Field notes: the defender's half

The other essays argue about detecting and taxing automated attackers. These are notes on wiring those ideas into one working thing on a fleet of static sites — which order the pieces went in, why each was built to fail safe, and which claims survived being made real.

Matthew J. Harmon

2026-09-11

The other pieces on this site argue about ideas: that an attacking agent runs its loop on your ground and is legible there, that you can make one class of wrong guess expensive, that the flashy control is rarely the real one. These are notes on wiring those ideas into a single working thing across a fleet of static sites — the order the pieces went in, why each one was built to fail safe, and which of the claims survived being made real. The seams are left in on purpose; that is what makes them field notes.

The through-line, stated once so the rest reads as variations on it: the attacker is doing search and the defender is doing judgement, so every component here is shaped by the fact that *the defender's mistakes are the expensive ones*. That is the argument of *The asymmetry does not survive contact*, turned into wiring.

Start by refusing what you are not

The sites are static. They run no code, hold no login, and have no database to reach. So the first move was the cheapest one: make the server answer only the two methods a static site has any use for, `GET` and `HEAD`, and refuse the rest with a `405`.

That turned out to matter more than expected, because there had been no method check at all. Every verb — `POST`, `PUT`, `DELETE`, `TRACE`, an invented one — was being answered exactly like a `GET`, handing back the page. Treating every verb as a `GET` bought nothing and left needless method confusion and proxy- and cache-behaviour surface. Refusing the rest removed it and, as a side effect, drew the first legible line: on a site with no forms, a `POST` has no honest reason to exist, so the refusal is also a signal.

The check that reassures you here is the boring one: confirm that nothing you actually serve depends on the surface you just removed. No page posted anywhere; no cross-origin headers were set, so nothing could be relying on a preflight. The refusal was safe precisely because the sites are dull in a way you can enumerate.

You cannot act on what you cannot see

The moment you want to *do* something about a client — log it, rate-limit it, ban it — you need to know who it is, and that is where the architecture bit back. The sites sit behind a connection-level proxy that routes by name without terminating anything, so every request reached the server from the proxy's own address. One internal address, identical for the whole internet. You cannot ban that; you would ban everyone.

Recovering the real address meant having the proxy prepend it to each connection and having the server trust that prefix — but only from the proxy, because a prefix anyone can send is a prefix anyone can forge. The interesting part was *how* it went in: in three steps, each reversible on its own, so that no single change could take the fleet down. First the server learned to read the address *optionally* — present or absent, it behaved the same. Then the trust was switched on while the proxy still sent nothing. Only then did the proxy start sending it. At no point did both sides have to move together, which is the whole reason it was safe to do to dozens of live sites at once.

That address is worth keeping straight, because it is tempting to overstate what it proves and forget what it doesn't:

- A completed handshake **does** vouch for the address in a way a header never can — the peer had to receive and answer real packets, so it cannot be forged by the blind spoofing that fakes a client-supplied field.
- It **does not** tell you who is behind it. A datacenter egress, a VPN, a relay, a carrier NAT — any of these can sit in front, and a shared gateway can carry people who did nothing. The address is the network source that spoke to you, which is what a firewall rule acts on. It is not an identity.

And a smaller lesson that cost a real bug to notice: the log line that decides who gets banned is written partly from the request path, which the attacker controls. A path with a newline in it can forge a second log line naming any address it likes. Escape the field — a newline as two harmless characters, a quote that cannot close its own field — or you have built a pipeline where hostile text writes your firewall rules. It was worth testing directly: a request whose path forged a complete fake entry nominating a bystander never got the bystander banned.

Make the wrong guess expensive, then ban

With a real address in hand, the trap from *A bomb is not a control* becomes actionable. A request for a login page, a dotfile secret, a package-manager tree or an admin panel is, on these sites, a scanner reading a wordlist — little else does. Rather than hand it a `404`, the server hands it a decoy: a small file, served as an archive, that decompresses to something enormous. The naive scanners that unpack it pay for the guess in memory.

The decoy is the cheap, theatrical half: a careful client reads the declared size and never inflates it. The control is the ban: every probe is logged with the real address and a few within a few minutes trip a temporary, port-scoped firewall rule. The ban does not care whether the bait was taken; it fires on the request. It is the half that actually shortens the engagement, because it stops the rest of the wordlist from running.

A design doc would not have predicted either of these. Before you assume you know what a

path means, look at what is already in your own tree: on one site the scanner-bait path *already existed* — someone had left a decoy there months earlier — so a blanket rule would have banned visitors to a working trap. And the pattern list is a standing liability you have to own — a rule for `/api` is safe only until the day you stand up a real one, and the exception has to be carved before that ships, not after the first complaint.

The defender's half is the hard half

Everything above is plumbing. The judgement — *is this thing hostile, and what is it* — is where the asymmetry actually lives, because the attacker's wrong guess is free and the defender's is not. So the piece that does the judging, a small agent that reads the trap telemetry and characterises each client, was built around that constraint rather than against it.

Its tools are read-only by construction. It can read the logs and query the ban state; it cannot ban, unban, or change anything. That boundary is enforced by the tool surface, not by an instruction a model could decide to ignore. A wrong classification therefore has no irreversible consequence — which is the only basis on which it is safe to let a model near the loop at all.

It reports two confidences, never one. How sure the activity is hostile — the *event* — and how sure the address is a single actor — the *attribution*. Against boring static sites the first is often high; the second is usually lower, for all the reasons the address is not an identity. Keeping them apart stops a strong event judgement from dragging attribution up with it.

The calibration is enforced, not requested. The model kept rating a cloud scanner's attribution as *high*. Asking it, in the prompt, to be more careful helped only some of the time. So the cap moved into code: the attribution ceiling is computed from the address's own network origin, and the model's answer is clamped to it regardless of what it proposed. The prompt still explains the rule — but the rule holds because a function enforces it, the same way the read-only boundary holds because the tools do. Anywhere a model's discipline matters, prefer an invariant you can enforce over an instruction you can only hope it follows.

Keep it on your own ground

Where the judging happens turned out to matter more than which model does it.

The models run on the operator's own hardware. A defensive analyser is reading your logs and your captured attacker artefacts; sending that to a third party to score means announcing both your defences and your incidents to someone else. Running it locally keeps the whole loop on ground you control — which is the same instinct as everything else here.

The other decision was about stability. In this deployment a single small model was not a stable judge — asked twice, it could answer differently. The fix was not a bigger model but *more* of them: gather the evidence about a client once — that part is deterministic and cheap — then fan only the *judgement* out to a handful of independent models and take a weighted vote. One big model anchors it; several light models, each running locally on a different node, each return an opinion.

That shape earns its keep. Watching a live cloud scanner, the individual light models disagreed with each other and varied from run to run — and yet three consecutive votes returned the same verdict, and every model agreed the event was hostile even while they argued about the label.

In those runs the vote held even while its members did not, and the one thing that matters for a defender — *is this worth acting on* — is the thing they agree on. Where they disagree, the agreement count says so, and low agreement is a flag for a human rather than noise to average away.

What it deliberately does not do

The limits, with nothing rounded off:

- It only sees requests that reach the trap. Names fronted by other layers are answered before the server sees them; a defence is only as wide as the traffic that reaches the thing enforcing it.
- The fine classification is still noisy under the hood. The vote stabilises the verdict and the event judgement; it does not make a 3-billion-parameter model a reliable taxonomist. Trust the event and the agreement count first.
- Every irreversible action stays narrow: the ban is port-scoped so it cannot lock the operator out, and short in time because the address may be shared. The agent that does the reasoning holds no such lever at all.

What survived deployment

None of this makes search expensive; that was never on offer. It makes one narrow class of wrong guess expensive, recovers enough identity to act on it, keeps every action reversible and every judgement calibrated, and runs the whole thing on hardware the operator owns. The machine does the reading — the tireless, million-line-wordlist reading that a person cannot keep up with — and a person keeps the last word on anything that cannot be taken back.

The individual tricks are new. The posture is the oldest one there is: scope the blast radius, enforce the invariant instead of trusting the actor, and never build a control whose failure you cannot undo. Automation did not change that. It just raised the volume until doing it by hand stopped being an option.