TEMPLATE · FILL-IN REFERENCE

Agent Incident Runbook

Ed Schaefer · Version 1.0 · August 2026 · edward-schaefer.com/templates/agent-incident-runbook · Free to share

No agent-caused incident yet doesn't mean your team is safe. It may just mean nobody's written down what happens on the day one occurs. I learned that running my own systems: an ordinary bug once burned through millions of tokens in a single day, and what actually stopped it was a hard spend cap I'd set months earlier, more out of habit than foresight, not a plan built for that moment. I catch most problems now by building ahead of myself, but some of that is luck, and luck isn't a plan. This template is that plan, filled in ahead of time instead of improvised in the moment. It also sets one rule early: catching the problem and fixing why nothing upstream caught it are two different jobs, and this template asks for both.

That story is a cost incident, but nothing here is scoped to cost. A credential leak, a bad customer-facing change, corrupted data — the same plan covers any of them.

1. What counts as an agent incident

A starting default, generic to any team running agents in real work. Fill in your own line for what "counts" here:

- An agent produced something that shipped, or nearly shipped, and shouldn't have.
- An agent acted outside what it was authorized to do.
- An agent's output cost something real: to a customer, a system, or the team's trust in the work.

Default threshold: if you're asking whether it counts, it counts. Write it down and start the runbook. Decide the severity after, not before.

- Our team's definition: _____

2. How serious, and what to do right now

First, the fork. It picks your immediate path:

	What it looks like	Immediate path
Contained	Caught before it reached anyone or anything outside the team	Run the immediate steps below, then handle the cleanup as routine work. Whether it also gets a review: the near-miss rule, under this table.
Serious	Reached a customer, another system, or can't be undone	Run the immediate steps below, name a lead, and open the severity ladder.

Near-miss reviews. Early on, a contained near miss gets the section 4 review too — reviewing the close calls is how the feedback loop that improves the system gets built. Once that loop is established, review a near miss when it's the first of its kind: a new category of failure, a new way in. A routine repeat of a kind the team has already reviewed can be logged and skipped.

Immediate steps, in order (both paths start here):

1. **Stop the line.** Whoever spotted it stops the flow right there, no permission needed first. This is the Andon-cord rule the Definition of Verified template also names: the authority to stop belongs to whoever's closest to the defect, not whoever has the title. The DevOps Handbook applies the same logic to software directly: "if someone asks me to review their code, I should drop whatever I'm doing." Peer review and postmortems are the Andon cord for a team that isn't standing on a factory floor.

   In software, stopping the line used to mean telling the team, and everyone stopped. Agents don't stop because the people did — a running loop keeps going until something makes it stop. Name the mechanism: - How agent work gets stopped or paused here: _____ (default: announcing in the team channel stops the humans; for the agents, name the kill or pause control for each place they run — if there isn't one, that's the first gap to close)

2. **Cut off what it could reach.** Block, revoke, or rotate the agent's credentials now. Check what they could reach, not only what the agent was instructed to do — that's usually where the real exposure sits. This step doesn't touch the evidence, so it doesn't wait.
3. **Capture the full chain.** Every action the agent took, in order, not just the final output, and the instruction it was working from. Do this before anyone edits it, cleans it up, or explains it away.
4. **Preserve before anyone reruns it.** Agent behavior doesn't always repeat: the run you just captured may be the only account you get. That's why preserving it comes first, not after. A full log lets you replay the chain step by step later, even on the runs a rerun can't reproduce.
5. **Then remediate.** Revert or roll back whatever can be reverted — after capture, because remediation can destroy the evidence steps 3 and 4 protect. The exception is genuinely time-critical harm still in progress: money still draining, data still leaving. Stop that first, and capture what remains.

If serious, the severity ladder. Classify now — it sets who's told (below) and how hard the team responds. This ladder governs response: how hard to react once something's gotten through. Prevention — what should have caught it before it shipped — is a different question, answered by Definition of Verified's blast-radius tiers, not this ladder. Unsure which tier? Default to the higher one. Reclassify after, never during.

- **SEV1: critical.** Customer-facing, data, or credential exposure; anything irreversible. Full review before anything similar runs again; told immediately.
- **SEV2: major.** Contained, but real cost: rework, a system down, a customer confused. Full review before the same task type runs again.
- **SEV3: minor.** Caught internally, no lasting effect. Logged; reviewed at the next regular check-in.

- **SEV4: cosmetic.** Worth a line in the log so the pattern stays visible. No separate review needed.

3. Who's told, and in what order

This is org-specific; fill it in rather than guessing at it. Higher severity moves people up this list, not down: a SEV1 belongs at "told immediately," not "told within the day."

- Who leads: _____ (default: whoever invoked this runbook, until they hand off to someone they name). Leading means running the steps and making the calls. It also means being the single voice for outside comms, not another pair of hands in the work.
- Told immediately: _____
- Told within the day: _____
- Told at the next regular check-in, no sooner: _____ (default: nothing on this list waits; if it isn't urgent enough to tell someone today, ask why it's on this list at all)
- The model or tool vendor, when relevant: _____ (default: if the behavior looks like it came from the model or tool itself, not just what you asked it to do, tell them too; this isn't only an internal problem)

4. The blameless-review rule

The instinct once the line stops is to praise whoever caught it. That instinct isn't wrong, but it isn't the review. The same question the source post asks about human heroics applies directly to agents: instead of focusing on the firefighter, focus on the fire. Why did it start? What broke down before this reached the person who caught it?

A review that ends at "good catch," without asking why nothing upstream caught it first, still counts as a hero story. Hero stories are exactly what let the same failure happen again.

Rule for this team: no incident review closes without an answer to "what should have caught this before a person had to."

If AI helped draft the review, a human checks it before it circulates. AI-drafted reviews can slide into naming the person in the transcript instead of the system around them — a human confirms it stayed blameless before anyone else reads it.

5. What changes afterward

Checklist: default items below, add your own.

- [ ] The instruction, prompt, or guardrail that let this happen is identified and changed.
- [ ] The check that should have caught this before it shipped is named. If it existed and didn't run, find out why. If it didn't exist, add it (cross-reference: Definition of Verified, section 2 — its blast-radius

tiers set what "should have caught this" means; that's the prevention axis, not the severity tier you classified above).

- [ ] The person who caught it is thanked. That's separate from the fix above, not a substitute for it.
- [ ] The fix itself is verified by someone other than the person who made it, same as any other work (Definition of Verified, section 1).

This is a fit question, not a recipe

The shape above is a strong starting plan, not a finished one. Whether your team's actual practice would hold up against it, and where the gaps get expensive, is what the Engineering Ways of Working Diagnostic is built to find.