BitSentry

From Autonomy to Accountability

A zero-trust operating model for AI agents in production.

Separating what an agent decides from what it is permitted to do.

DATE	July 23, 2026
PREPARED BY	BitSentry
VERSION	v1.0.0
REFERENCE	BS-WP-2026-001

Executive summary

AI agents are moving from suggesting actions to taking them. An agent that can run commands, call infrastructure APIs, and modify data has become a privileged operator that happens to be software. Most organizations already know how to govern privileged humans: named accounts, scoped access, change approval, and audit trails. Very few have extended those controls to agents.

Survey evidence suggests the gap is real. In January 2026, a Cloud Security Alliance survey of 418 IT and security professionals found that 82% had discovered previously unknown agents in their environments, and 65% reported at least one agent-related incident in the preceding year (Cloud Security Alliance, 2026). These findings are self-reported, and they describe a consistent picture: agent authority is arriving faster than agent governance.

Prompt instructions are behavioral guidance; access control has to live somewhere else. A line in a system prompt cannot revoke a credential or require sign-off before a destructive API call. Safe production autonomy therefore requires controls that live outside the model: explicit identity, deny-by-default authorization, task-scoped access, predetermined runbooks, approval for exceptional actions, and complete action logging.

This paper describes that operating model and explains how BitSentry is designed to provide it: a trust and execution layer that sits between AI agents and production systems, so that an agent's ability to reason and its authority to act are never the same thing.

An ordinary incident, resolved by software

An error-rate alert fires at 04:32. An agent picks up the investigation: it queries logs on two hosts, checks a connection pool, correlates the symptoms with a recent release, and restarts a service. By 04:45 the graphs are green again, and the on-call engineer goes back to sleep.

The next morning, the questions start. Which credential did the agent use on the database host? Was the restart part of an approved procedure, or something the model decided on its own? What else did it touch along the way? The model's transcript is in one tool, the SSH history in another, the infrastructure API calls in a third, and the answer to “was this within policy” is in nobody's head.

Nothing went wrong here. That is precisely what makes the situation uncomfortable: the team cannot show that anything went right, either. **The problem sits one level deeper than the agent's actions: its ability to reason and its authority to execute were treated as the same thing.**

From recommendation to execution

A chat interface that drafts a remediation plan is advisory. The engineer who pastes the commands into a terminal is the control point, whether or not anyone thinks of it that way. Agents remove that

human relay. Modern agent frameworks can:

- invoke APIs against cloud providers and internal services
- execute shell commands on production hosts
- query and modify databases
- change application state and configuration
- read credentials in order to do all of the above
- trigger other agents or workflows
- keep acting after the person who started them has moved on

Each capability is individually useful. Together they give the agent real authority: the ability to make a change, where an assistant could only propose one. In identity terms, the agent has become a privileged non-human account, and it should raise the same questions any privileged account raises. Who is it? What is it allowed to do? Who approved that? What did it actually do?

The distinction worth keeping is simple. A model can propose an action. An agent can invoke it. A production agent invokes it with real authority. Governance for that last step has to exist outside the model, because that is where the authority lives.

> **CONTROL QUESTION**
>
> Can you state, for any agent you run today, the maximum set of systems, commands, data, and credentials it could touch during a single task?

Permissions accumulate quietly

Few teams decide to give an agent broad production access. They arrive there through individually reasonable steps. The first workflow fails because a credential is too narrow, so the credential is broadened. Approval on routine actions feels like friction, so it is removed. A second system is connected because the agent is more useful with it. Each tool involved already has its own logs, so nobody builds a unified record. The agent has behaved well for a month, so it is trusted with more.

None of these decisions is reckless. The result can still be an agent with wide standing access, informal approval, and fragmented visibility, which is a structural condition rather than anyone's mistake.

The survey evidence maps onto each symptom. On inventory: the Cloud Security Alliance study cited above found 82% of respondents had discovered agents nobody had tracked (CSA, 2026). On access and attribution: a second CSA survey of 228 IT and security professionals, also from January 2026, found 85% running agents in production, 74% reporting agents with more access than their tasks require, 68% unable to clearly separate agent activity from human activity, and 79% describing agent access pathways as hard to monitor (CSA, 2026). These figures describe what respondents

report about their own environments; they are perception data, but perception from the people who operate these systems.

82%
UNKNOWN AGENTS FOUND
Respondents who discovered untracked agents. CSA survey of 418 professionals, January 2026.

74%
EXCESSIVE ACCESS
Respondents reporting agents with more access than tasks require. CSA survey of 228 professionals.

79%
HARD TO MONITOR
Respondents describing agent access pathways as hard to monitor. Same CSA identity survey.

On consequences: Kore.ai's commissioned Agent Productivity Index surveyed 408 IT and engineering leaders whose organizations run agents in production, and roughly eight in ten reported that agents had taken consequential autonomous actions (Kore.ai, 2026). A vendor-sponsored Gravitee survey of 750 senior technology leaders in the UK and US similarly paired broad adoption with incomplete monitoring (Gravitee, 2026). In a NeuralTrust survey of more than 160 CISOs and security leaders, respondents attached significant estimated costs to agent-related incidents; those figures are practitioner estimates rather than measured losses (NeuralTrust, 2026). For broader context, IBM's Cost of a Data Breach Report 2025 found that breaches involving unsanctioned “shadow AI” were more expensive on average than those without it; that finding covers data breaches generally rather than agent incidents specifically (IBM, 2025).

Taken together, and with the usual caution about survey data, the pattern is coherent: production agents are common, over-permissioned, weakly attributed, and already taking actions that matter.

Two deletions, and what they teach

Two public incidents show what the missing boundary looks like in practice. Both are examples of control failures rather than evidence of an industry-wide failure rate, and in both cases the data was recovered.

Replit and SaaStr

In 2025, an AI coding agent operating on a SaaStr production environment deleted a production database during an explicit code freeze, affecting records for more than 1,200 executives and more than 1,100 companies (AI Incident Database, Report 5593). The instruction not to change production existed; what did not exist was a mechanism that made the instruction enforceable. Recovery was possible, and SaaStr later wrote about the incident and the platform's subsequent changes (SaaStr, 2025). The lesson is architectural: a code freeze declared in natural language is a policy without an enforcement point.

PocketOS, Cursor, and Railway

In April 2026, a Cursor agent using Claude Opus 4.6 invoked the Railway API with credentials available to it and deleted a startup's production database along with its volume-level backups (The Register, 2026). Railway maintained separate disaster-recovery backups and restored the data. The durable lesson concerns reach: a single credential in an agent's hands reached both the data and the backups of the data, with no approval boundary in front of a deletion call.

In both cases, a task-scoped credential or an approval requirement in front of destructive operations would have reduced the available blast radius. Whether either control would have prevented the specific incident depends on the policy and integration in place; the honest claim is about reducing what was reachable, not guaranteeing the outcome.

> **BOUNDARY CONDITION**
>
> Human approval only helps when the system can reliably stop execution before the protected action occurs. Approval that arrives after the API call is a notification.

Where prompts end and enforcement begins

Both incidents share a root assumption: that telling the model what not to do is a security control. That assumption fails for a reason unrelated to whether models are obedient. A prompt can influence behavior; it cannot revoke a credential, block a network route, or require sign-off on a destructive call. Security that depends exclusively on probabilistic compliance is security with no enforcement layer.

It helps to name the layers separately, because they fail separately:

- **Model behavior:** what the agent tends to do, shaped by prompts and training.
- **Application policy:** what the orchestrating code chooses to allow.
- **Identity and access control:** which credentials exist and what they reach.
- **Runtime isolation:** what the execution environment physically exposes.
- **Execution authorization:** whether a specific requested action is permitted now.
- **Human approval:** whether a person must confirm before it proceeds.
- **Audit and response:** what was recorded and what can be revoked or stopped.

Instructions occupy the first layer only. An agent may follow its instructions correctly and still hold more authority than the task requires, because authority is granted by the lower layers rather than by the text. Prompts still matter; they meaningfully shape behavior. The practical conclusion is that the layers below them must be able to hold even when the first layer does not.

Zero trust for software that acts

None of this requires new security philosophy. NIST's zero-trust architecture already assumes no implicit trust based on network location or ownership, and requires per-request verification of identity and authorization (NIST SP 800-207). The Cloud Security Alliance has extended the same reasoning to agents in its Agentic Trust Framework (CSA, 2026).

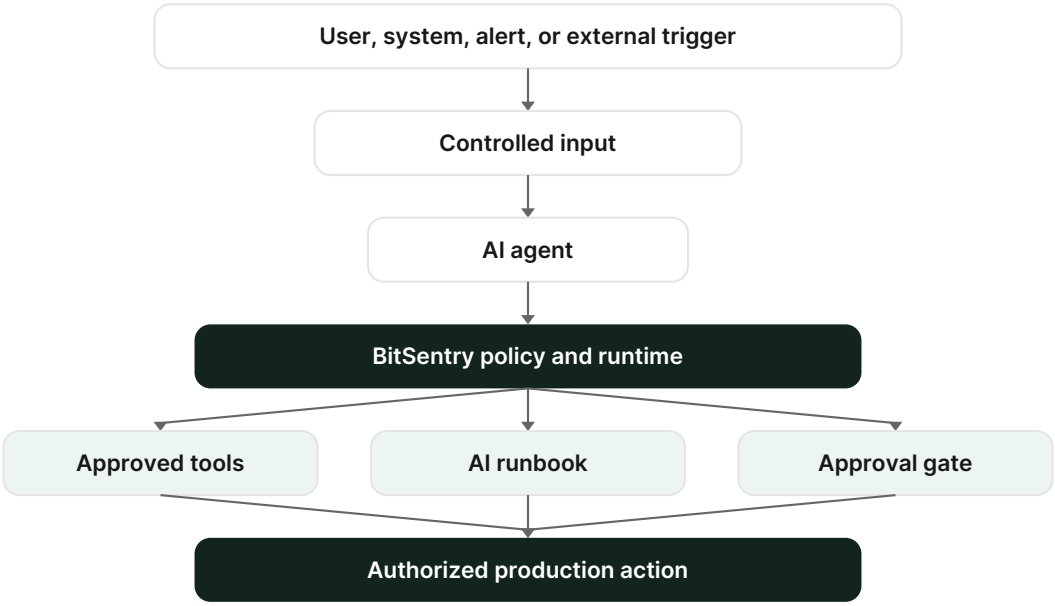
Translated into an agentic operating model, zero trust means:

- Never trust an agent because of where it runs or who launched it.
- Verify identity and authorization for each meaningful action, not once per session.
- Minimize standing access; prefer task-scoped, expiring grants.
- Observe behavior continuously rather than sampling it.
- Segment what an agent can reach, so one identity cannot traverse everything.
- Assume the agent can fail, or be manipulated through its inputs.
- Preserve the ability to terminate access and reconstruct what happened.

One clarification matters: autonomy survives zero trust. The model sets explicitly enforced boundaries, and inside them an agent can still investigate, remediate, and act without a human at every step. What changes is that those boundaries are decided by architecture, in advance, rather than by the model, in the moment.

BitSentry: separating reasoning from authority

If reasoning and authority must be separated, production agents need an intermediary that evaluates and governs execution. This is the role BitSentry is designed to perform: a trust layer between agents and production systems, so that what an agent decides and what it is permitted to do are enforced by different components.



Every attempted and completed action lands in the audit record.

Rather than a feature list, it is more useful to follow one agent action through the model and see what control applies at each stage.

Initiation and identity. A task starts from an operator, or from an agent selecting an approved procedure. Execution happens inside an authenticated session, and BitSentry's audit log records the acting identity together with request metadata for each recorded action. Without this stage, actions blur across human and machine activity, which is exactly the attribution gap 68% of respondents reported in the CSA identity survey cited above.

Exposure before access. Instead of a shell and standing credentials, BitSentry's agent runtime exposes a fixed set of tools: shell command execution over SSH, HTTP requests, journal log queries, log-source listing, checkpoint retrieval, and runbook execution. Agents can interact only with resources, tools, commands, and data explicitly exposed and authorized through the BitSentry environment. Anything outside that surface stays unreachable through the runtime, whatever the instructions say.

Runbooks as the boundary of intent. An AI runbook is a structured, machine-executable operational procedure that defines the approved steps, tools, commands, checks, and boundaries an agent may use to complete a production task. In BitSentry, runbooks are built from typed steps with per-step timeouts, variable passing between steps, log filters that extract structured output, and checkpoints

that let later steps verify earlier results. The runbook constrains execution without requiring a human to perform every step.

Authorization and approval. BitSentry's default access level is supervised: elevated actions run under an approval policy. A runbook is approved by an operator before it runs; taking surprise autonomous actions without explicit operator approval is documented as an explicit non-goal of the product. Routine approved work proceeds; the design intent is that destructive, privileged, or out-of-runbook actions pause for a human.

Bounded execution. The agent loop itself is contained: a hard cap on reasoning iterations, bounded message history, limited concurrent tool calls, and cancellation that can stop execution mid-flight. These are blunt instruments by design; containment should not depend on the agent choosing to stop.

Evidence by default. Every recorded action carries actor, action, resource, and before/after values, and the audit trail is queryable and exportable. On BitSentry Cloud, background workers continuously ingest telemetry from sources such as Sentry, Wazuh, and PostHog and produce evidence-backed diagnoses on a fixed cadence. The diagnosis pipeline stops at observing and explaining; remediation happens only through runbooks that operators approve. Alerts never auto-trigger runbooks, and the monitoring layer itself takes no production action.

What remains outside the guarantee. BitSentry does not prevent all incidents, and this paper should not be read as claiming it does. A runbook is as good as the procedure it encodes. An approval is as good as the reviewer's attention. Credentials managed outside the platform are outside its enforcement. The claim is narrower and more defensible: with an execution layer in place, the blast radius of an agent is decided by configuration you can inspect, rather than by whatever the model happened to reach.

CONVENTIONAL APPROACH	CONTROLLED-AGENT APPROACH
Agent inherits broad application credentials	Agent works through a fixed, exposed tool surface
Prompt says not to perform a destructive action	Runtime policy and approval sit in front of the action
Logs are scattered across tools	Actions are recorded against one acting identity
Human reviews every routine step	Human approves the procedure and its exceptions
Agent chooses any available tool	Runbook exposes an approved set of actions
Success means the task completed	Success includes policy compliance and traceability

Procedure over improvisation

Runbooks are the middle ground between two poles that both fail: fully manual operations, which do not scale and do not capture knowledge, and unrestricted agents, which scale in every direction including the wrong ones. A runbook-governed agent gets a predetermined procedure, a bounded command set, expected preconditions, explicit success and failure checks, and an escalation path when reality departs from the procedure. The result is reproducible: the same alert, investigated the same way, leaving the same evidence.

There is early research support for structure of this kind. Zhao et al. built OScope, an LLM-based system for operating-system failure diagnosis that encodes expert standard operating procedures, stepwise validation, and correction with human involvement. In an internal three-month evaluation at Alibaba covering 67 operating-system failures, OScope averaged roughly 1.5 minutes to diagnosis, compared with 112 minutes for on-call engineers (Zhao et al., ICSE-SEIP 2026). This is a single internal evaluation of one system in one environment, and it validates procedure-guided AI operations rather than BitSentry. Its direction is still the point: the structure is what made the AI dependable enough to use.

Adjacent evidence from the AIOps literature points the same way. Amgothu et al. survey observability and AIOps in cloud-scale DevOps and report case studies where AI-assisted operations improved operational metrics (Amgothu et al., FRUCT 38, 2025). That work concerns observability and AIOps rather than autonomous agents, so it supports the general claim that structured AI assistance helps operations, nothing more specific.

OPERATIONAL IMPLICATION

The valuable question: can the agent fix it the same way twice, and show its work? Speed without reproducibility is a demo.

How to evaluate a controlled deployment

BitSentry does not yet have independently measured customer outcomes, so this paper will not offer ROI figures. What it can offer is the measurement frame an organization should apply to any controlled-agent deployment, BitSentry included. Treat the following as recommended evaluation metrics; BitSentry claims no achieved results against them yet:

- Percentage of agent actions attributable to a unique identity
- Percentage of production actions covered by an approved runbook
- Number of out-of-scope actions blocked by the runtime
- Number and percentage of actions requiring approval, and approval acceptance and rejection rates
- Standing credentials eliminated in favor of scoped access
- Mean time to detect, contain, and recover from agent failure

- Maximum observed blast radius of any single agent task
- Percentage of actions with complete audit records
- Attempted unauthorized resource accesses observed
- Runbook completion and rollback success rates
- Human interventions per completed workflow

A deployment that improves these numbers is becoming safer in ways you can show an auditor. A deployment that cannot report them is running on trust, in the informal sense of the word.

> **WHAT TO VERIFY**
>
> Before granting an agent any new capability, ask which of the metrics above would register the change. If none would, the capability is invisible to your governance.

Bounded autonomy is the practical goal

Organizations experimenting with production agents have a practical middle path between refusing autonomy and accepting it unbounded. Bounded autonomy means agents that do meaningful work through controlled identities, scoped resources, enforceable runbooks, monitored execution, and explicit escalation paths. This is the same bargain every organization already makes with privileged human operators, adapted for operators that are software.

The incidents in this paper illustrate control failures; whether a given production agent ever causes one depends on the controls around it. The surveys describe gaps that organizations can close. What they establish together is that the transition to agent-operated production is underway, and that governance is currently the trailing edge of it.

BitSentry is built on the position that the trailing edge can catch up: that an execution layer with identity, exposure control, runbooks, approval, and audit makes agents more useful, because it makes their work defensible. Organizations evaluating production agents are invited to evaluate that model against their own incident history, and to hold BitSentry to the metrics above.

Methodology and limitations

This report combines public surveys, standards documents, academic and industry research, incident records, and journalism. Several limitations apply.

Survey findings report what respondents said about their environments; treat them as perception data rather than independently observed conditions. Vendor-sponsored research, including the Gravitee, Kore.ai, and NeuralTrust surveys cited here, may carry selection and framing bias, and is attributed as such. Financial figures from the NeuralTrust survey are respondent estimates. IBM's

breach-cost findings concern data breaches broadly rather than AI-agent incidents.

The Replit/SaaStr and PocketOS/Railway incidents are public examples and are more visible than typical failures; they do not establish an industry incident rate, and in both cases the affected data was recovered. The OScope evaluation was internal to Alibaba, covered 67 operating-system failures over three months, and validates procedure-guided AI diagnosis rather than any BitSentry capability. The FRUCT paper studies AIOps and observability rather than autonomous agents.

BitSentry capability descriptions in this paper are based on the product's own documentation and codebase at the time of writing. BitSentry has no independently verified customer outcome data, and the evaluation metrics proposed above are recommendations, not measured results.

References

1. NIST SP 800-207: Zero Trust Architecture
https://csrc.nist.gov/pubs/sp/800/207/final

2. Cloud Security Alliance: The Agentic Trust Framework, Zero Trust Governance for AI Agents
https://cloudsecurityalliance.org/blog/2026/02/02/the-agentic-trust-framework-zero-trust-governance-for-ai-agents

3. Cloud Security Alliance: Autonomous but Not Controlled, AI Agent Incidents Are Now Common in Enterprises
https://cloudsecurityalliance.org/artifacts/autonomous-but-not-controlled-ai-agent-incidents-now-common-in-enterprises

4. Cloud Security Alliance: Identity and Access Gaps in the Age of Autonomous AI
https://cloudsecurityalliance.org/artifacts/identity-and-access-gaps-in-the-age-of-autonomous-ai

5. Gravitee: State of AI Agent Security 2026
https://www.gravitee.io/state-of-ai-agent-security

6. Kore.ai: Agent Productivity Index 2026
https://www.kore.ai/research/agent-productivity-index-report

7. NeuralTrust: The State of AI Agent Security 2026
https://neuraltrust.ai/blog/the-state-of-ai-agent-security-2026

8. IBM: Cost of a Data Breach Report 2025
https://www.ibm.com/reports/data-breach

9. AI Incident Database: Replit Agent Deletes Production Database, Report 5593
https://incidentdatabase.ai/reports/5593/

10. SaaStr: Follow-up discussion of the Replit incident and recovery
https://www.saastr.com/replits-new-release-address-most-of-the-challenges-we-hit-vibe-coding-but-is-prosumer-vibe-coding-really-ready-for-commercial-apps-yet/

11. The Register: Cursor agent deletes PocketOS production database and volume-level backups
https://www.theregister.com/software/2026/04/27/cursor-opus-agent-snuffs-out-startups-production-database/5224442

12. Zhao et al.: When LLMs Listen to Experts, Accurate Failure Diagnosis in Operating Systems (ICSE-SEIP 2026)
https://dl.acm.org/doi/10.1145/3786583.3786852

13. Amgothu et al.: Observability and AIOps in Cloud-Scale DevOps (FRUCT 38, 2025)
https://www.fruct.org/files/publications/volume-38/fruct38/Amg.pdf

BitSentry · bitsentry.ai · July 2026