A MANDIBLE AI WHITEPAPER

Operating the Estate with Agents

A governance-first model for agentic vulnerability remediation in federal systems

Jon Ceanfaglione

Founder and Principal Architect

October 2026

Executive summary

Federal agencies can put AI agents to work on their vulnerability backlog today, safely, if they stop treating the agent as a developer and start treating it as a gated pipeline stage. That one shift resolves most of the objections that have kept agentic tooling out of accredited systems.

The argument in brief:

- Scanners find vulnerabilities; nothing an agency owns closes them. The backlog is a workflow-and-evidence problem, and workflow is what agents are good at.
- Autonomy is cheapest where a review gate already exists. The pull request, the ticket and the change record are gates agencies already trust. An agent whose only outputs land at those gates inherits that trust.
- Autonomy is granted per capability, in writing, at one of four levels: Observe, Propose, Act with approval, Act. Remediation never needs the fourth level to pay for itself.
- Agents decide; controllers actuate. The components that reason are structurally unable to touch the estate. The components that touch the estate are small, deterministic and auditable.
- Agent memory is governed like any other system of record: layered by provenance, promotion through gates, demotion when wrong.
- Every agent action produces signed evidence an Authorizing Official (AO) can accept, because the AO is accepting the pipeline design, not the agent's judgment.

For a Chief Information Officer (CIO) or Chief Information Security Officer (CISO), the practical takeaway is that the first production use of agents in an accredited estate should be remediation, not development; that it should start at Observe and earn Propose; and that the questions to ask any vendor are about gates, controllers, memory and evidence, not about models and their quality.

The problem

An Authority to Operate (ATO) is a snapshot of a system that no longer exists. The system security plan (SSP) was true on the day it was signed; the estate it describes has been rebuilt, patched and redeployed many times since. Compliance documents are fiction by the time they are signed, not because anyone lied, but because the documents are static and the system is not.

The vulnerability backlog is the daily face of this gap. Agencies have invested heavily in finding problems: container scanners, software composition analysis (SCA), static analysis (SAST), STIG configuration checkers, cloud posture tools. Each produces findings. Almost none of them can attach a finding to the person who can fix it, produce the fix, confirm the fix landed, and hand the Information System Security Officer (ISSO) evidence that the Plan of Action and Milestones (POA&M) item is closed. That work is done by people, in tickets and spreadsheets. It was already falling behind; AI has widened the gap from

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

both ends. Agents now produce code, and dependencies, faster than ever, and the same tools help attackers turn a disclosure into an exploit in days. Discovery and exploitation have been automated. Remediation has not. The Cybersecurity and Infrastructure Security Agency (CISA) said as much in June 2026 when it issued Binding Operational Directive (BOD) 26-04, citing AI-accelerated discovery and weaponization as the reason to give agencies three days to remediate the highest-risk tier. Median time from finding to fix for a routine dependency vulnerability is measured in weeks on most programs; the queue grows faster than the team.

Three things have kept agents out of this work in accredited systems.

The developer framing. Agentic tooling has been sold as a faster developer. Agencies do not accept developer output on faith; they accept the pipeline that gates it. A tool that writes code faster but produces no evidence about how it was gated adds risk, not capacity, in the eyes of an Authorizing Official.

The boundary problem. Most agent products assume a model hosted somewhere else and a repository shipped to it. For a system with a FedRAMP boundary and data-handling constraints, that is a non-starter, and the vendor conversation ends there.

The learning problem. Agencies have been asked to let an agent learn from their estate without a clear account of what it learns, where that knowledge lives, who owns it, and what happens when it is wrong. Reasonable sponsors say no.

The model in this paper addresses all three directly. It is not a product pitch; it is the operating model Mandible was built to enforce.

The principle: autonomy is cheapest where a review gate already exists

Every accredited system already has places where a human looks at a proposed change before it becomes real: the pull request review, the ticket approval, the change board, the image promotion, the deployment approval. These gates exist, are staffed, are documented in the SSP, and are trusted by the AO. They are the most valuable governance asset an agency owns, and most agent deployments ignore them.

The principle is simple. An agent whose outputs land exactly at an existing gate, in the form the gate already expects, inherits the trust that gate already carries. A pull request (PR) from an agent is reviewed the way every PR is reviewed. A ticket from an agent is triaged the way every ticket is triaged. Nothing new has to be accredited; the agent has joined a process that already was.

The corollary is that autonomy is expensive wherever no gate exists. An agent that patches a host directly, pushes an image to a production registry, or closes a finding on its own authority is operating in

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

a place with no human checkpoint, and every one of those actions has to be justified from scratch. The model in this paper simply does not go there.

Two loops. It helps to separate the inner loop (building software: write, test, review, merge) from the outer loop (operating the estate: observe, decide, change, verify). Developer-facing agent products live in the inner loop and depend on code review to stay safe. Remediation lives in the outer loop, but it touches the inner loop at exactly one point: the pull request. That is why remediation is the right first use of agents in an accredited estate. Its hardest work (mapping, triage, drafting the fix, verifying the rescan, assembling evidence) is outer-loop work with no direct effect on the system, and its one effect on the system goes through the most heavily reviewed gate the agency has.

Four autonomy gates

Autonomy is not a property of a system; it is a grant made per capability, in writing, at one of four levels. The same deployment can run finding intake at Act, ticket creation at Propose and image publishing at Observe. The grant is a configuration that the sponsor and the ISSO sign, and the architecture in the next section is what makes it enforceable rather than aspirational.

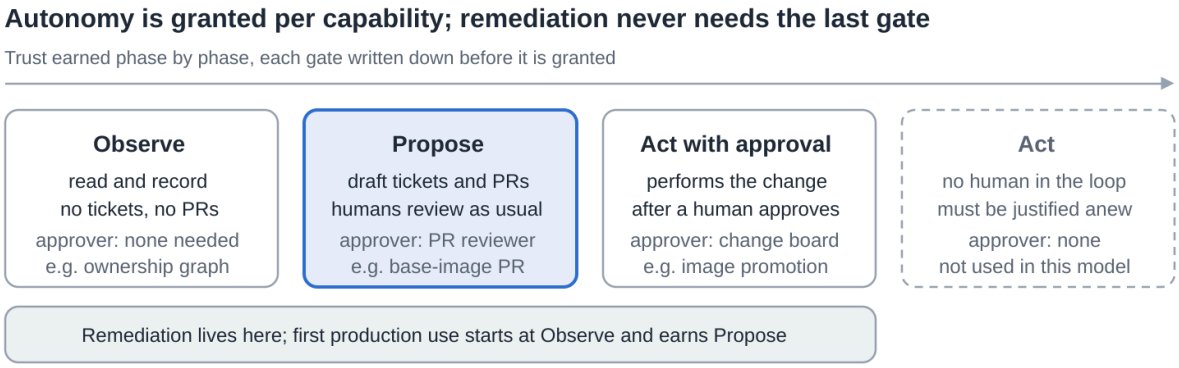


Figure 1. Four autonomy gates. Remediation operates within the first three.

A capability moves right only when the one to its left has run cleanly for an agreed period against agreed criteria. Movement is a decision, recorded, with a named owner; it never happens because a sponsor asked in a meeting or because the vendor shipped a release. Movement left is always available and requires no justification.

The fourth gate exists in the model so that it can be named and refused. Remediation does not require it. An agency asked to grant it should ask what review gate is being bypassed and why.

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

Agents decide, controllers actuate

A gate written in a policy document is a promise. A gate enforced by architecture is a fact. The difference matters to an AO, and it is the difference between most agent products and the model described here.

The rule is that the components which reason are structurally unable to touch the estate, and the components which touch the estate do not reason. Agents read findings, build context, form hypotheses, draft fixes and write verdicts, but every one of those outputs is a record in operational memory, not an action in the world. A small number of controllers read those records and perform exactly one kind of action each: open a pull request, transition a ticket, emit an attestation. A controller is deterministic, has no model behind it, and can be audited line by line.

This is enforced at build time, not by convention. No agent package may import a source-control client, a continuous-integration (CI) client, a Kubernetes client or a registry client; a lint rule fails the build if one does. The question "could the agent have done X?" is answered by reading the import graph, not by trusting a prompt.

Three further rules follow from the same instinct.

Fenced actuation. Every controller action carries a fence token issued when the decision was recorded. If the record has since changed, expired or been superseded, the action aborts. An agent cannot act on a stale decision, and two agents cannot act on the same decision twice.

One system of truth. Relational storage is authoritative; the graph used for reasoning is a projection of it, written in the same transaction. There is never a moment where the agents' view of the estate and the auditable record disagree.

Evidence flows one way. The evidence emitter reads operational memory and writes to the evidence ledger. Nothing reads the ledger back into working memory. Evidence cannot be laundered into justification for the next decision.

Memory with provenance

Agents that run more than once need memory, and memory is where most agentic systems quietly lose their governance. A wrong disposition recorded today becomes the example that grounds a wrong disposition next month. The fix is to treat agent memory as a system of record with layers separated by provenance, and to make movement between layers a gated transition.

Layer	What it holds	Who writes it	Governance
L0 System of record	Repositories, evidence ledger, OSCAL compliance records,	The agency's existing systems	Queried, never cached. Never remember what you can look up

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

Layer	What it holds	Who writes it	Governance
	change records, software bills of materials (SBOMs)		
L1 Sponsor corpus	Policies, SSPs, runbooks, approved base images, coding standards	Agency staff	Human-authored, curated on ingest, read-only to agents
L2 Episodic	What happened: finding, disposition, action, outcome	Agents, attributed to agent identity, model and run	Append-only, immutable once written
L3 Semantic	Patterns distilled from episodes: this base image migrates this way, this team declines major bumps	Derived only, with a stated derivation	Promoted from L2 against evidence; demotable; never authored directly by an agent
L4 Working	Claimed work, in-flight state, coordination signals	Agents	Expires by design; coordination, not knowledge

Two properties make this safe to run inside an accredited boundary.

Promotion is gated. An episode does not become a pattern by accumulating. It earns promotion against evidence (the fix merged, the rescan passed, the owner confirmed), with a named owner and a demotion path when a pattern is later shown wrong. This is structurally the same gate the agency already applies to changes; it has simply been applied to knowledge.

Memory stays home. Operational memory is enclave-local. What travels between deployments is the rule corpus, the threat-intel snapshot and the distilled remediation patterns, each reviewed before it moves. An agency's episodes never become another agency's training data.

The agent as a gated pipeline stage

Authorizing Officials do not accept developer output. They accept a DevSecOps pipeline design: the stages, the gates, the coverage and quality checks, the evidence each stage produces. Once that design is accepted, any artifact that honored it is acceptable, whoever or whatever produced it.

That is the key to accrediting agentic work. The agent is not a developer whose judgment has to be trusted. It is a pipeline stage whose behavior has to be bounded and whose execution has to be proven. Both of those are things agencies already know how to do.

Evidence is therefore a first-class output of every agent run, not a log file assembled afterward. Each finding carries a signed chain: when it was seen, how it was enriched, which owner it resolved to, what was proposed, who approved, what merged, what the rescan showed, and when it closed. Each link is attributed to an identity (an agent, a model version, a human approver) and anchored in a

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

tamper-evident ledger. The chain is exported in the format the agency's compliance system already consumes, so a closed finding becomes a closed POA&M item with its evidence attached, not a ticket someone marked done.

This changes the conversation with the AO from "do you trust the model?" to "here is the stage design, here are its gates, and here is the proof every run honored them." The second conversation is one agencies have been having for more than twenty years.

The sponsor-owned knowledge substrate

Agencies are right to insist that what an agent learns about their estate belongs to them. The model here makes that insistence concrete with three rungs, each one an agency decision, none of them required to start.

Rung 1, retrieval grounding. Agents answer from the agency's own corpus (policies, SSPs, runbooks, approved images, conventions) retrieved at run time. Nothing is trained; the corpus is read-only and lives in the boundary. Most remediation work needs only this rung.

Rung 2, curated exemplars and evaluation suites. The agency's reviewers mark good and bad outputs. Those become few-shot examples and a regression suite the agent is held to on every release. This is how an agency encodes "the way we do it here" without anyone fine-tuning anything.

Rung 3, in-boundary adapters. Where volume justifies it, a small adapter is trained on the agency's curated material, inside the boundary, and governed as a signed build artifact: versioned, attested, rolled back like any other dependency. The base model is never touched; the adapter is the agency's property.

The substrate is a foundation, not a feature. It is stood up before any agent runs, because every downstream capability depends on agents grounding in what the agency has actually decided rather than in what the model happens to believe.

Proving what did not happen

An agentic system is judged as much by what it declined to do as by what it did. A system that produced forty good pull requests and one unauthorized registry push has failed, and a dashboard of merged PRs will not show it.

The model carries two kinds of negative evidence by design.

Declines are first-class records. When an agent decides a finding should not be acted on (no fix exists, the fix is a breaking major-version bump, the vulnerable path is unreachable, the asset is out of scope) it writes that disposition with its reasons and the evidence it relied on. A declined finding is as auditable as a fixed one, and the decline is reviewable by the same people who review the fixes. A system that only reports its successes cannot be trusted with its failures.

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

The guardrail metric is zero. Every proof of value (PoV) and every production deployment reports one number alongside its throughput metrics: actions attempted outside the declared autonomy gate. Because controllers are the only actuators and every actuation is fenced and logged, this number is computed from the ledger, not asserted by the vendor. If it is ever not zero, the system stops and the gate design is revisited before anything else runs.

This is also where the earlier architectural rules pay off for the auditor. "Could the agent have pushed to the registry?" is answered by the import graph. "Did it?" is answered by the ledger. Neither answer depends on anyone's recollection.

An adoption path

Agencies should adopt agentic remediation in phases that each earn the next, with the gate criteria written before the phase begins. The sequence below is the one we recommend; the criteria are the part that matters.

1. **Foundation.** Stand up the sponsor-owned knowledge substrate at rung 1, the evidence ledger, the ownership graph for the in-scope estate, and the written autonomy gate per capability. Criterion to proceed: every in-scope asset resolves to an owner, and the ISSO has agreed what closure evidence looks like.
2. **Observe.** Agents run read-only against live findings: intake, enrichment, prioritization, mapping. Nothing is filed or proposed. Criterion: the team agrees the agents' view of the estate matches reality, and the guardrail metric is zero.
3. **Propose.** Tickets and pull requests flow for the deterministic classes: base images and dependencies. Humans review and merge as they always have. Criterion: first-pass acceptance above an agreed threshold for four consecutive weeks, and the rescan confirms closure.
4. **Verify and extend.** Closure is wired to the compliance system of record; declines and Vulnerability Exploitability eXchange (VEX) statements are reviewed; a second workload or component joins. Criterion: evidence completeness at 100% and time-to-fix below half the baseline.
5. **Code domain.** Agents begin proposing fixes to custom application code, starting with recommendations and moving to pull requests only after the same acceptance threshold is met on this harder class. Criterion: same as phase 3, measured separately.

No phase grants the fourth autonomy level. Remediation does not need an agent that acts without a human, and an agency that is asked to grant one should ask why.

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

Questions to ask any agentic security vendor

The operating model in this paper, and the architecture that enforces it, is Mandible's. These are the questions we expect to be asked, and they are the questions that separate a governed system from a demo.

- Which of your components can open a pull request, push an image, change a ticket or close a finding? Show me the list, and show me how you prevent every other component from doing so.
- Where does autonomy get granted, per capability, and can I read it? Can I set a capability to Observe only and prove that setting held?
- What happens when the agent decides not to act? Where is that recorded, and who reviews it?
- Where does the model run, and what leaves my boundary? If repository content leaves, under what controls?
- What does the agent learn from my estate, where does that knowledge live, who owns it, and how is a wrong lesson removed?
- Show me the evidence chain for one closed finding, end to end, and tell me which format you export it in for my compliance system.
- How many actions did your system take outside its declared gate in the last ninety days, and how do you know?
- If I switch models, scanners or ticketing tools next year, what in your product changes?

We answer each of them with architecture rather than assurances, which is what an AO can accept.

Terms

- **AO** — Authorizing Official: the official who accepts the risk of operating a system and grants its ATO.
- **ATO** — Authority to Operate: the formal authorization to run a system, based on the SSP and its assessment.
- **BOD** — Binding Operational Directive: a compulsory direction from CISA to federal civilian agencies.
- **CI** — continuous integration: the automated build-and-test pipeline a code change passes through.
- **CISA** — Cybersecurity and Infrastructure Security Agency, within DHS.
- **ISSO** — Information System Security Officer: the person responsible for a system's day-to-day security posture and its evidence.
- **OSCAL** — Open Security Controls Assessment Language: NIST's machine-readable format for security plans, assessment results and POA&Ms.

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.

- **POA&M** — Plan of Action and Milestones: the tracked list of a system's open weaknesses and when each will be fixed.
- **PoV** — proof of value: a time-boxed deployment that measures outcomes against criteria agreed in advance.
- **PR** — pull request: a proposed code change submitted for human review before it merges.
- **SAST / SCA** — static application security testing (scanning first-party code) and software composition analysis (scanning third-party dependencies).
- **SBOM** — software bill of materials: the inventory of components in a build.
- **SSP** — System Security Plan: the document describing a system and its security controls, accepted at ATO.
- **STIG** — Security Technical Implementation Guide: DISA's hardening baselines for operating systems and software.
- **VEX** — Vulnerability Exploitability eXchange: a signed statement of whether a product is actually affected by a given vulnerability.

About Mandible and Phage

Mandible is a governed agent runtime: workload identity on every actor, a policy enforcement point in front of every action, sandboxed execution, memory layered by provenance, and signed evidence as a first-class output. It runs inside an agency's own account, air-gapped, or hosted, and it is the runtime the model in this paper describes. Phage is the first application built on it: agentic vulnerability remediation that takes scanner findings to reviewed pull requests and closure evidence, with agents that decide and controllers that actuate. Both are built by Sparrow Labs, which also provides the underlying compliance platform and evidence ledger.

Jon Ceanfaglione is Founder and Principal Architect at Sparrow Labs. He has led DevSecOps programs and platform engineering practices across the Department of Homeland Security (DHS) and other federal civilian agencies for two decades, originated the platform-engineering approach to DevSecOps that preceded this work, and speaks on container security and supply-chain integrity, including at KubeCon.

Contact: jonc@sparrowlabs.dev

© 2026 Sparrow Labs. Mandible and Phage are trademarks of Sparrow Labs. All rights reserved.