PURETENSOR
INTELLIGENCE DISTILLED

Heat Is the Only Signature

Agent-Spawned Processes That Pegged a Core While the Node Looked Healthy

Paper: PT-R-2026-044 v1.0

Author: PureTensor Ltd

Date: 8 June 2026

Status: Published

Abstract

Coding agents run all day on our flagship workstation, interactively and unattended, and they leave things behind. Over about eleven weeks, from 24 March to 8 June, we found four cases in which processes spawned by an agent burned CPU for long periods while nothing failed and nobody noticed until someone looked at a temperature. Two were scans: three or four recursive text searches (the April ones orphaned), killed by hand. Two were genuine spin loops, processes that pin a core and make no progress: a shell wait loop with an empty body that ran for about three hours on 29 May, and the server process of an editor plugin, about 21 hours old when found on 8 June, which the diagnosing session concluded was looping on errors against a directory that no longer existed.

The failure class is a process that is running, not crashing, not leaking memory and doing nothing. "Healthy" here has a narrow meaning: on 8 June a first node-level snapshot of 97 running services (none failed), the containers, the cluster and the GPUs concluded that the node was healthy. We have no per-process liveness result for any offender. Only a per-second trace of on-CPU state found the process. Its removal moved the control temperature of the CPU from a baseline of about 76 C (peaks of 81 C) to 42.5 C, and the chassis fans from a 440 to 545 RPM range to about 370 RPM. Heat, fan speed and power draw were the only signature.

We built two layers in response. An authoring guard blocks no-op loops before an agent's shell tool runs them (19 test cases, 7 to block and 12 to allow). A runtime reaper, a one-minute systemd timer in plain shell, kills processes that peg a core while their I/O, system-call and voluntary context-switch counters stay frozen. A synthetic runaway was reaped at 325 seconds against a 300-second gate. That is the whole of the validation: no real incident has yet been caught by the reaper, and the guard would not have caught the 8 June process. We also report where our working story was wrong: two of the four incidents were scans the reaper does not cover, and "21 hours of spinning" is a process age, not a measured spin duration.

1. A Class of Failure With No Failure in It

Most monitoring is built to notice things that stop: a process exits, a port closes, a probe times out, memory climbs until the kernel intervenes. This paper concerns a process that keeps running with steady resources and does nothing useful at full speed.

The machine is a Threadripper-class workstation with 64 hardware threads and two RTX PRO 6000 Blackwell GPUs, also a Kubernetes agent node running about 97 systemd services. Coding agents spawn child processes on it all day: shells, searches, wait loops, plugin servers. When a parent session ends, children can be reparented to the init process and carry on. Nothing is watching them once the session ends.

In each of the four incidents below, a person noticed something physical or thermal; no alert fired. Can a running, conventionally healthy process that does nothing be told apart from legitimate compute by what it fails to do, rather than by how busy it is? Busyness is the wrong axis, because a training job or a backup also pegs cores; the right axis is whether the work is going anywhere.

We should be exact about "healthy", because our first framing overstated it. No artefact records a listening port or a liveness-probe result for the offending processes, so we make no claim that they passed liveness checks or held a port. The supported claim is weaker and still useful: on 8 June our first snapshot of the node reported it as steady-state and healthy, because no service had failed, GPU utilisation at that instant was low, and load was modest. On 29 May the load was about 3 on 64 threads while the control temperature stood at about 82 C.

2. The Four Incidents

#	Date	What ran	Count	Duration seen	Found by	Class
1	24 March	recursive grep spawned by an agent subagent, searching a home directory for configuration strings of a tool	4	5+ min each, about 416 MB resident each	control temperature of 77.9 C during a sensor check	scan
2	16 April	recursive grep spawned by a subagent doing manifest discovery, orphaned	3	each at 99.9% CPU for 37+ min	operator report of 71 C	scan
3	29 May	shell busy-wait loop with an empty body, waiting for a marker file that never appeared	1	about 3 h at 99.9%	control temperature about 82 C at load about 3 on 64 threads	spin
4	8 June	server process of an editor plugin, running against a deleted working directory	1 (plus its parent wrapper, 2 killed)	about 21 h process age, one core at 99.9%	operator noticed periodic power peaks and fan ramps	spin

Table 1. The four incidents. The March and April durations are minimums.

In March the control temperature read 77.9 C during a routine sensor check. Four recursive greps, spawned through the shell by an interactive agent session rather than its dedicated search tool, were each near 100% of a core. They were children of a live session, not orphans. All four were killed at the operator's request, with no code or configuration change.

In April the operator reported 71 C. By the time we investigated, the sensors showed cores at 39 to 46 C, so the 71 C figure came from another sensor source or another moment, as the report itself says. Three orphaned greps over manifest and script directories were each at 99.9% CPU after 37 or more minutes; the report's own timestamps disagree (a 07:43 UTC header, an 08:04 spawn time), so 37 minutes is the report's figure. They were killed; the only follow-up recorded was to consider timeouts or limits if it recurred.

These two are scans: they pinned cores because they were reading large trees, and a scan makes steady forward progress. The reaper we built detects the absence of progress and would not have caught either. They are a separate class that our runtime layer does not address.

The 29 May incident was a model-written wait loop of the form "until the marker file exists; do nothing; done", with no sleep. The commit message describes the shell as detached from its parent session, though the session report lists a live parent session; the sources disagree. The marker never appeared because the task that would have produced it had died at 09:52 (UTC, per the report), before the report was written and well after the loop started. Killing the loop dropped the control temperature from 81.9 C to about 49.5 C within about 8 seconds, with load falling and no sibling spinners.

3. The Incident the First Fix Missed

The 29 May response (Section 5) had added a guard and a first reaper, and neither applied here.

The operator reported periodic wattage peaks and fan-speed rises. The diagnosing session first took an inventory: 97 systemd services with none failed, containers, pods, more than 50 Kubernetes cron jobs, listening ports and GPU state. It concluded the node was healthy. The GPUs were about 94% memory-resident, holding an inference server sharded across both cards, but near-idle in utilisation, and GPU temperatures stayed between 35 and 47 C. The only requests in the inference server's journal were health and metrics probes, with zero inference.

Meanwhile the CPU control temperature oscillated between 63 and 81 C per minute in our own thermal log while the GPUs stayed flat, and the fans followed in lockstep. (The 76 C in Table 2 is the report's rounded baseline; 63 to 81 C is the minute-level oscillation.) The top consumers were the hourly backup tool at 211% and one process at 101%. Only per-second on-CPU tracing over 80 seconds settled it: that process was in the running state in every sample. It consumed 3.04 CPU-seconds in 3 seconds, a single hot thread at 99.9%.

It was the server process of a plugin for a coding agent, about 21 hours old, its parent reparented to init. The plugin had been updated at 16:36 (day not recorded) and the node diagnosed about two hours later. The update swapped the plugin directory: the live one renamed to a backup, the backup deleted, new code put in place. The server process running before the update was never restarted. Its working directory showed as deleted. The session concluded that it had fallen into a tight error loop against files that no longer existed; the loop's content was never inspected, only its running state and deleted directory. It ignored SIGTERM and needed SIGKILL.

Measure	Before	After
Control temperature	about 76 C baseline, peaks of 81 C	42.5 C steady
Chassis fans	440 to 545 RPM	about 370 RPM
Top process by CPU	the offender at 101%	a normal service at about 30%
Pegged core	1 at 99.9%	released

Table 2. Before and after removal of the 8 June runaway. Single readings; the baseline fell 33.5 C.

The first node-level checks reported healthy because the process did nothing a check looks for. It did not crash, memory did not grow, and no service failed. A working process reads, writes, makes system calls and sleeps while waiting; a tight loop with no system calls never blocks or yields. Whether the 8 June process met that description was not measured, because its counters were not recorded. A utilisation check cannot see the difference either way, because utilisation is what the process maximises. We do not propose thermal alerting as a detector, since many legitimate things heat a machine; temperature and fans were simply the only telemetry that disagreed.

4. What the Record Does Not Pin Down

There is an inconsistency in the 8 June report that we cannot resolve. The plugin update is timed at 16:36 and the diagnosis about two hours later, yet the process is described both as about 21 hours old and as the pre-update process. Those cannot all hold if the update was on 8 June. The record also does not agree with itself on onset: the diagnosis was at about 17:46 UTC, so a 21-hour-old process started about 20:46 UTC on 7 June. Either the age or the day of the update in the report is wrong. The process no longer exists and no process-level log was kept.

We cross-checked against our own metrics, taking the hourly minimum of the CPU control temperature from the node exporter (a Prometheus metrics agent). The floor was 43.25 to 48.9 C from 6 June 12:00 to 7 June 10:00 UTC; it read 51.75 at 12:00 and 48.6 at 14:00 on 7 June; it then sat between 48.9 and 59.3 C from 7 June 16:00 to 8 June 04:00 (one dip to 48.9 at 02:00 on 8 June, otherwise 51.75 to 59.25), and fell to 42.75 C at 18:00 on 8 June, after the kill. The rise at 7 June 16:00 precedes the age-implied start by about five hours.

Time (UTC)	Floor (C)	Time (UTC)	Floor (C)
7 Jun 12:00	51.75	8 Jun 06:00	55.0
7 Jun 14:00	48.6	8 Jun 08:00	52.1
7 Jun 16:00	59.25	8 Jun 10:00	45.0
7 Jun 18:00	51.75	8 Jun 12:00	53.25
7 Jun 20:00	58.25	8 Jun 14:00	52.25
7 Jun 22:00	57.6	8 Jun 16:00	52.5
8 Jun 00:00	58.4	8 Jun 18:00	42.75
8 Jun 02:00	48.9	8 Jun 20:00	44.0
8 Jun 04:00	57.4	8 Jun 22:00	43.25

Table 3. Hourly minimum of the CPU control temperature, every two hours. The kill came before the 18:00 value on 8 June.

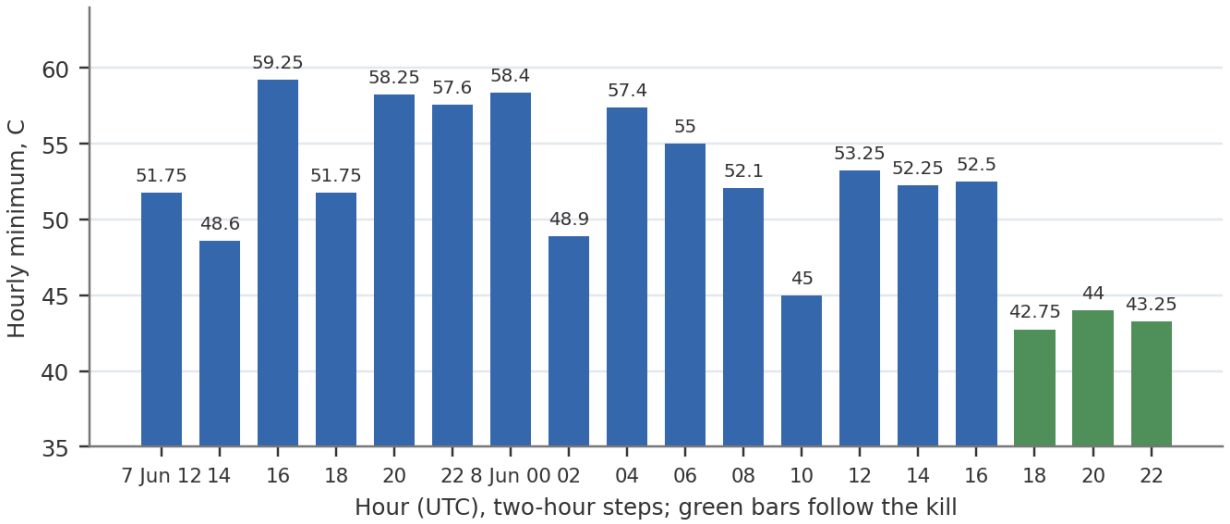


Figure 1. Hourly minimum CPU control temperature, 7 to 8 June (UTC). The floor is higher on most readings for roughly a day, with dips at 7 June 14:00, 8 June 02:00 and 10:00, and returns to its earlier level after the process was killed. Values are in Table 3.

The pattern is consistent with a long-running spinner, but the series is noisy (backups and cron bursts also raise temperatures) and the 45.0 C at 10:00 on 8 June is hard to square with a

continuously pegged core. We therefore treat 21 hours as the process's age at diagnosis and make no claim about how long it spun. An internal summary stated that lifetime CPU accounting showed a full core for the entire 21 hours; the session report does not contain that, and we could not reproduce it.

5. What We Built

The first layer is an authoring guard, added on 29 May: a pre-execution hook on the coding agent's shell tool that blocks loops whose body is a no-op. The block message tells the agent to do real work with a sleep and a timeout. Its test battery had 19 cases, 7 to block and 12 to allow, including sleep-throttled loops, read loops and bounded for loops; all behaved as intended. The count comes from the session report, as no committed test script existed at that commit. The guard covers only shell loops in the agent's own shell tool and would not have caught the 8 June process.

The second layer watches what is already running: a systemd oneshot service on a 60-second timer, written in pure shell reading the process filesystem, with no dependencies.

Tier	Matches	Action
argv	no-op loop body in the command line	kill after CPU and age gates
shell	a bare shell interpreter	kill after stricter CPU and age gates
runtime-deleted	a process whose executable or working directory is deleted, pegging a core with zero forward progress, age at least 300 s	kill
runtime-noprogress	any process pegging a core with zero forward progress, no deletion needed	log and alert only unless armed

Table 4. Reaper tiers as of 8 June, with only the production age gate of the deleted-code tier given.

The first two tiers match on how the command line or interpreter looks, and did not help on 8 June, because the offender was neither a no-op loop nor a bare shell. The last two use a forward-progress test.

The reaper samples a candidate twice, a few seconds apart, and reads three things: CPU time in jiffies (the kernel's accounting ticks); one I/O counter that covers bytes moved and system calls made; and the count of voluntary context switches, the times the process gave up the processor to wait. A process has no progress when neither the I/O-plus-syscall delta nor the voluntary context-switch delta moves. A working process is expected to move at least one counter in the window. To limit cost, only processes above a lifetime CPU prefilter are probed, and one shared window serves every candidate.

A reaper that kills needs more caution than a detector. The deleted-code tier needs three simultaneous facts: deleted code, a pegged core and no progress. A daemon whose binary was merely replaced by a package upgrade also shows a deleted executable, but it keeps doing I/O and must be spared. Version 1.1.0 would have reaped on deleted code plus a pegged core alone; 1.1.1 added the no-progress requirement before any rollout, and its commit message states the intent: a busy process with a deleted working directory is not reaped, and a no-progress one is. Further protection comes from a conservative allowlist and a GPU exemption.

6. What the Validation Shows

Time (UTC, 8 June)	Test	Outcome
17:55:22	busy loop with deleted working directory, dry run, process age 5 s	would reap (1.1.0, no progress test yet)
17:56:11	same, age 54 s, 101% CPU	would reap
17:56:18	same, age 60 s	reaped
18:16:18	same shape with 1.1.1, age 6 s, dry run	would reap, flagged no I/O and no yields
18:21:39	same, real run, age 325 s, 100% CPU	reaped

Table 5. Synthetic runaway tests in the reaper log. The age gate was lowered for the first four lines; the last shows the production 300 s gate.

These are five log lines about one process shape, a Perl busy loop in a deleted working directory, and it is on this synthetic process that the frozen-counter signature was observed. The first four used a lowered age gate (the 6 s dry run cannot be a would-reap under a 300 s gate) and, in the first, ran before the progress test existed. The last is the only run with the production gate, and it reaped at an age of 325 seconds.

The latency bound is simple. The process must reach 300 seconds of age, then waits up to one 60-second timer interval and a 4-second probe: 364 seconds, about 6 minutes in the worst case once the gates are met, before prefilter and per-run overheads. Against the 21-hour process age of 1,260 minutes, that is a ratio of about 210 (1,260 divided by 6). This compares a configured bound with the time a person took; it is not a measurement of a real catch. As at 8 June, no real reap appears in the reaper's log on the workstation.

7. What These Numbers Will Not Carry

The evidence is four incidents on one machine, two of them spin loops. The validation is five log lines with a single process shape. There is no real-incident test of the reaper and no precision or recall figure. The temperature figures are single before-and-after readings, the thermal baseline is confounded by backup and cron activity, and the March and April readings come from different sensors or moments. The causes given for 8 June are the diagnosing session's reading.

The claims that survive: the failure class exists on this machine, a busy, useless, healthy-looking process whose only signature is heat; the two-layer design is as described; the five-line validation behaved as designed; and the 8 June before-and-after thermal readings are real.

The claims that do not: that the processes held a port or passed liveness probes, since no artefact records either; that there were four spin incidents, since there were two and two scans; that the 8 June process spun for 21 hours, since that is its age and the onset is unresolved; that the 8 June process would have met the frozen-counter test, since its counters were never recorded; that detection fell from 21 hours to 5 minutes as an observed result, since 5 minutes is a configured gate plus one synthetic run; and any false-positive rate, since the reaper has not met a real incident.

Two points remain open. Whether the March and April scan class is a reaper concern is undecided; detecting it would need a signal other than zero progress, since a scan has abundant progress. And the source of the 8 June orphan is not fixed: restarting plugin servers when their directory is swapped was listed as a next step and has not been done.

8. What We Changed

The guard and the reaper are in place on the workstation, with the reaper at version 1.1.1. Three of its tiers (Table 4) kill: the command-line, shell and deleted-code tiers. The fourth, which needs no deletion, only logs and alerts unless armed. We added no thermal alert, for the reasons in Section 3.

Next: restart plugin servers when their directory is swapped, which removes the cause of the 8 June incident rather than its effect; look for a signal for the scan class that does not depend on zero progress; and record the first real catch with its measured latency, so that the 5-minute figure stops being a configuration setting.

9. Methods and Artefacts

Evidence came from four session reports (24 March, 16 April, 29 May, 8 June), the reaper's repository history and script as of the 8 June commits, the reaper's log, and Prometheus range queries of the node exporter's hardware-monitor temperature (CPU control temperature, hourly minimum over one-minute subqueries, two-hour steps, UTC). The reaper's version at publication was 1.1.1; commits were made at 12:22 on 29 May (1.0.0) and at 18:59 and 19:17 on 8 June (1.1.0 and 1.1.1), local time. The hardware was one workstation with 64 hardware threads and two Blackwell workstation GPUs. No statistics beyond arithmetic were used. Session reports, repository history, the reaper log and the temperature series are retained; the offending process itself was not.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-044. The incidents occurred between 24 March and 8 June 2026, and the reaper validation was recorded on 8 June 2026; the session reports, repository history, reaper log and temperature series are retained.