ON FEEDBACK, VALIDATION AND THE COST OF A BAD RULE

What a platform learns should become what it enforces

Most security learning ends as advice — a report, a recommendation, a ticket someone will action next quarter. The interesting engineering problem is turning it into a rule, safely, in hours.

IN BRIEF

A finding — from an incident, a threat report, a simulation, a tuning session — is worth nothing until something in the estate behaves differently because of it. The gap between the two is usually filled by human transcription, which is slow, lossy, and where the mistakes that cause outages live. This paper treats that gap as a compilation problem: a pipeline from intelligence to enforced rule with validation, canary exposure and rollback as stages rather than as process. It also argues the uncomfortable half — that a system able to push rules quickly is a system able to push a bad rule quickly, and that the safeguards are the product.

1 · The transcription gap

An incident concludes. The write-up names a technique, an indicator, a sequence that should not have been possible. Everyone agrees. Then someone opens a rule editor and types an approximation of what the write-up said, against a query language, from memory, three days later.

That step is where most of the value leaks out. The rule is narrower than the finding — matching the specific host, the specific filename — because the specifics are what the author remembered. It is untested against the estate's actual traffic, so nobody knows whether it will fire ten times or ten thousand. And it carries no link back to the incident that justified it, so in eighteen months a successor will find a rule nobody can explain and will either leave it forever or delete it blind.

The same gap appears everywhere learning happens: a threat feed that becomes a spreadsheet, a red-team report that becomes a slide, a tuning insight that becomes a message in a channel. In each case the platform learned something and **the estate did not change**, or changed by hand and imprecisely.

FIGURE 1 · WHERE LEARNING STOPS



The failure is not that people are careless. It is that the finding and the rule live in different systems, in different languages, separated by a memory.

2 · Treating it as compilation

The useful framing is borrowed from software delivery. A finding is source; an enforced rule is a build artefact; the path between them is a pipeline with stages that can fail. Nothing in that framing is exotic — what is unusual is applying it to detection content, where the industry still largely edits production directly.

Intake. The finding is captured as a structured object — the behaviour, the entities, the evidence subgraph — not as prose. If it came from an incident, the subgraph already exists; the rule is derived from it rather than remembered.

Validation against history. Before it is enabled anywhere, the candidate rule is run backwards over retained events. This answers the question nobody can answer by inspection: how often would this have fired last month, and on what?

Canary. Enabled in observation mode on a bounded slice of the estate — a subset of agents, one environment — where it produces records but enforces nothing. Volume and precision are measured against a real population rather than a sample someone chose.

Compilation and push. The validated rule is compiled into the form the edge executes and distributed. No redeploy, no agent restart, no maintenance window — because a rule that requires one will wait for it.

Attribution and rollback. The deployed rule carries its provenance — the finding, the validation result, the person or process that approved it — and a single reversible identity, so withdrawing it is one action rather than an archaeology exercise.

The stage that changes the economics is **validation against history**, and it is only possible if the platform retained the events in a form a new rule can be run against. A system that keeps alerts but not observations cannot backtest, and a system that cannot backtest is guessing every time it ships a rule. This is the first paper in the series where the event model stops being an architectural preference and starts being the thing that makes an operational practice available.

> **Speed is a safety property, not a convenience.** The argument for hours rather than quarters is not efficiency. A finding that takes a quarter to enforce is a finding an adversary has a quarter to use — and the same window applies to your own misconfiguration, discovered on Monday and still live in April.

3 · The uncomfortable half

A pipeline that can push a rule to every enforcement point in twenty minutes can push a bad one just as fast. This industry has had public, well-documented instances of exactly that, and any paper advocating rapid distribution that does not address it is selling something.

3.1 · Staged exposure is not optional

A rule reaches a percentage of the fleet, then a larger one, with a hold between stages long enough for a problem to become visible. The stages are enforced by the distribution system rather than by the operator's intention — because the case where someone skips them is precisely the urgent case, and urgency is when this goes wrong.

3.2 · The edge must survive a bad rule

Content is data interpreted by the agent, not code executed by it. A malformed or pathological rule should cause the agent to reject the content and report, not to fail — and the previous known-good content must remain resident so rejection degrades to yesterday's protection rather than none. An enforcement point whose failure mode takes the host with it has converted a security control into an availability risk.

3.3 · Rollback is measured, not assumed

Time-to-withdraw is a number an operator should be able to state, and it should be tested on the same schedule as a backup restore. A rollback path exercised for the first time during the incident it exists for is a hypothesis, not a control.

FIGURE 2 · A FINDING, COMPILED

T+0h **incident closed** · the subgraph is the source, not the write-up

T+40m **backtested over 90 days of retained events** · 14 historical matches, 2 needing adjudication

T+3h **canary — 2% of the fleet, observing only** · volume within the predicted band

T+9h **staged to the estate, enforcing** · carries the incident id and one rollback handle

Nine hours, four gates, and a rule whose existence is explainable in two years by someone who was not in the room. The two historical matches that needed adjudication are the point of the backtest: they were found before the rule shipped, not by an analyst at 02:00 afterwards.

4 · What it changes

Detection content acquires a lifecycle. Rules have authors, justifications, measured precision and retirement — the same treatment code gets. A rule nobody can justify becomes a rule that can be removed with confidence rather than fear.

Simulation becomes worth running. Adversary emulation produces findings continuously; without a compilation path they accumulate as a backlog, and the exercise quietly becomes theatre. With one, a simulated finding and a real one enter the same pipeline.

Coverage becomes a measurement. If every rule traces to a finding and every finding to a technique, what the estate does and does not cover is a query — and the gaps are nameable rather than a matter of opinion.

Your content becomes portable. Rules compiled from a stable vocabulary can be exported in a form someone else can execute. A pipeline that only produces vendor-private artefacts has made your learning an asset of the vendor.

5 · Questions worth asking, whoever you are buying from

01 Can I run a candidate rule over the last 90 days of my own events before enabling it?

02 Is staged rollout enforced by the system, or is it a practice someone can skip when it is urgent?

03	What does an agent do with content it cannot parse — reject and report, or fail?
04	What is your measured time-to-withdraw for a rule already on the fleet, and when was it last tested?
05	Does a deployed rule carry the finding that justified it, retrievable by an operator who was not there?
06	Does new content require an agent restart, a redeploy, or a maintenance window?
07	When your simulation finds a gap, what is the shortest honest path from that finding to enforcement?
08	If we leave, in what form do our rules leave with us — and can anything else execute them?

6 · Where this sits in the series

Paper 01 argued for enforcement at the edge; Paper 02 for alerts that carry their own evidence; Paper 03 for legitimacy as a distinct judgement. Each describes one direction of the loop — signals rising, judgement forming. This paper is the return leg, and without it the others describe a very well-instrumented system that never changes its own behaviour.

The loop closes when a conclusion reached centrally becomes an enforcement decision locally, with the evidence intact on both ends. The stages between are not bureaucracy. They are the reason a security team is willing to let the loop close at all.

> *A finding that does not change what the estate enforces is a well-written record of something you already knew.*

EventLake

ABOUT THIS PAPER

Paper 04 in a series on the architecture of active defense, published by **EventLake Research**. Papers 01–03 cover enforcement at the edge, explainable alerts, and the legitimacy judgement. The series is deliberately vendor-neutral, and the questions in §5 are meant to be asked of us as readily as of anyone else.

EventLake compiles findings into enforced content through backtesting against retained events, canary exposure, staged distribution and one-action rollback, with every deployed rule carrying the finding that justified it. **eventlake.com**