THE DEAD MAN SWITCH

Technical Breakdown

The engineering behind Build Note N-014, Nobody Was in Florida. A companion to the essay at michaelgaff.com/journal/nobody-was-in-florida.

1. System overview

The production server is a single MacBook that runs unattended in a Florida house, handling scheduled jobs for a small operation: donor outreach, financial reconciliation, and the software that powers the rest of the build. It is normally reached over a private mesh network from wherever the operator happens to be. There is no on site staff. If the machine stops answering, the nearest human is typically hours away by car and up to a full day away by plane.

On the incident date, the machine's load average rose to roughly 128 against a normal baseline of about 6, and the machine stopped responding to the network entirely. This paper describes the failure, why the obvious remote fix does not work on this hardware, and the three layers built afterward so the same failure resolves itself in minutes instead of days.

2. What actually happened

- A background process responsible for talking to the local mail client began hanging on every invocation instead of returning.
- A scheduler kicked off that process on a fixed interval, roughly every fifteen minutes, without a timeout.
- Each hung invocation stayed resident. Over several hours, hung processes accumulated into the hundreds.
- Load average climbed in step with the process count until the kernel could no longer service new network connections.
- The machine became unreachable by every remote channel at once: no SSH, no mesh network, no web app.

The proximate cause was a stuck integration with a local mail app. The systemic cause was the absence of a timeout and the absence of any process that watched the watcher.

3. Why the obvious fix does not apply

A smart power plug is the standard remote recovery tool for a headless server: cut power, restore power, the machine performs a cold boot. It does not work here, and the reason is the hardware itself. The production machine is a laptop with an internal battery. Removing wall power does not power the machine off. It silently switches to battery and continues running in its stuck state, now also draining. A

plug placed in front of a laptop, in other words, is not a reset button. Any recovery design for this class of hardware has to originate from software running on the machine itself, because there is no external power path that reliably restarts it.

4. Layer one: the health snapshot

A lightweight job already ran every fifteen minutes to publish a snapshot of machine health (disk, memory, job status) to a small dashboard. After the incident it gained one more field: a live count of the specific helper scripts implicated in the failure. A count that climbs past a small threshold is now visible on the dashboard roughly an hour before load average itself would show a problem, giving the earliest possible signal without taking any action.

5. Layer two: self surgery

The same snapshot job was given narrow, explicit authority to act on its own reading. The rule: if load average exceeds a fixed threshold and the helper script count exceeds a fixed threshold in the same sample, the job terminates every running instance of exactly those three named helper processes, nothing else, and writes a log entry describing what it killed. This is intentionally narrow. It does not touch unrelated processes, does not restart services, and does not modify configuration. It exists to relieve the exact pressure that caused the incident described above before the situation reaches total unresponsiveness.

6. Layer three: the watchdog

- A separate, minimal daemon runs with root privileges, independent of the application layer and the scheduler that failed.
- It samples system load every five minutes on a fixed timer.
- Three consecutive samples over a fixed load threshold (roughly fifteen minutes of sustained overload with no recovery in between) triggers an unconditional machine reboot.
- A single spike does not trigger it. Recovery between samples resets the counter.
- A disarm file, placed manually before planned heavy workloads such as large deploys, suppresses the reboot logic entirely until removed.
- Every service on the machine is designed to be stateless on restart: scheduled jobs, the local web application, and the mesh network client all come back automatically after a reboot with no manual steps.

Because recovery after reboot is fully automatic, the watchdog can be aggressive about triggering. The cost of a false positive is a brief, unattended restart. The cost of not triggering is a multi day outage.

7. Testing approach

Before the watchdog was trusted, its trigger logic was exercised directly against synthetic load: a single spike below and at threshold to confirm it does not fire, three sustained breaches to confirm it fires exactly once and does not fire repeatedly, and a disarm file present during a breach to confirm it correctly suppresses the reboot. Each scenario was run in isolation with the results logged before the daemon was allowed to run against production load.

8. Design principles

- Nothing important should live in only one place. Data the operation depends on lived in synced storage independent of the failing machine, which is why the outage did not stop donor outreach or financial reporting.
- Detection and remediation should be layered, from cheapest and narrowest (a counter on a dashboard) to most drastic and broadest (an unconditional reboot), so the system tries the gentlest fix first.
- Any component with authority to take a destructive action, such as killing processes or rebooting a machine, should have that authority scoped as narrowly as the failure that justified it.
- A recovery mechanism is only as trustworthy as the automatic recovery that follows it. A watchdog that reboots into a broken state is worse than no watchdog.
- Provide an explicit, manual override for known heavy workloads, so the safety mechanism does not fight planned operations.

9. Placeholders and scope

This document intentionally omits real hostnames, IP addresses, credentials, API keys, and any donor or personal data. Thresholds shown here are representative of the production configuration's order of magnitude and are not exact operational values. Anyone adapting this pattern should size load and sample thresholds to their own hardware's normal baseline.