platform. All identifying details—company, product, people, services, repositories, domains—are removed; the numbers and the shape of the work are as they happened.

3.1 The scope that wasn't

The engagement began as a targeted vulnerability assessment of a small set of services. The automated discovery pass (§2.2) immediately changed the picture: an organization-wide crawl enumerated an estate of **more than 200 microservices across 127+ repositories**, spanning an API gateway, trading and matching engines, a custody/payments tier, KYC/identity services, a sprawl of frontends, and a long tail of proof-of-concept and duplicate repositories that were still live. The "narrow scope" was a keyhole view of an estate an order of magnitude larger—and, as is typical, the keyhole did not happen to be pointed at the worst of it.

3.2 What the swarm found

A parallel static-analysis pass—one worker per repository, with the specialized lenses of §2.1—produced an internal source-code audit of **93 findings: 22 Critical, 30 High, 32 Medium, 9 Low**, concentrated in the gateway, trading, and custody services. The findings were not subtle once seen; they were invisible only because no one had looked across the whole surface. By category:

- **Authentication.** A bypass-route list loaded from a mutable database table (any DB write could mint an auth bypass); JWT verification via an async callback that could race the next middleware; tenant identity taken from a client-controllable header, allowing tenant switching; tokens without expiry; no rate limiting on OTP/credential endpoints; fail-open middleware that admitted requests when the auth service was unreachable.
- **Secrets.** Plaintext cloud keys, payment-processor secrets, and database passwords committed to environment files across the custody and payment tiers; an OAuth client secret shipped inside a frontend JavaScript bundle; a hardcoded webhook secret in source; build-time tokens leaked into container layers.
- **Injection.** SQL built by string concatenation in a payments-rail query path; cross-site scripting via unsanitized rendering of user-controlled data in multiple frontends.
- **Access control.** Missing ownership checks on order modification (IDOR); object lookups that crossed tenant boundaries; withdrawal flows lacking an approval gate; missing KYC and velocity gates on crypto withdrawals.
- **Transport & webhooks.** Wildcard CORS *with credentials* enabled across several services; missing or unverified webhook signatures on inbound custody and payment events; missing security headers.
- **Financial correctness.** Floating-point arithmetic on monetary values—an integrity and compliance defect as much as a precision one.

A later forensic pass added the kind of finding that only a full-history, full-estate sweep surfaces—a class of cryptographic misuse, in which data was "protected" by a scheme that was not in fact secure, that had propagated across multiple services and survived years of review because no prior audit ever reached every service. The specifics are immaterial here; the lesson is that estate-scale, full-history coverage finds what sampling structurally cannot.

3.3 The dependency debt

Running the dependency scanner (§2.3) across the estate exposed the technical debt of a team that had shipped volume for years without ever running an audit. The full ecosystem carried **∼780 known dependency vulnerabilities**, including **18 critical and 350+ high-severity**. Narrowing to the eight core services, **141** vulnerabilities sat in their trees—published CVEs in exactly the load-bearing packages you would least want them in: the HTTP client, the JWT library, the ORM, the web framework. Every one was fixable; none had been fixed, because no one had looked.

3.4 Remediation to zero actionable

Remediation followed the discipline of §2.3. The 141 vulnerabilities in the core services were driven to **0 actionable**, and the full ∼780-vulnerability ecosystem was driven to **0 actionable** as well—every advisory either fixed or formally adjudicated as unreachable. The mechanical work was exactly the catalog automation is built for: transitive overrides to patched versions, deprecated-package migrations (a v2 cloud SDK to its modular v3, an unmaintained HTML minifier to a maintained fork, an old crypto library swap that dropped a vulnerable transitive dependency), and removal of dead dependencies that existed only to pull in vulnerable transitive packages. The handful of residual advisories with no upstream fix were documented with reachability arguments, not buried.

On the source-code side, the findings were closed across **eleven audit rounds**. Round one surfaced the initial 93; subsequent rounds drove remediation, caught regressions (including a reintroduced wildcard CORS found during a verification pass), hardened the multi-tenant authentication path, purged the legacy service framework, and re-verified each closed finding. Across all eleven rounds the series recorded **143 findings, 115 fixed**, with the remainder accepted or mitigated as medium/low items carrying no exploitable path. The remediation classes were the standard hardening moves, applied estate-wide rather than service-by-service:

- Secrets moved out of plaintext environment files and into a KMS, with machine-identity access rather than shared static credentials.
- Wildcard CORS replaced by strict origin allowlists everywhere it appeared.
- Security headers added; webhook ingress hardened with HMAC signatures plus timestamp and nonce replay protection.
- JWTs verified synchronously with a pinned algorithm, expiry, and rotation; for the next-generation build, JWKS-verified tokens against a real identity provider.
- Access control corrected with explicit ownership checks and tenant-scoped queries at the middleware and datastore layers.

3.5 The real fix: deleting the attack surface

Patching findings hardens the code you keep. The larger security win was deciding how much code to keep at all. In parallel with remediation, the estate was consolidated: **200+ microservices collapsed into three Go binaries**, and the active codebase fell from **3.07 million lines to 1.06 million** —roughly **3.5 million lines of dead code deleted**, git-verified, with the superseded repositories archived. This is the most underrated security action available, because the cleanest defense against a vulnerability in a service is for that service not to exist. Each deleted service is one fewer credential to leak, one fewer dependency tree to patch, one fewer CORS policy to misconfigure,

one fewer forgotten endpoint on the public internet. Less code is less attack surface, and a 70% reduction in active code is a 70% reduction in the places a flaw can hide—a result no patch campaign can match because it removes the substrate the flaws live in.

3.6 Outcome

The pairing of automated breadth and human judgment took the platform from 93 source findings and ∼780 dependency vulnerabilities to zero open critical or high findings and zero actionable dependency vulnerabilities, on an estate it had not even fully enumerated at the start, while shrinking that estate by an order of magnitude. The platform achieved **FINRA/SEC approval and SOC 2**—the regulatory and audit outcomes that are the real test of whether security work was not just performed but *evidenced*.

4 Security by Construction in the OSS Stack

Auditing finds the holes; architecture decides how many there are to find. The same Hanzo open-source stack that runs the agents is built so that whole classes of finding cannot occur, and so that the evidence an auditor needs is produced as a byproduct of normal operation rather than reconstructed after the fact. The case study's remediation was, in effect, a migration toward these properties.

- **KMS for secrets.** Secrets live in a key-management service and are delivered to workloads by machine identity, not committed to environment files, source, or manifests. The single most common critical finding in the case study —plaintext credentials—is structurally prevented when there is no plaintext credential to commit. Access is mnemonic-derived and consensus-authorized, with no open fallback mode.

- **IAM for identity.** Authentication and authorization run through one identity provider over standard OIDC, with tokens verified against published JWKS. Bespoke per-service auth—the source of the bypass-list, the header-driven tenancy, and the race-prone verification in the case study—is replaced by a single audited path, so there is one place to get right instead of hundreds to get wrong.

- **Post-quantum identity and transport.** Service identities and inter-service transport use post-quantum primitives (ML-KEM for key establishment, ML-DSA for signatures, NIST FIPS 203/204). Every service-to-service call is mutually authenticated and forward-secret against a future quantum adversary, closing the "harvest now, decrypt later" exposure on traffic and on the audit record itself.

- **Cryptographically signed actions.** Consequential actions—a secret access, an administrative change, an agent's action—ride a signed envelope binding the actor's identity, the operation, a timestamp, and a nonce, signed with a post-quantum key. Authority is verifiable and non-repudiable: not "a log says this happened" but "this actor provably authorized this, and the signature proves it."

- **Immutable on-chain audit trail.** Security-relevant events are recorded to an append-only, tamper-evident ledger. The audit trail cannot be edited or deleted after the fact—the precise property that books-and-records and custody regulations demand, and the precise property an after-the-fact log store cannot guarantee.

The throughline is that confidentiality, identity, authority, and auditability are *architectural defaults*, not features bolted on per service. An estate built this way starts where the case-study estate finished, and the agentic auditor then has dramatically less to find—and what it does find, it finds against a system that is already producing the evidence to prove the fix.

5 Why It Matters for Audit, Certification, and Accreditation

Regulated enterprises, critical-infrastructure operators, and government systems—defense among them—face the estate-scale security problem in its most acute form: larger sprawls, longer lifetimes, higher consequences, and a formal audit or accreditation regime that demands evidence, not assertion. The capability described here maps directly onto those requirements, from a commercial SOC 2 or ISO 27001 audit through to a government Authorization to Operate.

- **Boundary and control evidence.** Every audit and accreditation regime—SOC 2, a FINRA/SEC examination, and (for government systems) an Authorization to Operate—requires an enumerated system boundary, an assessed control set, and a documented residual-risk posture. Automated discovery produces the boundary (including the forgotten services that would otherwise sit outside it), the agent swarm produces the assessment at a coverage no sampling approach reaches, and the signed, immutable audit trail produces the evidence—turning the evidence package from a periodic manual artifact into a continuously regenerated one.

- **Supply-chain assurance.** The dependency scanner enumerates the complete transitive software bill of materials, matches it against advisories, and drives it to zero actionable with a documented adjudication for every residual. This is the concrete substance behind software-supply-chain mandates (SBOM, known-vulnerability remediation), which now run from commercial procurement through to federal acquisition: not a one-time scan but a standing, re-run guarantee.

- **Continuous authorization.** The shift from periodic to continuous re-audit is exactly the shift modern frameworks are pushing toward—continuous SOC 2 monitoring on the commercial side and continuous ATO (cATO, ongoing authorization) on the government side. Every closed finding becomes a standing regression check; every commit is re-assessed; drift is an event, not a year-end surprise. The posture reflects the system as it is, not as it was at the last assessment.

- **Sovereign and auditable.** The entire stack—the review agents, the models behind them, the KMS, the IAM, the post-quantum cryptography, the ledger—is owned and open. It runs offline, at the edge, in an air-gapped or classified enclave, with no dependency on a foreign or black-box service and no telemetry leaving the boundary. The customer can inspect the auditor itself, which is the only honest basis for trusting an auditor's output.

The deeper point the case study makes for any high-assurance buyer is about *consolidation as defense*. A 200-service estate reduced to three auditable binaries is not only cheaper to run; it is a fundamentally smaller thing to defend, certify, and reason about. In a domain where every additional service is additional attack surface, additional audit or accreditation scope, and additional supply-chain exposure, the ability to find what you have, prove it secure, and then make it radically smaller is itself a security capability.

6 American Open Source for a Sovereign Security Posture

The stack described here is built and owned end-to-end by Hanzo AI, an American-owned applied-AI lab and venture studio (Techstars '17, founded 2014) responsible for more than 100 venture-funded startups and several unicorns, founded by an expert in both AI/ML and cryptography. Two facts make this relevant to any security-conscious buyer—a regulated enterprise, a critical-infrastructure operator, or a government and national-security program. First, sovereignty of the security tooling itself: an audit is only as trustworthy as the party that controls the auditor, and this stack—agents, models, key management, and cryptography—is American-owned, auditable, and free of dependence on foreign or black-box components, implemented to the same standards (FIPS 203/204/205, CNSA 2.0) the U.S. government mandates and inspectable by the party that has to rely on it. Second, sovereignty of the AI doing the work: U.S. frontier AI has narrowed to a closed commercial duopoly, while the only broadly open frontier weights available today are largely foreign, leaving regulated and defense buyers to either rent from the duopoly or take a dependency on foreign models. Hanzo is the American open-source frontier alternative—it owns both the Zen model family and this cryptographic stack, so the agents that read your code, the keys that protect your secrets, and the ledger that records your evidence can be deployed offline, at the edge, and under full audit, with nothing leaving your boundary. That this is engineering rather than research is evidenced in production by the case study itself: an American-owned agentic stack took a FINRA/SEC-approved, SOC 2 securities platform—an estate of 200+ microservices—to zero actionable dependency vulnerabilities and zero open critical findings, and consolidated it into three compiled binaries, with a small team.

7 Conclusion

Security debt hides in the part of the estate nobody is looking at, and the reason nobody is looking is that human review does not scale to hundreds of repositories and millions of lines. Hanzo's contribution is to make the looking scale: an agentic stack that first discovers the real extent of an estate, then sweeps it in parallel with specialized review agents, scans every dependency tree to zero actionable, and re-audits continuously as the code moves—always with a human engineer holding triage and sign-off. The anonymized case study is the proof: a narrow engagement that uncovered a 200+-service, 127+-repository estate; 143 findings (22 Critical) surfaced over eleven rounds; ∼780 dependency vulnerabilities driven to zero actionable; the estate consolidated into three Go binaries with ∼3.5 million lines of dead code deleted; and FINRA/SEC approval plus SOC 2 as the result. The same stack that audits is built so that whole classes of finding cannot occur in the first place—secrets in a KMS, identity in one IAM, post-quantum transport, signed actions, an immutable on-chain trail—and all of it is American-owned, open, and sovereign. For any audit, certification, or accreditation regime—SOC 2 and FINRA/SEC, and for government systems the ATO—that is the combination being asked for: enumerate what you have, prove it secure with evidence rather than assertion, keep proving it on every change, and make it smaller. Find everything; fix what matters; delete the rest; and let the auditor be a machine you are allowed to inspect.