ON PROVENANCE, CORRELATION AND EXPLAINABLE ALERTS

An alert is a subgraph, not a row

What changes when every signal a system collects — from a process launch to a badge swipe to a certificate expiry — is the same kind of object.

IN BRIEF

Security platforms are usually assembled from per-domain products, each with its own record shape. Correlation across them becomes a schema-mapping exercise, and because the mapping is lossy, the interesting cases — the ones that cross domains — are the ones that get lost. The alternative is to make one decision early: everything a system observes becomes one canonical event type, entities are nodes, events are edges, and time is a first-class dimension rather than a column. This paper describes what that buys, what it costs, and the three properties that separate a genuine provenance graph from a search index with a graph view.

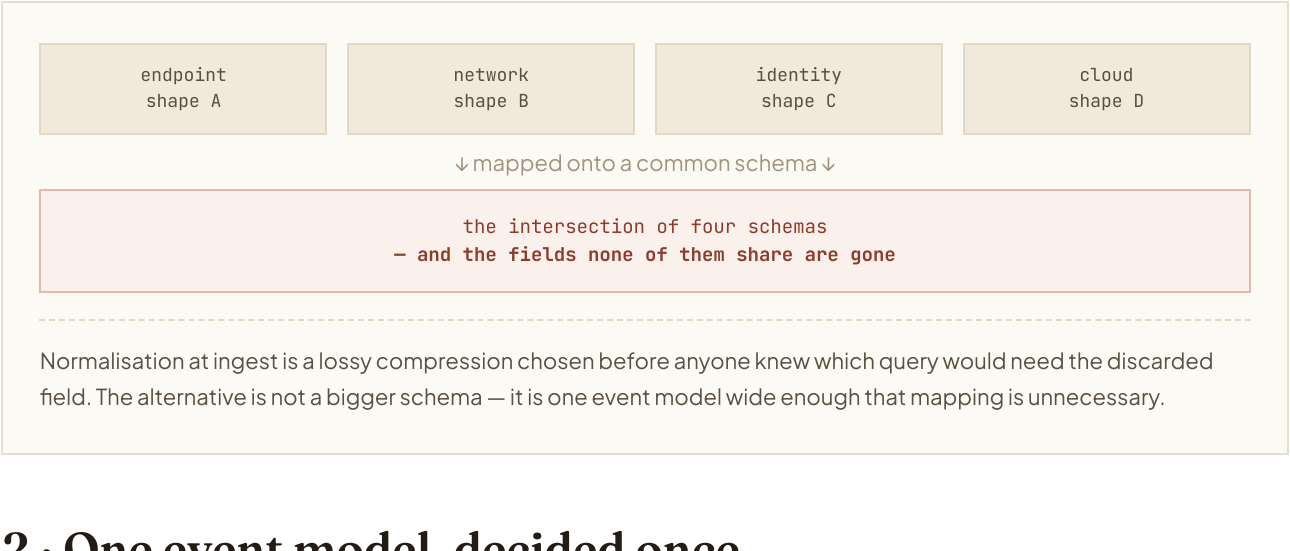
1 · The schema problem nobody chose

A security estate accumulates by acquisition — of products, and of the vendors that made them. Endpoint telemetry has one record shape. Network flow has another. Identity events have a third. Cloud audit logs have a fourth, and it changes when the provider revises their API. Each shape is reasonable for its own domain; none of them was designed to join to the others.

The industry's answer is normalisation: map each source onto a common schema at ingest. This works, in the sense that the fields line up. What it does not do is preserve the parts of a record that the common schema has no column for — and those are frequently the parts that matter. A field dropped in mapping is not recoverable later; the query that needed it simply returns nothing, and nobody is told why.

The result is a specific and predictable blind spot: **single-domain detections work well, and cross-domain detections are unreliable.** An endpoint tool catches endpoint tradecraft. A network tool catches beaconing. Neither catches the sequence where a document macro spawns a process that assumes a cloud role and reads a bucket — because that sequence is three records in three shapes, and joining them was a mapping decision made years earlier by someone optimising for storage.

FIGURE 1 · WHERE THE CROSS-DOMAIN CASE GOES



Normalisation at ingest is a lossy compression chosen before anyone knew which query would need the discarded field. The alternative is not a bigger schema — it is one event model wide enough that mapping is unnecessary.

2 · One event model, decided once

The premise is simple enough to state in a sentence and consequential enough to reorganise a platform around: **everything is an event.** A process launch, a failed sign-in, a firewall drop, a certificate approaching expiry, a badge swipe at a door, a consent grant, a model promotion, an invoice — all of them are the same kind of object, carrying a common envelope and whatever domain-specific attributes they need.

The envelope is what makes the model work. Every event declares a *class* (domain, category, type, activity), a *subject* (the entity it is about), a time, and a severity. Attributes hang off that envelope without being flattened into it. Nothing is discarded to fit, so the question "was this field collected?" has a stable answer and the question "was it judged?" is a separate one.

Collected is not the same as judged, and both are useful. A model that keeps un-alarmed observations as first-class records can answer "what runs automatically on this host" and "who may become root here" — questions no alert stream answers, because the honest answer includes everything nobody flagged.

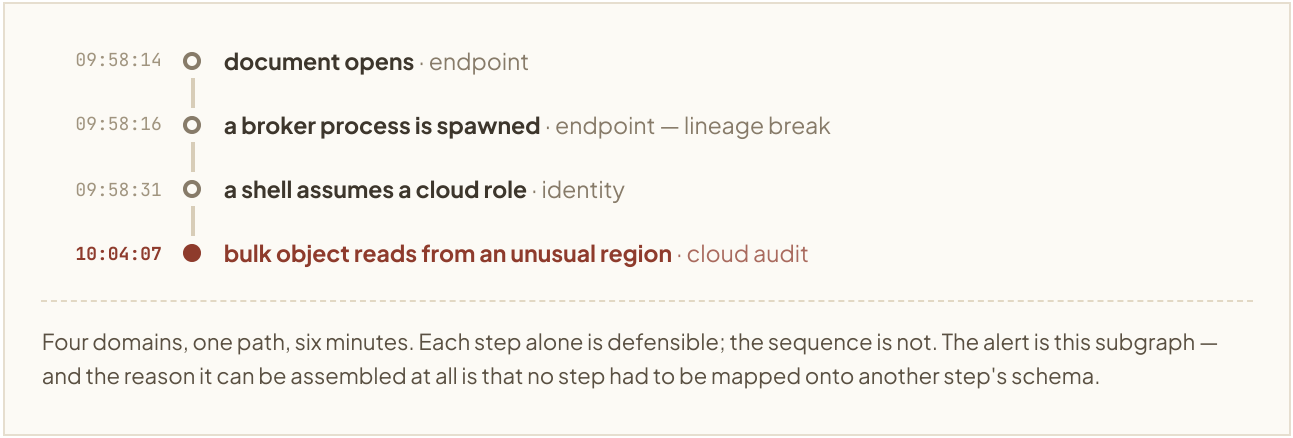
The cost is real and worth naming. One model means the domain experts do not get to define their own record shapes, and each new domain requires someone to decide how it maps onto the shared class vocabulary. That decision is governance work, not engineering work, and it does not go away. What it does is happen *once per domain*, in the open, rather than once per query, invisibly.

3 · Entities are nodes, events are edges

Once every observation shares an envelope, a second structure falls out of it for free. Each event names a subject and usually an object — a user and a host, a process and a file, an identity and a role. Those are nodes; the events between them are edges; and because every edge carries a timestamp, the resulting graph is temporal rather than static.

This is the difference between a graph of *relationships* and a graph of *provenance*. A relationship graph says this user has access to that host. A provenance graph says this process, started by that one, at this time, opened this connection — and the path is walkable in both directions.

FIGURE 2 · A CROSS-DOMAIN PATH, WALKED



Four domains, one path, six minutes. Each step alone is defensible; the sequence is not. The alert is this subgraph — and the reason it can be assembled at all is that no step had to be mapped onto another step's schema.

Two details in that figure carry more weight than they appear to. The **lineage break** at 09:58:16 is the case direct-parent matching misses: a broker process re-parents the child, so the document is no longer the parent of the shell. Walking the ancestry chain finds it; matching immediate parents does not.

And the chain must be joined on **stable identifiers rather than reused ones**. Operating-system process identifiers are recycled; on a long-lived host, a graph joined on them will eventually attach an unrelated subtree to a suspect process. That is the worst available failure in a system whose purpose is attribution, and it is invisible until someone audits a conclusion.

4 · Three properties that separate provenance from a search index

Most platforms can render a graph. Fewer have one. The distinction is not visual, and it shows up under three specific pressures.

4.1 · The alert carries the subgraph, not a reference to it

If an alert stores the specific events that justified it, the explanation is stable: it says the same thing in two years as it did on the day. If it stores a query that will be re-run against a store with a retention window, the explanation decays — and it decays silently, returning fewer events rather than an error. An audit that depends on the second kind is an audit of whatever survived.

4.2 · Time is a dimension of the model, not a filter on it

Correlation is a claim about order and interval, so the model has to hold both. “A then B within five minutes on the same host” is a different assertion from “A and B both occurred today”, and only the first distinguishes an attack from a coincidence. A store that treats timestamps as a sortable column can express the second cheaply and the first only awkwardly, which is why so many detections end up written as the second.

4.3 · The model is honest about the limits of its own observations

Not every timestamp is a measurement. A record produced by periodic sweep knows when it was *noticed*, not when it happened — and the difference is the width of the sweep interval. A model that carries that distinction lets a surface say “ended within the last minute”; a model that flattens it lets a surface print a precise time that is not one, and somebody will build a timeline around it. The same applies to absence: “collected, nothing there” and “never collected” are different facts, and a system that cannot tell them apart cannot be trusted about what it did not find.

> **The unifying idea.** An explainable alert is not a well-written summary of a score. It is a set of specific observations, in order, with the intervals between them, retained alongside the conclusion they support. Everything else is a description of a decision rather than the decision itself.

5 · What this changes downstream

The event model is an internal decision with external consequences, and four of them are worth planning for.

> **New domains stop being integrations.** When a badge reader, a certificate authority and a model registry all emit the same kind of object, adding one is a decoding exercise, not an architectural one — and correlations across it work the day it lands.

Detection content becomes portable. Rules written against a stable vocabulary survive the sources beneath them changing, and they can be exported in a form somebody else can execute. A rule written against a vendor's private record shape cannot leave the vendor.

The audit artefact is a by-product. If the record of what was observed, decided and done is the same object type as everything else, producing evidence is a query rather than a project — and the number in a report can carry a link to the events that produced it.

Field meaning becomes a governed thing. One vocabulary across every domain means "what does this field mean, and who decided that" has an answer worth publishing — and an operator reading an unfamiliar attribute in an alert can find it without asking support.

6 · Questions worth asking, whoever you are buying from

01 When you ingest a source, what happens to the fields your schema has no place for?

02 Show me a detection that spans three domains. Was it written by you, or by a customer?

03 Does an alert store the events that justified it, or a query that will be re-run later?

04 Can I express "A then B within N minutes on the same entity" without writing code?

05 How do you join a process to its ancestors — by operating-system identifier, or by something that is not reused?

06 Which of your timestamps are measurements and which are poll times? How does the interface tell me?

07 Can you distinguish "we looked and found nothing" from "we never looked", on screen?

08 Where is the definition of this event field written down, and who is accountable for changing it?

7 · Why this is a foundation rather than a feature

The first paper in this series argued that closing the gap between detection and action requires the sensor and the enforcement point to be one process, and requires decisions explainable enough that an operator will authorise them. This is where that explainability comes from. An edge cannot be trusted to act unattended on the strength of a score; it can be trusted to act on a traceable chain — and the chain only exists if the model was built to hold it.

Which makes the event model an unusual kind of architectural decision: almost invisible in a product demonstration, and determinative of what the product can ever do. It is made early, it is expensive to revisit, and by the time its absence is felt — during a cross-domain investigation, or an audit, or a migration — it is years too late to change.

> *Everything leaves a trace. Whether you can follow it is a decision somebody made before you bought the product.*

EventLake

ABOUT THIS PAPER

Paper 02 in a series on the architecture of active defense, published by **EventLake Research**. Paper 01, *The gap between knowing and acting*, argues the case for enforcement at the edge and is the companion to this one. The series is deliberately vendor-neutral, and the questions in §6 are meant to be asked of us as readily as of anyone else.

EventLake is built on the model described here: one canonical event type across every domain it observes, a temporal provenance graph as the working memory, and alerts that carry the specific observations that produced them. eventlakes.com