Xiaotime *Labs*

POSITION PAPER

The *Governed* Fleet

Why a fleet of AI agents is only as trustworthy as the control point they all share.

Position paper.

Version 1.0
May 2026

For CISOs, security teams, and security-minded founders evaluating what it takes to run autonomous AI agents in an environment that has to pass a real audit.

The question your architecture has to answer

When a security team evaluates an AI platform, the capability questions come first. What can the agents do. How fast. How well. The demo is built to answer those, and it usually does.

Then the harder question arrives, the one that decides whether the platform can run in a regulated environment at all. Not what can the agents do. What can an agent *not* do without it being recorded. Any security team worth its budget asks it. The platforms that stall in review do not stall because the question is a surprise. They stall because they have no architecture that answers it.

A fleet of autonomous agents is two new things at once. It is a new attack surface: every agent calls models, reaches tools, and touches data. It is also a new audit surface: every action an agent takes is something a regulator, an auditor, or a customer's security team will eventually ask you to account for. Most platforms answer neither question well. The reason is almost always the same. The agents were assembled one integration at a time, each wired to the model and the tools it needed, and an architecture that grew one integration at a time has no single place to stand and see all of it.

This is not a capability problem. It is an architecture problem. And it is the same problem whether you are building the fleet yourself or buying one.

A fleet, not a monolith

Start with a definition, because the rest of the paper depends on it.

A *harnessed agent* is an AI agent with four things fixed before it ever runs. A scoped persona that sets what it is for. A scoped set of tools that sets what it can reach. A scoped memory that sets what it can recall. And a binding to a single control point that it cannot make a model call without crossing. The first three scope the agent. The fourth governs it. An agent with the first three and not the fourth is contained. It is not governed. The difference is the subject of this paper.

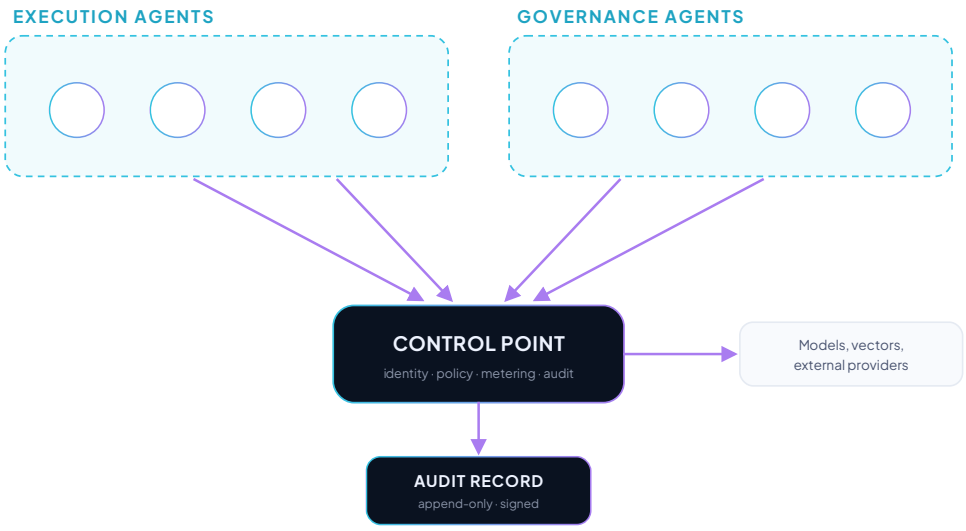
Xiaotime Labs runs a fleet of harnessed agents. They fall into two groups by the work they do. One group executes: it implements changes, reviews code, remediates findings, ships software. The other group governs: it interviews the organization for its risk context, projects controls across frameworks, monitors posture, operates the audit workflow. It is tempting to call these two systems. They are not. They are one fleet, observed from two sides.

This is where Everything as Code and Governance as Code stop being two ideas. Everything as Code is what the fleet looks like from the execution side: infrastructure, pipelines, and now the act

of building software itself, expressed as code and run by agents. Governance as Code is the same fleet from the governance side: controls, policy, and evidence, expressed as code and run by the same kind of agent through the same control point. They are not two products that integrate. They are one fleet read twice.

So governance is not a set of agents standing next to the work, watching it. That model has a gap built into it: the watcher is itself unwatched. Governance, done correctly, is a *property*. It is the property that every agent's every model call is attributed to a named identity, metered against a budget, and written to an audit record, because the call cannot physically happen any other way. An agent does not choose to be governed. It is governed the way a process is governed by the operating system it runs inside.

That property does not come from the agents. It comes from what they all share.



Execution agents and governance agents alike. One control point every call crosses. One record.

Four primitives a security team can defend

Beneath the fleet sit four primitives. Each one is something a security team has defended before, in a context that had nothing to do with AI, and can defend again. None of them is novel on its own. The architecture is novel only in that all four are present, in production, at the same time.

1 Transport

Every deployment runs inside its own isolated private network. The agents are not reachable from the public internet. There is no public attack surface to harden, because there is no public surface. This is the oldest idea in the paper, and it is first because nothing above it matters if the transport layer is open.

2 Identity

Access is ephemeral. An agent run does not hold standing credentials. Credentials are minted at the start of a run, scoped as tightly as the run requires, and revoked when the run ends. The window in which a credential exists is the window in which it is used. This is the harness around a single run: the agent holds power for the length of a task and not one moment longer.

3 Chokepoint

Every call an agent makes to a model, a vector store, or an external provider passes through one mediated control point. The control point holds the real credentials, so the agents never do. It stamps each call with the identity of the agent that made it. It enforces which models an agent may use and what it may spend. It writes an audit event for every call. There is no second path. An agent cannot construct a route around the control point, because the control point is where the credentials are, and a call without credentials is not a call.

4 Evidence

Every state transition is written to an append-only, hash-chained, cryptographically signed record. Not a log that can be edited, and not a report assembled after the fact. A canon: ordered, tamper-evident, and complete, because the things that write to it have no way to act without writing to it.

Transport contains the fleet. Identity contains each run. Evidence preserves what happened. The primitive that turns containment into governance is the third one.

Why the chokepoint is load-bearing

Picture the fleet without a shared control point. Each agent integrates with the providers it needs. Each agent holds, or can reach, the credentials for those providers. Each agent is its own small attack surface and its own small blind spot. Ten agents are ten integrations, ten credential copies, ten places to audit, ten things to get right. The eleventh agent costs as much to secure as the first. Security effort scales with the size of the fleet, and a fleet that grows is a security program that never finishes.

Now give the fleet one control point that every agent must call through. The credentials live in one place. The attribution happens in one place. The spending limits, the model rules, the per-call audit events, all in one place. The eleventh agent costs almost nothing to secure, because it inherits the control point the other ten already use. Security effort stops scaling with the fleet. It scales with the control point, and there is one of those.

Follow a single call. An agent picks up a task and needs to reason about it. It makes one call toward a model. The call reaches the control point first. The control point checks the agent's identity, checks the model against what that agent is allowed to use, checks the spend against the budget, attaches the real provider credential, and forwards the call. When the result returns, one audit event is written: which agent, which model, what it cost, when. The agent never saw the credential. The record was not something anyone remembered to write. It was the cost of making the call.

This is the mechanism, and it is worth naming plainly. A shared chokepoint is what gives governance *leverage*. Without it, governing the Nth agent costs what governing the first one did. With it, the Nth agent is governed for free.

It is also what makes Everything as Code and Governance as Code one motion instead of two. An execution agent shipping a code change and a governance agent projecting a control make the same kind of call through the same control point. The audit record does not distinguish between them, and it does not need to. The record is the byproduct of a call that no agent in the fleet can avoid making. Evidence is not collected. It accumulates, because the alternative to producing it is not running.

Recursive integrity: the chokepoint governs the governors

Here is the part a bolt-on tool cannot copy.

The governance agents are on the control point too. The agent that reviews code is itself subject to review. The agent that monitors posture is itself monitored. The agent that operates the audit workflow is itself audited, by the same canon it writes to for everything else. There is no agent in the fleet that sits outside the fleet. The governance layer is not exempt from itself.

RECURSIVE INTEGRITY

There is no agent in the fleet that sits outside the fleet. The agent that enforces the rule and the agent that ships the feature reach the model the *same way*, and land in the record the same way.

We call this the recursive integrity test, and it is a test in the literal sense. A governance tool wired *beside* a pipeline is, structurally, the one component not governed by the thing it governs. It has to

be the exception, because it was added afterward, from outside. A control point the whole fleet shares has no outside.

This is why we run the platform on itself with no carve-outs. Every change to the platform is authored by an agent in the fleet, reviewed by an agent in the fleet, and merged by a human, through the same control point and the same gates a customer's changes pass through. Not as a discipline we impose on ourselves from the side. As a fact of the architecture: there is no second pipeline to use. If the governed fleet did not hold up against the team that builds it, we would have no way to build it. An architecture either passes that test by construction or it cannot pass it at all.

What changes when the fleet shares one control point

When the architecture above is in place, several things that are normally hard stop being hard.

Adding an agent stops adding risk. A new agent is one more named identity on a control point that already exists. It inherits the attribution, the limits, the audit, and the model rules. The fleet grows; the attack surface does not.

A new model provider is a configuration change. Because the agents never held provider credentials, adding or changing a provider happens at the control point, once. It is a single change, not an integration project repeated for every agent.

Cost and abuse are bounded before the fact. Budget caps and model rules are enforced at the moment of the call, not discovered later in a bill or an incident review. An agent cannot overspend or reach an unapproved model, because the control point will not carry the call.

Audit preparation is a query. The evidence an auditor asks for was written continuously, by agents that could not act without writing it. The question "show me every privileged action an agent took last quarter, attributed, with the control that governed it" is a query against a record that was never not being kept. It is a lookup, not a project.

None of these are features added on top. They are consequences of the third primitive. You get them for the price of the architecture, or you do not get them.

The objections, answered first

This argument is contestable, so here are the hard questions before you raise them.

A single control point is a single point of failure. It would be, if it were single. It is not. The control point runs as redundant active instances, and its audit record is durable independently of any one of them. The honest comparison is not one point of failure against zero. It is one point of failure you can see, instrument, and make redundant, against many points of failure spread across a fleet,

most of which you cannot see at all. Consolidation is what makes the failure mode addressable in the first place.

Mediating every call must slow the agents down. The control point's work is identity, policy, and metering: bounded, fast operations next to the model inference they precede. Their cost is paid once per call and does not grow with the fleet. The cost of not having them is paid in incident response, audit fire drills, and enterprise security reviews that stall for weeks, and that cost does grow.

What about an agent that needs something the control point does not mediate. Then the agent does not get it. The scoped harness is the answer and also the boundary: an agent that requires an unmediated path to do its job is an agent that does not ship. This is a constraint, and it is deliberate. The set of things the control point does not mediate is the set of things the fleet cannot do, and we would rather that set be small and visible than convenient and unknown.

About Xiaotime Labs

Xiaotime Labs builds the GRCDevSecOps platform: a governed fleet of AI agents that ships software and the evidence of its own governance in one motion. ICRG, Integrated Cyber Risk Governance, is the governance face of that fleet. The AI-SDLC pipeline is its execution face. This paper describes the substrate they both stand on.

Xiaotime Labs was built by Craig George and Stephan Hundley, practitioners who have worked together for nearly fifteen years. One a working account executive. The other a veteran CISO and security engineer. They built the platform because they needed it, and they run it on itself because the architecture leaves no other option.

Next steps

If you are building or buying a fleet of AI agents, and your environment has to pass a real audit, carry the question the security review turns on. Not what can the agents do. What can an agent not do without it being recorded. If you cannot answer that about the fleet in front of you, the architecture is the place to look.

Read this as an argument. If it holds up against your environment, we should talk.

Contact · info@xiaotimelabs.ai
Learn more · xiaotimelabs.ai/icrg.html

A technical companion to this paper, covering the control-point architecture and implementation detail, is available to prospective customers under NDA.