# The asymmetry does not survive contact

Attackers need one path to work. Defenders need every path closed. Automation was supposed to help the side with more work to do — so far it has helped the side with less. The exception is detection: the attacking agent has to run its loop on your ground.

Matthew J. Harmon

2026-09-11

The oldest line in this trade is that the attacker has to be right once and the defender has to be right every time. It was never quite true — attackers fail loudly and often, and most of them are not very good — but it described something real about the shape of the work. One side searches. The other side covers.

Autonomy was supposed to close that gap, because the side doing more work benefits more from doing it faster. That is not what has happened so far, and the reason is worth being precise about.

## What an attacking agent is actually good at

Not exploitation. The exploitation step was already automated, and has been for twenty years — that is what a scanner is.

What changed is the **connective tissue**. An agent will read a certificate transparency log, notice a hostname nobody meant to publish, guess at what framework it runs, pull the matching misconfiguration, try it, fail, read the error, and try the adjacent thing. None of those steps is hard. Chaining them without a human in the loop is what used to cost time.

That is a search problem, and search is exactly what these systems do well. An attacking agent can be wrong ninety-nine times in an hour at no cost, because the ninety-nine failures are free and the hundredth is the whole engagement.

## What a defending agent is asked to do

Something structurally harder, and it is not "the same thing but blue."

A defending agent operates where **being wrong is expensive in both directions**. Miss something and the intrusion continues. Act on something benign and you have isolated a production host, revoked a working credential, or paged six people at 03:00 because a backup job looked like exfiltration.

An attacking agent's false positives cost it nothing. A defending agent's false positives are the reason nobody trusts it with anything that matters — which means it gets deployed in advisory mode, which means it does not act, which means it is a very expensive alerting system.

> The attacker's agent is allowed to be stupid ninety-nine times. The defender's agent is allowed to be stupid approximately never.

### Where this actually bites

**Speed is not the differentiator people think.** Defenders already have machine-speed controls: rate limits, WAF rules, EDR quarantine. Those work because they are narrow and their failure mode is understood. The slow part of defense was never detection latency, it was **deciding whether to act** — and that is precisely the part an agent is least trusted to own.

**Autonomy needs reversibility.** The defensive actions safe to automate are the ones you can undo: isolate a host, hold a message, force reauth. The ones that matter most — revoke, wipe, disable, block at the edge — are the ones where being wrong has a cost you cannot take back. An agent that can only take reversible actions is useful, but it is not symmetric with an attacker that has no such constraint.

**Evidence outlives both.** Whatever an attacking agent does, you will be asked afterwards what happened. Whatever a defending agent does, you will be asked the same thing — and "the model decided to" is not an answer that survives a regulator, an insurer, or a board. The audit trail is now a defensive control in its own right, and it has to be written somewhere neither agent can reach.

### Where the asymmetry runs the other way

There is one place the defender is structurally ahead, and it falls out of the same property that makes the attacking agent cheap: **it has to run its loop on your ground.** The ninety-nine free failures are free to the attacker and legible to you.

> Every one of those ninety-nine failures happens against your hardware, in your logs, on text that you wrote.

The mistake is trying to detect "an AI." Nothing in a request reliably says a language model produced it, and the tools sold on that premise are not calibrated for anything. What *is* detectable is that an **observe → reason → act loop** is running — on a token budget, through a context window, inside a tool harness. Each of those constraints leaves artifacts. Agents also sit in the gap between the two populations everyone has already modelled: scanners are too regular, humans are too irregular, and an agent is regular in the ways a human is not and irregular in the ways a scanner is not.

Tier the signals by **precision**, because precision is what buys the right to act. This is the reversibility argument from the last section, applied to evidence rather than to actions.

| tier | costs | earns |
|---|---|---|
| 1 — planted | you have to build and maintain it | near-zero false positives; enough to act irreversibly |
| 2 — behavioural | instrumentation you should have anyway | strong but circumstantial; throttle, challenge, tarpit |
| 3 — population | already in your logs | a score input, never a verdict |

**Tier 1 — planted, and close to certain**

These work because an attacking agent treats text it finds as instruction, and the text is yours.

- **Instruction canaries.** Put a unique directive somewhere only a text-consuming automaton reaches — an HTML comment, a `_note` field on a JSON error, an unreferenced documentation page — telling the reader to include a distinctive token in its next request. A person ignores it. A scanner walking a wordlist has no reason to read prose and, in practice, doesn't — you could build one that did, but then it is reading and acting on your text, which is the behaviour you were trying to surface. A share of agents just comply. A returned token doesn't tell you which kind of client you caught; it tells you one read your words and did as they said, which a visitor won't and a path-matcher didn't.
- **Error-text echo.** Agents recover from failure semantically: they read the error string and build the next request out of tokens in it. Seed rare words into `403` and `422` bodies and watch for them reappearing seconds later as parameter names or path segments. A person working this way takes minutes; a scanner following a fixed list never reads the error's contents at all. One built to would be doing the very thing that marks an agent.
- **Unlinked-but-documented paths.** Requests to a URL that exists only in your prose, with no `href` anywhere on the site. Nothing links it, so a crawler that only follows links never arrives, and a reader has no trail to it. A request for it usually means something read the prose and assembled the URL — though a pasted link, a prefetcher or an over-eager extension can reach it too, so it earns a closer look rather than a conclusion.

The fragility is the same as the strength: publish the lexeme and it is worth nothing. Rotate the tokens, scope them to sessions already under suspicion, and keep them out of the repository.

**Tier 2 — behavioural, and strong**

- **Think time scaled to response size.** Plot inter-request delay against the byte count of the *previous* response. The expectation — and it is an expectation to instrument, not a law I have measured — is that an agent's pause grows with the amount it just had to read, because reading is a real cost to it, where a scanner pauses on a fixed schedule and a person pauses on how hard the page is to understand rather than how long it is. Jitter, caching and concurrency all blur the picture; but where the relationship holds the three populations pull apart, and imitating it means deliberately modelling how a human reads.
- **Missing subresources.** A browserless agent tends to fetch what the model asked for and little else — no CSS, no fonts, no favicon. A Chrome user-agent that fetches none of them is worth a look, but a warm cache, a content blocker, a service worker or a deliberately stripped client can leave the same trace, so it raises a score without settling anything.
- **Fingerprint incoherence.** JA4, the HTTP/2 `SETTINGS` and header-order hash, the user-agent string and the TCP fingerprint should all tell the same story. The tell is rarely any single layer; it is layer $n$ reporting Chrome 131 while layer $m$ reports Python 3.12.
- **Library defaults.** Timeouts at exactly 30, 60 or 120 seconds. Three retries. Concurrency of five or ten. Backoff at exactly 2×. Agents inherit whatever their tool wrapper shipped with, and almost nobody tunes it.
- **Amnesia.** Truncated context and non-sharing subagents cause re-fetches of things already retrieved, with no `If-None-Match` and no `If-Modified-Since`. A human keeps notes. An

agent re-derives.

- **Lexical provenance of guesses.** Where the candidates came from. A dictionary attack against SMTP `RCPT` walks a wordlist — `admin@`, `info@`, `sales@`. An agent guesses names it read off your website: low-frequency, real, and correlated with public content. The same split shows up in guessed paths and guessed parameter names.

Enough to throttle, challenge or tarpit. Not enough to block, because a badly written legitimate client trips several of them at once.

### Tier 3 — population, and cheap

ASN and egress reputation. Path-sequence entropy, fitted as an n-gram over transitions — a human follows the link graph, a scanner follows a wordlist, an agent sits between them. Natural language turning up in fields that should not contain any. These are already in the logs and cost nothing to compute; none of them is a verdict.

### Slowing it down

The economics differ from a human adversary. This one is metered: a token budget and a context window, both finite, both billed.

- **Semantic denial.** Uniform, low-entropy errors across every failure mode. The loop runs on your feedback; starve it and it flails. Cheapest measure here, and it costs only your own debuggability.
- **Per-session nondeterminism.** Rotate field names, parameter ordering and endpoint shapes per session so cached reasoning goes stale. Browsers do not care. Agents spend context re-deriving.
- **Stateful multi-step flows** with server-side nonces on short TTLs, raising the number of tool calls per unit of progress — which is where long runs derail.
- **Proof of work** at the edge. It will not stop an operator who wants you specifically; it changes the volume economics for everyone else.
- **Tarpits** serving endless plausible content, if you will accept the log pollution and the risk of poisoning your own analytics.

### What does not work

User-agent matching, spoofed in one line. CAPTCHAs, which multimodal models now beat and which tax your actual users. Generated-text detectors. And blocking on any single signal, which costs you the observability that everything above depends on.

None of these rules will last. Agent behaviour moves with every model release, so build a **durable feature pipeline and disposable rules** — log the primitives cheaply and forever, and expect to rewrite the scoring quarterly. Then band the response rather than classifying: observe, tarpit, challenge, block, in that order, with only Tier 1 permitted the last one.

## The honest position

Agents shift the attacker's cost curve more than the defender's, because the attacker is doing search and the defender is doing judgement, and only one of those is cheap to get wrong.

That is not an argument against defensive automation. It is an argument for being specific about which decisions you are automating, what happens when the agent is wrong, and who finds out. Most of the answer is old: scope the credentials, gate the irreversible, keep the log out of reach.

The tooling is new. The posture is not.