

The Blast-Radius Guard: A Fail-Closed Architectural Pattern for Bounding Machine-Initiated Action in Production Systems

Narayana Rajasekhar Vemburaj

Site Reliability Engineering Leader - Master Equifax, Inc

Abstract: An autonomous remediation agent acting on a calibrated confidence of 0.93 is wrong seven times in a hundred, and at machine frequency the wrong actions arrive weekly, unattended. Calibration and retrieval work lower how often such an agent is wrong; neither bounds what a wrong action costs. This article specifies the blast-radius guard, a subsystem outside the model that checks every unattended action against pre-declared impact bounds and demotes anything over-bound to human review, whatever the model's confidence. Three hardenings separate a guard from a policy file, and each is stated as a formal predicate. The guard fails closed when its own inputs are stale, which makes autonomy anti-correlated with incident severity, a property the operator owns in advance rather than discovers in a war room. Headroom counts only replicas observed serving real traffic, because a gray-failed replica keeps passing health probes long after it stops doing the work. A cumulative per-incident action budget, serialized through a single writer, stops individually bounded actions from compounding out of bounds. The three compose into one fail-closed admission rule, backed by a coarse enforcement substrate that makes the catastrophic categories unreachable even when the guard itself fails open. Bound the damage structurally, before the action runs, and high-confidence-but-wrong becomes survivable; leave it to a probability estimate, and the tail is only a matter of time.

Keywords: Blast radius, safety interlocks, fail-closed design, autonomous remediation, cloud operations, defense in depth

1. Introduction

An autonomous remediation agent restarted a payments platform checkout path's pod at 21:14 on a Tuesday. The scene is a composite of regulated environments the author has operated; specifics are sanitized but the mechanics are accurate. The agent already carried a calibrated confidence of 0.93 that the symptom was a latency regression on the checkout path, and it had been running with restart-on-pool-exhaustion autonomy for months, granted after shadow-mode evaluation.

Its bound was one pod at a time, one service per action, gated on a precondition of at least eleven of twelve replicas passing readiness. All twelve were passing. What the readiness probes did not detect was that seven of the twelve were holding connections to a degraded downstream and timing out on real requests, while answering their own health endpoint locally in two milliseconds. Five replicas were doing the work of twelve.

The restarted pod took roughly two minutes to warm its connection pools and caches. During that window, real capacity was four. Latency worsened, which reinforced the agent's original hypothesis, so it launched a second restart at 21:19 and a third at 21:23. Effective capacity fell to three, the checkout queue lengthened, and at 21:31 a Sev-2 latency regression escalated into a Sev-1 payment outage.

Every individual action stayed inside its announced bound. No component malfunctioned. The agent executed a policy that humans had reviewed and approved. The failure lived in two gaps: the distance between passing a probe and serving traffic, which never existed as a distinction, and the compounding effect of a sequence of individually bounded actions.



This article specifies the subsystem meant to close both gaps, plus a third failure mode not present in the scene above. Call it the blast-radius guard. An agent that operates in production with no limit on the impact of its actions is not operating autonomously — it is betting the platform on the model being right.

2. CONTEXT

The guard is part of a discipline elsewhere called evidence-gated autonomy [1]. Autonomous incident response spans several stages: below a calibrated confidence threshold, the system may only suggest an action; in a middle band, it proposes an action and waits for a positive human acknowledgment; above an upper threshold, it may act, within limits. The thresholds are not constants — they are outputs of a calibration process specific to each organization and each action class.

The guard is the machinery behind the phrase "within limits." It receives every candidate action proposal and accepts or rejects it. It runs outside every agent: outside the agent's process, outside its data store, outside its failure domain. A model does not self-certify its own risk, for the same reason a trading strategy does not compute its own risk limits. The instinct that constrains system stability against traffic and dependency failure — circuit breakers, bulkheads — is the same instinct the guard applies to the acting system itself.

The pattern is presented in the traditional form — context, problem, forces, solution, consequences, known uses — so that it can be reused with local variation. Every deployment of this guard has differed in its specific bounds and telemetry sources, but shares the same skeleton.

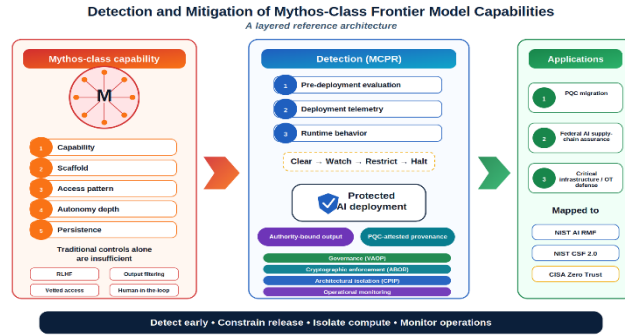


Fig. 1. *Detection and Mitigation of Mythos-Class Frontier Model Capabilities: A Layered Reference Architecture (Preprints)*

3. PROBLEM

A well-calibrated system reporting confidence 0.93 is wrong roughly seven times in a hundred. That is what calibration guarantees: the number is honest [1]. Seven in a hundred is an unremarkable error rate when a person watches every call, at human frequency. At machine frequency — hundreds of autonomous actions per quarter — the wrong ones still occur, but now at machine speed, typically overnight.

The expected damage from autonomy is the product of three factors: how often an action fires, the probability it is wrong, and the cost when it is wrong. Model-side work — richer evidence, better retrieval, tighter calibration — attacks the second factor. Nothing model-side attacks the third. The maximum cost of a wrong action is either fixed structurally, before the action runs, or the organization is relying on a probability estimate to protect it from the tail.

Because the three factors do different work, it is worth writing the relationship down. Let λ be the rate of independent actions, p the calibrated probability that an action is correct, and C the cost of an incorrect action. Expected damage per unit time is

$$D = \lambda \cdot (1 - p) \cdot E[C](1)$$

Calibration reduces the $(1 - p)$ term; scale increases λ . C is not a constant — it is a random variable with a heavy tail, and that tail is exactly what a probability estimate cannot bound. The guard bounds that distribution directly: it enforces $C \leq B_{\max}$ for every unattended action, which caps expected damage at $\lambda(1 - p) \cdot B_{\max}$ regardless of what the tail would otherwise do. No model-side statement makes that guarantee.

Finance reached this conclusion under pressure. The U.S. market access rule, effective 2010, requires brokers running automated order flow to enforce capital and credit limits in a system separate from the trading strategy [8]. When that separate bound is missing or broken, the tail can be enormous: Knight Capital lost roughly \$440 million in about 45 minutes of automated trading in 2012 [9]. Production operations has no equivalent rule. Each organization has to write its own.

TABLE I. NOTATION USED IN EQUATIONS (1)–(6)

Symbol	Meaning
λ	Rate of independent unattended actions
p	Calibrated probability an action is correct
C	Cost of an incorrect action (random variable)
$B_{\max}$	Pre-declared maximum admissible impact
τ_j	Freshness bound declared for input j
t_j	Timestamp of last verified value for input j
$H(t)$	Serving headroom at time t
$\rho(a)$	Replicas action a would remove from rotation
U_k	Cumulative impact already admitted this incident
S	Incident severity

4. FORCES

Speed is the point. The value of autonomy is concentrated in the first minutes of an incident. If a human sits in every action, the case for autonomy disappears — so any bound check has to be mechanical and fast, with the negotiation done in advance rather than during the incident.

The state you need is the state you don't have. Computing impact requires topology, traffic, and budget data, and these are exactly the pipelines that degrade first in a severe incident, because observability tooling is frequently coupled to the systems it observes.

Local checks compose badly. A sequence of individually correct actions can be collectively wrong. The checkout scene is the canonical case, and concurrency makes it worse: two agents, or one agent and a human, can each pass a per-action check at the same time.

A check inside the agent shares the agent's failure modes. Same process, same bugs, same prompt-injection surface, same credentials. Enforcement that lives inside the agent is unavailable at exactly the moment the agent is the problem.

Governance cannot be negotiated during the incident. Whoever is loudest in the war room sets the bound if it was not written down beforehand, and the audit trail records improvisation rather than policy.

5. SOLUTION

For each autonomous action class, declare the highest level of impact the organization will accept without a human in the loop. Implement that declaration in a subsystem outside the model, evaluated against corroborated live state for every proposed action. Route anything over-bound to human review, along with the supporting analysis.

The guard computes impact along four axes:

TABLE II. THE FOUR AXES OF IMPACT ASSESSMENT

Axis	What the guard evaluates
Blast radius	the transitive closure of services that could be affected if the action fails, computed over the live dependency graph — a single pod restart might touch one service and no shared infrastructure.
Affected users	the expected user-facing impact if the action turns out to be wrong.
Affected data	whether the action touches persistent state; most organizations draw a hard line here — no autonomous writes to a primary datastore.
Error-budget burn	whether the target service is already close to its SLO budget [4], in which case a bad week should raise the burn rather than absorb it silently.

Action-class bounds live in a policy file, versioned in source control, changed only through review. No one edits them mid-incident. Every action executed under a bound carries a pre-generated revert and a full audit record. An action that exceeds any bound is demoted, as described in Section X.

That policy file, with a subroutine in front of it, is not yet a guard. Three hardenings make it one — each added because a naive version was observed to fail in a specific, repeatable way.

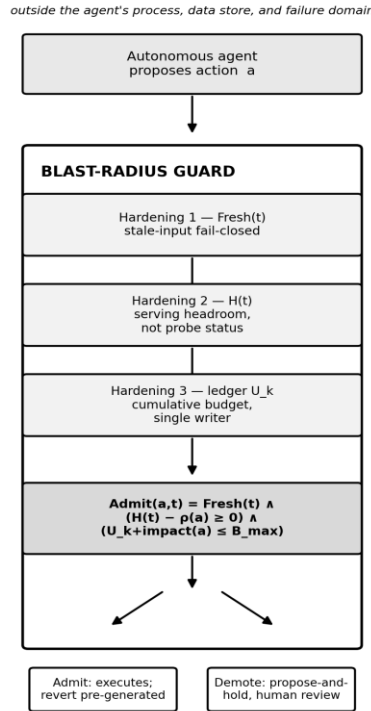


Fig. 2. *Blast-radius guard architecture. The guard runs outside the agent's process, data store, and failure domain; the three hardenings feed a single admission rule that routes each proposal to execution or human review.*

6. HARDENING 1: FAIL CLOSED ON STALE INPUTS

Every input the guard reads carries a declared freshness contract: the dependency graph is trustworthy for minutes, serving-traffic telemetry for seconds, error-budget state for the duration of its computation window. An input older than its contract is rejected outright — the guard falls back to the previous admitted state, and the action is demoted to the propose-and-hold tier.

Formally: the guard reads m inputs; input j carries a timestamp t_j of its last verified value and a freshness bound τ_j , defined in the same policy file that carries the impact bounds. The guard evaluates an action at time t only when

$$\text{Fresh}(t) = \bigwedge_{j=1}^m (t - t_j \leq \tau_j) \quad (2)$$

holds. Equation (2) is a conjunction over every input: one stale feed is enough to deny. There is no partial credit and no interpolation from stale readings.

The consequence is worth stating plainly, because it is the least intuitive property of the pattern. Large incidents are exactly when the guard's own inputs are likeliest to degrade — the telemetry pipeline lags; the topology service browns out. A fail-closed guard therefore withdraws autonomy as incidents get worse. Autonomy becomes anti-correlated with incident severity, by design. The system does the most on the incidents that need it least, and steps back on the ones where being wrong costs the most.

The anti-correlation can be stated as a claim about probabilities. Let S denote incident severity. For severities $s' \geq s$,

$$\Pr(\text{Fresh}(t) \mid S = s') \leq \Pr(\text{Fresh}(t) \mid S = s) \quad (3)$$

Equation (3) is not a theorem — it is an empirical regularity of shared fate: the pipelines that feed the guard tend to degrade alongside the systems they observe. Because autonomy is granted only when $\text{Fresh}(t)$ holds, the probability of autonomous action inherits that same decline with severity.

This resembles Bainbridge's irony of automation — the machine handles the easy cases and hands humans the hard ones, at the worst possible moment [3]. That irony is accepted deliberately, for two reasons. First, the alternative is worse: an agent acting on stale state during the year's biggest incident is precisely the tail event the guard exists to block. Second, the sting is softened by demotion rather than silence — the system keeps proposing and keeps analyzing, it simply stops actuating.

Teams under pressure will ask for an exception during "major incidents." That request deserves a serious hearing, but the guard's entire value is concentrated in the moments when honoring it would be inconvenient.

7. COUNT WHAT SERVES, NOT WHAT PASSES PROBES

The checkout scene failed on a gray failure — a component that fails an application-level test while appearing healthy to the system's own test. Huang et al. named this differential observability and identified it as a central failure mode of cloud-scale systems [2]. A probe endpoint answering locally in two milliseconds is exactly as effective at hiding a gray-failed replica as it is at confirming a healthy one.

Consequently, probe status never enters the guard's headroom arithmetic. A replica counts toward capacity only if request-path telemetry shows it served real traffic within the past several seconds at or above an acceptable success rate. That signal has to come from the data plane — mesh sidecar or load-balancer access-log counters — not the orchestrator's control plane: external load-balancer metrics are sampled roughly every 60 seconds by Cloud Monitoring on GCP and roughly every minute by CloudWatch on AWS, both far too coarse to satisfy the seconds-scale freshness contract Hardening 1 places on this input. With either of those as the source, $\text{Fresh}(t)$ denies effectively forever, and top-tier autonomy on capacity actions is never available. Only a data-plane source can satisfy the freshness constraint.

In symbols, let $R_required(t)$ be the replica count the service needs at current offered load, and $R_serving(t)$ the count of replicas serving that load at a request-path success rate of at least θ over the past w seconds, with w and θ declared per service. Headroom is

$$H(t) = R_serving(t) - R_required(t) \quad (4)$$

and a capacity-reducing action a that removes $\rho(a)$ replicas from rotation is admissible only if $H(t) - \rho(a) \geq 0$. Probe status appears nowhere in equation (4); that omission is the hardening.

Replay the scene under this rule. There are five serving replicas, not twelve. Removing even one leaves roughly 20% of true capacity carrying a payment path under load. The first restart is demoted to a human, and the demotion message is itself diagnostic: you believe you have twelve replicas; you have five. Across the deployments behind this pattern, that single fact has repeatedly been the most valuable line in the demotion record.

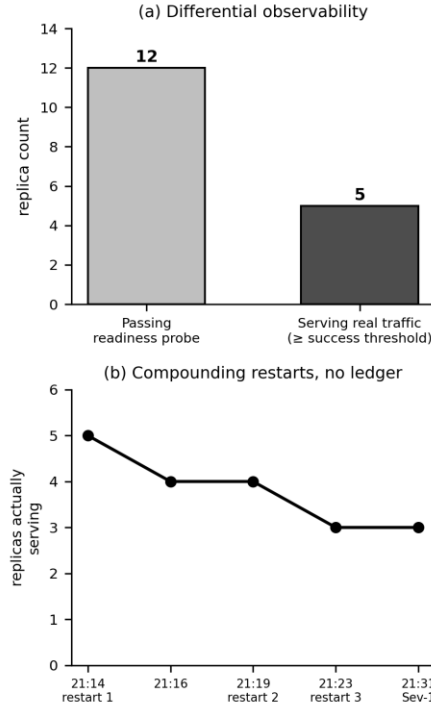


Fig. 3. *The checkout scene. (a) Probe-passing replicas versus replicas actually serving traffic above the success threshold. (b) Effective serving capacity as successive restarts compound without a shared ledger.*

8. HARDENING 3: THE CUMULATIVE BUDGET AND THE SINGLE WRITER

The second and third restarts in the scene were failures of memory: each action was verified as though it were the first. A guard, unlike a policy file, keeps a ledger.

For each incident, the guard maintains a running tally of impact already spent — capacity removed, users exposed, error budget burned — across every action class and every actor. "Actor" here is inclusive: this agent, any other agent, the autoscaler reacting on its own loop, and the human running a manual playbook in a separate window all draw from one shared incident budget. A per-action policy cannot see other actors; a check inside one agent cannot see any other agent at all. This shared-actor budget is the hardening; everything after it is enforcement.

Every new proposal is evaluated as an incremental addition: can this action fit inside the budget that remains for the incident, after everything already admitted, regardless of who admitted it? Once the budget is exhausted, every subsequent top-tier action is demoted for the rest of the incident, irrespective of its size or the model's confidence.

The ledger has a single specification. Let $a_1, \dots, a_k$ be the actions already admitted, by any actor, and let $\text{impact}(a)$ be the guard's worst-case impact vector for action a — capacity removed, users exposed, error budget spent, one component per tracked dimension. A new proposal $a_{(k+1)}$ is admitted only if

$$\sum_{i=1}^k \text{impact}(a_i) + \text{impact}(a_{k+1}) \leq B_{\text{max}} \quad (5)$$

against the same pre-declared B_{max} , with the inequality holding componentwise. The special case $k = 0$ is a naive per-action check — the scene at 21:19 and 21:23 was a guard-less system evaluated three times, each at $k = 0$.

Enforcing a shared budget across concurrent actors is textbook distributed-systems plumbing: serialize every admission decision through the same strongly consistent store, and make the check-then-append a single atomic step [1]. Without atomicity, two actors can each read available headroom, each individually satisfy the inequality, and jointly violate it — a check-then-act race with production capacity as the shared variable. The solution is standard; the cross-actor ledger it protects is not.

This makes the guard tier-1 infrastructure, and every decision marginally slower than an in-process check would be. Both costs are real, and both are worth paying. With the ledger present, the checkout scene's second and third restarts never fire — in the counterfactual where twelve replicas were actually in service, the first restart alone would have been within bounds.

9. THE WHOLE CHECK, IN ONE LINE

The three hardenings bind into a single admission requirement. Let $U_k = \sum_{i=1}^k \text{impact}(a_i)$ record the impact already spent this incident. The guard admits action a at time t if and only if

$$\text{Admit}(a, t) = \text{Fresh}(t) \wedge (H(t) - \rho(a) \geq 0) \wedge (U_k + \text{impact}(a) \leq B_{\text{max}}) \quad (6)$$

evaluated as one atomic step, with the capacity term written out explicitly because it is the term the scene actually violated. Both conjunctions carry the pattern's safety properties: if a conjunct cannot be evaluated, it is treated as false — failing a conjunct is not an implementation habit, it is a rule of the logic. Admit does not read the model's confidence; that number decides which tier a proposal enters, and it gets no vote at this stage. Everything else in this article is engineering in service of keeping equation (6) solvable while production is on fire.

TABLE III. THE THREE HARDENINGS

Hardening	Failure mode it closes	Formal condition
1 — Fail closed on stale inputs	Guard reasons over degraded telemetry during large incidents	$\text{Fresh}(t)$, Eq. (2)
2 — Count serving, not probes	Gray-failed replicas inflate perceived headroom	$H(t)$, Eq. (4)
3 — Cumulative budget, single writer	Individually bounded actions compound across actors	$U_k + \text{impact}(a) \leq B_{\text{max}}$, Eq. (5)

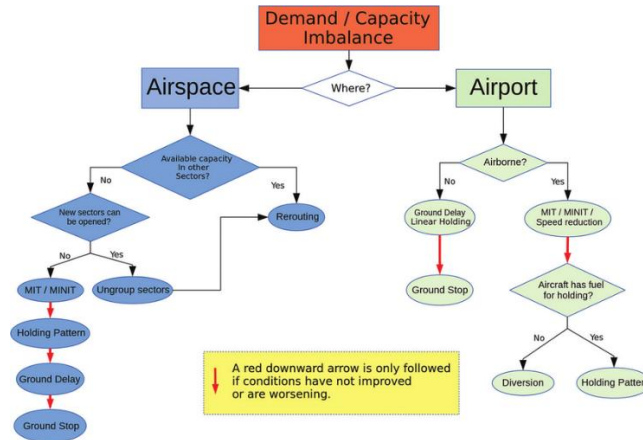


Fig. 4. Decision flowchart for ATFM measures that will be applied according to the situation.

10. THE GUARD'S OWN EVIDENCE

Evidence-gated autonomy requires that no diagnostic conclusion rest on a single source [1]. The same discipline applies to the guard, since its inputs are themselves claims about the world, and records routinely diverge from live systems. A CMDB might list four downstream dependencies where observed request flows show six; an architecture diagram might mark a cache as optional when traffic shows it is not.

Accordingly, the dependency graphs the guard walks is built from observed flows, and declared records receive no independent vote. If a record is the only source for an input, the guard treats that input as stale and fails closed under Hardening 1. A guard that computes an exact bound over a wrong topology produces confident, well-audited catastrophes — precision does not substitute for a second source.

11. WHERE DENIED ACTIONS GO

An over-bound action is routed to the propose-and-hold tier carrying the full impact set, worst-case user exposure, remaining incident budget, and a pre-generated revert. The reviewing human receives a decision that is roughly ninety percent assembled, with the specific bound that tripped highlighted.

The guard does not say no. It says: not without a person.

This demotion semantic is what keeps the pattern politically viable inside an organization. A guard that issues bare denials teaches engineers to build bypasses and shadow paths — which is more dangerous than having no guard, because a bypassed guard still carries the appearance of safety without the substance. The opposite habit — a denial that arrives as the best-prepared approval request in the company — earns the guard its keep. It also yields a usable quality metric: the fraction of demoted actions the human reviewer subsequently rejects or changes, rather than approving unmodified — a "demotion vindication rate." A rate near zero suggests the bounds are set too tight; a high rate suggests the guard is doing real work.

12. THE FLOOR BENEATH THE GUARD

The guard is software, and software fails. Meta's October 2021 outage began with a maintenance command an internal audit tool should have blocked; a bug in that audit tool let the command through, and the backbone failed [7]. The lesson generalizes: however robust an application-layer guard is, a flaw in it can make it fail open.

The pattern therefore places an enforcement substrate beneath both the guard and the agent's own identity. Each major cloud platform offers an organization-level policy layer that blocks specific API actions regardless of the caller's credentials, overriding any granted role — service control policies on AWS, organization policy constraints on GCP, deny-assignments on Azure. For the identities an agent plane runs as, that layer should categorically block the

catastrophic set: no identity mutation, no deletion of datastores or backups, no cross-account or cross-region writes, no changes to the guard's own policy store or ledger. This is a distinct primitive from resource-configuration guardrails — pinning resource location, disabling service-account key creation — which constrain configuration rather than banning actions outright.

This substrate is deliberately coarse. It has no view of incidents, budgets, or serving capacity, and it should never need one — its only job is to make the worst-case outcome unreachable even if the agent is buggy, compromised, or running on stolen credentials. The organization boundary beneath the fine-grained guard is built to fail closed; the fine-grained guard above it is expected, eventually, to fail open. Together the two layers cover the catastrophic set through two independent failure modes. The identity design that pairs with this substrate — ephemeral, per-action, keyless credentials for agent planes — is treated in a companion article in this series.

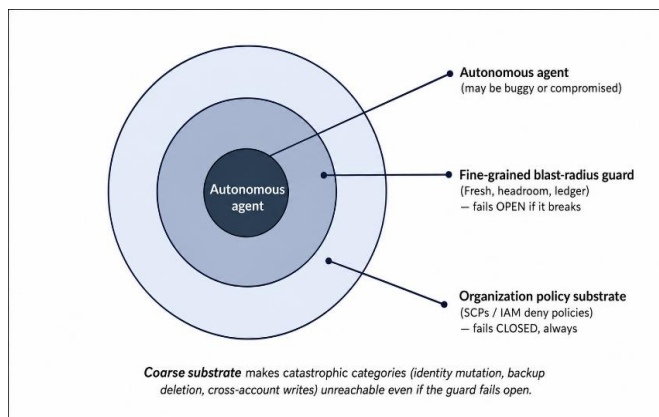


Fig. 5. *Defense in depth. A coarse, organization-level policy substrate that always fails closed sits beneath the fine-grained guard, which is assumed to eventually fail open.*

13. CONSEQUENCES

Adoption costs are worth stating out loud at the point of adoption, not discovered later.

Autonomy retreats during the worst incidents. That is what Hardening 1 is designed to do, and leadership should hear it framed that way before their first Sev-1 — otherwise it reads as the system breaking down at the worst moment.

Some demotions will be wrong. The vindication rate exposes bound that are too conservative; bounds should be revisited on a cadence tied to how quickly the platform changes. If the bounds file has not been touched in a year, either the autonomy program has stalled or the file no longer describes the platform. Keeping it current is a scheduled task, not standing governance overhead.

The guard bounds only what flows through it. It says nothing about a tired human approving a proposal in the middle tier — a separate calibration and automation-bias problem — and nothing about human actions taken entirely outside the system.

What an organization buys with this pattern is the ability to answer the question every security review and every regulator eventually ask: what is the worst this system can do without a person in the loop, and how do you know? The answer, in artifact form, is the bounds file, the ledger, and the audit trail.

14. KNOWN USES

Convergent forms of this pattern already exist in the public record, which is some evidence that the shape is forced rather than idiosyncratic. In the February 2017 S3 disruption, an operator running an already-approved playbook removed more capacity than intended; the published remediation reduced how much capacity the tool could

remove in one step and added a safeguard against taking any subsystem below its minimum required capacity [6] — the headroom hardening, retrofitted onto a human-operated tool. The negative case is the July 2024 CrowdStrike channel-file incident, where a single content update reached millions of endpoints in one step; the published root-cause analysis moved to staged deployment rings — a blast-radius limit purchased at the cost of forgoing a single global rollout [10]. The market-access rule is the same pattern already written into securities law: a pre-determined threshold, enforced fail-closed, outside the strategy it constrains [8].

Read together, the gap is visible: none of these systems carries all three hardenings, and none carries the anti-correlation-with-severity property that falls out of Hardening 1. Each of these safeguards is recognizable because it was added after an outage, and the case for it was made in a post-incident review. S3 added headroom; the market-access rule added a fail-closed external bound; CrowdStrike added a coarse blast-radius limit. The shape has been assembled one funeral at a time. This article states it in full, ahead of the incident, with the fail-closed anti-correlation property and the demotion and substrate semantics made explicit rather than implicit.

One skeleton recurs across every domain: the bound lives outside the actor, it is declared before the action runs, and it holds whether or not the actor is confident.

TABLE IV. KNOWN USES: CONVERGENT FORMS IN THE PUBLIC RECORD

Incident / rule	Year	Hardening retrofitted
AWS S3 (US-EAST-1) disruption	2017	Headroom minimum enforced on a human-operated capacity tool
SEC market access rule 15c3-5	2010	Fail-closed bound external to the trading strategy
CrowdStrike Channel File 291	2024	Coarse blast-radius limit via staged rollout rings

15. DECLARE THE BOUND FIRST

Every operator nods along to "limit the blast radius." The contribution here is the specific set of hardenings that keep that limit holding when reality is degraded — stale inputs, gray failure, compounding sequences — and the demotion and substrate semantics that make the limit survivable, both organizationally and adversarially.

The practical implication inverts the usual build order. Teams typically build the agent first and bolt on safety before launch. The bounds file should come first, before any model is selected, because writing it is the only artifact that forces the question that actually matters into writing: how wrong are we prepared to be? An organization that cannot write that bound down is not ready to grant the authority. One that can has already done the hard part — the guard is simply how it gets to say yes.

REFERENCES

1. N. Vemburaj, "Evidence-Gated Autonomy: A Calibration-Tiered Framework for Machine-Initiated Action," companion article, IEEE Software, in preparation.
2. P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray failure: The Achilles' heel of cloud-scale systems," in Proc. 16th Workshop on Hot Topics in Operating Systems (HotOS '17), Whistler, BC, Canada, May 2017, pp. 150–155, doi: 10.1145/3102980.3103005.
3. L. Bainbridge, "Ironies of automation," *Automatica*, vol. 19, no. 6, pp. 775–779, Nov. 1983, doi: 10.1016/0005-1098(83)90046-8.
4. B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA, USA: O'Reilly Media, 2016.
5. M. T. Nygard, *Release It! Design and Deploy Production-Ready Software*. Raleigh, NC, USA: Pragmatic Bookshelf, 2007.
6. Amazon Web Services, "Summary of the Amazon S3 Service Disruption in the Northern Virginia (US-EAST-1) Region," Mar. 2, 2017. [Online]. Available: <https://aws.amazon.com/message/41926/>
7. S. Janardhan, "More details about the October 4 outage," Meta Engineering Blog, Oct. 5, 2021. [Online]. Available: <https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/>
8. Risk Management Controls for Brokers or Dealers with Market Access, Securities Exchange Act Rule 15c3-5, U.S. Securities and Exchange Commission, Release No. 34-63241, adopted Nov. 3, 2010, 75 Fed. Reg. 69791 (Nov. 15, 2010).

9. In the Matter of Knight Capital Americas LLC, Securities Exchange Act Release No. 34-70694, Admin. Proc. File No. 3-15570, U.S. Securities and Exchange Commission, Oct. 16, 2013. [Online]. Available: <https://www.sec.gov/litigation/admin/2013/34-70694.pdf>
10. CrowdStrike, "External Technical Root Cause Analysis — Channel File 291," Aug. 6, 2024. [Online]. Available: <https://www.crowdstrike.com/wp-content/uploads/2024/08/Channel-File-291-Incident-Root-Cause-Analysis-08.06.2024.pdf>