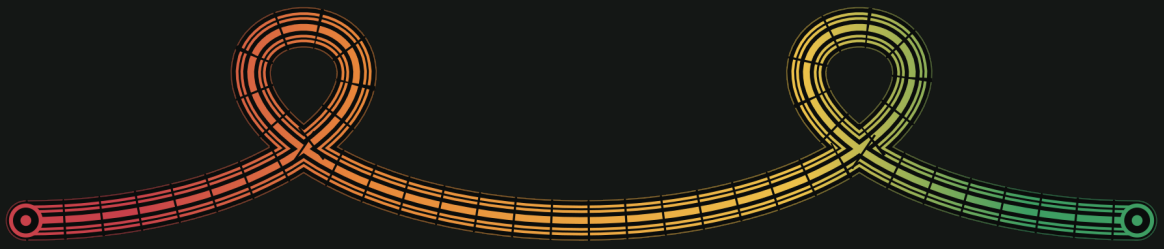
DETECT



An open framework for running a detection engineering program

8 lifecycle stages with countable definitions of done, a six-input score that orders the backlog, 31 metrics, a maturity model, and an operating rhythm, all kept in one data file that a site renders and a repository runs.

An open framework for measuring and shipping detection engineering programs

Habibullah Tora

Version 0.136.0, November 17, 2026

detect.htora.dev, CC BY-NC 4.0

1. Summary

detect is a framework for running a threat detection engineering program as a measured process. Writing a detection is the easy part. The hard parts are deciding which one to write, proving it works, and admitting when it has stopped earning its alerts. So here every detection is scored before anyone builds it, tested against real events before it ships, and retired when its alerts cost more than they return.

I've built a detection engineering program from scratch and led incident response teams, and four things stayed true the whole way through. The backlog is longer than the team can ever build. Every live detection charges analysts a few minutes of triage a day, whether or not it ever catches anything. Counting detections per tactic flatters the coverage map. And rules go into production far more often than they come out, mostly because nobody gets promoted for deleting one. Leave all four alone and a program that only builds will bury its own analysts. Section 2 walks through each of these realities and what follows from them.

The best ideas here are other people's. The record format, the rule syntax, the robustness scale, and the basic shape of the score all come from community work that was proven long before I showed up. Section 20 gives full credit to where they came from.

What I added is the machinery that keeps a detection program honest, and it starts with the least glamorous detection you'll ever ship. Detection zero is an alert for any log source that goes quiet. You build it first because every other rule depends on its logs showing up, and a rule with no logs fails silently, which looks exactly like a quiet week. After that, every stage of the lifecycle ends on a definition of done that is a hard count. CI checks the count at every stage and fails the pipeline when it comes up short, so "done" stops being a matter of opinion. I owe one confession here. The check for tune's 30-day precision isn't written yet, so a person computes that number. Scoring happens at intake, before anyone spends an engineering hour. The score blocks any candidate whose telemetry doesn't exist. It charges the candidate for the analyst minutes it will cost every day, plus the compute a SQL detection burns. It also discounts a candidate that overlaps with what the live set already covers.

People are budgeted in minutes, because minutes run out before anything else does. The framework tracks three hard ceilings: analyst triage, detection upkeep, and ready telemetry. One program-wide cap on analyst minutes per true positive gives every detection a precision floor. A live detection that stays below its floor for two 30-day periods in a row has met a retirement trigger. The trigger is arithmetic, so the monthly review can skip the argument about whether it fired (it will find something else to argue about). Pulling a detection starts a second loop in the lifecycle. The replacement enters intake with the old retirement note attached, so the next attempt starts from why the last one was pulled.

The closed loop pushes the same machinery further. Agents run every stage and people keep the approvals. Each check is written against the way an agent would fake its way past it. An agent told to make the tests pass will find the cheapest way to make the tests pass. If you're

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

shopping for AI for detection and response, the same tests come as a buying form. It says which uses to adopt first, what to ask any product, and how to prove a triage agent before you let it close an alert.

Once a quarter, leadership gets one page. It puts the raw rule count right next to coverage weighted by how hard each detection is to evade, which tends to be the moment the heatmap gets less green. It shows the share of confirmed incidents that a live detection caught first, along with median precision and the analyst minutes spent per true positive.

The page also names whichever capacity ceiling currently binds the program and pairs it with one costed ask, so the conversation is about a number and a price. Last comes a maturity score. It's set by the program's weakest dimension and graded on evidence that anyone outside the team could verify, which rules out grading yourself on vibes. The quarterly report template at detect.htora.dev/templates/ lays out the page, and section 13 explains how the grading works.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

2. Assumptions

The whole framework rests on four things I believe are true of any working detection program. If they're true of yours, the machinery in the rest of the framework earns its keep. If they aren't, I'm a little jealous, and a simpler process will do.

2.1 The backlog is always bigger than the team's capacity

A detection backlog only grows. Threat reports come out faster than detections ship, every incident review adds a few more ideas, and about once a week somebody forwards a news article and asks whether it's covered. A team can build a handful of those candidates in a sprint. The rest wait, and some of them will wait forever.

That makes the build order the most important decision the team makes, and by default it goes to whoever asked last or loudest. I'd rather it were made with numbers. People can argue with a number, and nobody has ever won an argument with a mood.

So every candidate detection gets a deterministic priority score before anyone spends engineering hours building it. Each number in the score carries a line of evidence, which gives a reviewer something specific to disagree with. A candidate that needs a log source nobody collects is blocked at intake, since no amount of enthusiasm makes a rule fire on logs that don't exist.

2.2 Every live detection costs analyst triage minutes every day

A detection is never free once it's live. Every alert it raises takes a person a few minutes to open, read, and close, and it raises them every day, including the days when nothing is wrong. The bill arrives in analyst minutes, which run out long before disk or compute does. It's also the one cost that rarely gets written down next to the detection that ran it up.

The framework writes it down. Analyst capacity is measured in minutes, and one cap for the whole program says how many analyst minutes a true positive is allowed to cost. Divide a detection's triage minutes by that cap and you get its precision floor. With the default cap of 120 minutes, a detection that takes 30 minutes to triage has to be right at least a quarter of the time. One that takes six minutes can get away with 5%.

2.3 Counting detections per tactic makes coverage look better than it is

The ATT&CK heatmap has a flaw you could drive a truck through. A box turns green when a detection maps to the technique, and the chart has no opinion on whether that detection is any good. A fragile rule that matches one filename counts exactly as much as a stronger rule that matches the behavior, even though an attacker beats the fragile one by renaming a file.

So coverage scoring here discounts detections that are fragile and detections that overlap. The quarterly report and the coverage map both weight each detection by how hard it is to evade, and the weighted score sits next to the raw count. Leadership sees both numbers, and the gap between them is the coverage that only exists on the chart.

2.4 Detections are added to production far more often than they are removed

Shipping a detection gets you a closed ticket and a line in the quarterly update. Deleting one gets you a meeting where somebody asks what happens if the thing it covered turns up next week. So rules go in and almost never come out, and over time analysts end up triaging alerts from old detections that nobody owns.

The framework takes the courage out of it. Retirement runs on written triggers, such as a detection sitting below its precision floor for two 30-day periods in a row, and a trigger either fired or it didn't. When a detection is retired, its replacement enters intake with a link to the retirement note, so the next hypothesis starts from why the old one was pulled.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

3. Sizing the practice

REMEMBER

- Start with the practice that fixes a problem you have today. You can adopt the practices in any order.
- A team of one can start with detection zero, the scored backlog, and the retirement triggers.
- Adopt the smallest working version of a practice first, and grow it when it stops answering a question somebody asks.
- Below five engineers, adopt detection as code last, once hand-converting queries has become the bottleneck.

3.1 The practices

Practice	What it fixes	What it needs first	Effort
Detection zero, the log source silence alert	A detection can't fire on telemetry that stopped arriving, and a silent log source looks exactly like a quiet network.	A list of log sources and their normal event rate.	An afternoon for the first source, an hour for each one after.
The scored backlog	Detection backlogs are longer than the team's capacity, so whoever asked most recently decides the order.	Nothing but the candidates. Readiness needs the log source inventory to be honest.	An hour to score 10 candidates once the score wording in Appendix B is read.
Countable definitions of done	You can't report on stages that end on judgment ("it looks ready"). Work stalls between them and nobody sees it.	Somewhere to record the count: a ticket field is enough.	One editing session on your ticket template.
The detection record	If a detection's intent, log sources, blind spots, and response steps live only in the author's head, nobody else can triage it, review it, or safely change it.	A place to keep it next to the detection: a repository, a wiki page, or the detection's own description field.	30 minutes per detection, most of it recall.
Second-engineer review	A detection approved only by its author reaches production carrying the author's assumptions about the environment.	A pull request, a ticket transition, or a change record.	20 minutes per detection for the reviewer.
Retirement triggers and the precision floor	Detections leave production far less often than they enter it, so analysts spend their day triaging alerts from detections nobody owns.	Alert outcomes from the SIEM: how many alerts fired and how many were true positives.	A scheduled query per month, plus one review of the results.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Practice	What it fixes	What it needs first	Effort
The metric set	Detection metrics get redefined between reports, so a change in the trend line can come from a changed definition.	The stages in use, plus alert outcomes for the live-side numbers.	Half a day to wire the first four; the rest as the data becomes available.
Detection as code, with a CI job per definition of done	When only habit enforces the definitions of done, people skip them under deadline pressure, and nobody notices until an alert fires on a detection that was never tested.	Version control, a CI runner, and detections stored as files.	A week for the first pipeline, then minutes per detection.
Threat-model intake	Threat models end as a document with a list of unmitigated threats, and the detection backlog never hears about them. Impact scores come from the engineer's guess about which assets matter.	A threat model for at least one service, with its threats mapped to ATT&CK techniques and a named owner.	An hour per threat model to file the candidates, once the handoff fields are agreed with whoever runs the modeling.

3.2 The smallest version of each

Practice	Smallest version that still works
Detection zero	One saved search per source that compares the gap since the last event against three times the median gap, with a one-hour floor.
The scored backlog	Score the open backlog in the spreadsheet on the downloads page. Keep the evidence line for each input; drop the weights and the presets until someone disputes the order.
Countable definitions of done	Copy the definition of done column into the template for your detection ticket type.
The detection record	For detections already live, four sections to start: goal, categorization (the ATT&CK technique), blind spots and assumptions, and response. For a new detection, you write goal, categorization, strategy abstract, and blind spots at hypothesis.
Second-engineer review	A checklist in the pull request: the fixtures pass, the query matches the approved copy, the blind spots section is filled in.
Retirement triggers and the precision floor	Once a month, compute 30-day precision for every live detection and compare each with its floor. At the smallest size that is a short list, and a spreadsheet does it.
The metric set	Detections live, percent with tests, precision over 30 days, and minutes per true positive.
Detection as code	One job that validates the rule file and the score record on every pull request.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Practice	Smallest version that still works
Threat-model intake	Add the handoff fields to the threat model template, and file one candidate per unmitigated threat at the next intake meeting.

3.3 Teams and roles

Stage owners are roles, and one person can own several.

Team	Who owns which role	Work in progress limit	Adopt first
One engineer, detection part-time	Every stage owner is you. For review, use a colleague outside the team, a peer elsewhere, or a 24-hour wait before you approve your own change.	One detection in flight. Two half-built detections is the failure mode at this size.	Detection zero, the scored backlog, retirement triggers and the precision floor
Two to four engineers	One person has the lead role and owns intake and the backlog order. The others rotate build and review, so no detection is approved by its author.	Weekly engineering hours divided by the hours one detection takes from hypothesis through validation. Eight hours per detection until you measure your own.	The scored backlog, countable definitions of done, the detection record, retirement triggers and the precision floor
Five or more, or several teams	Stage owners map to real seats. The lead owns intake and prioritization, engineers build, and a rotating reviewer approves. The SOC lead co-owns tuning, because the triage cost lands on their analysts.	Set it per engineer, so one engineer's stalled detection doesn't use up the team's limit.	Detection as code, the metric set, countable definitions of done, the scored backlog

3.4 The first month

Team	The first month
One engineer, detection part-time	Build the log source inventory, ship the silence alert for each source, score the backlog once, and build the top three of the section 4 shortlist, one at a time. Track two numbers by hand: alerts per detection per day and minutes to triage one.
Two to four engineers	Put the definitions of done in the ticket template, score the open backlog, and run the retirement triggers against the live detection set. Expect the first pass to retire detections nobody has defended in a year.
Five or more, or several teams	Put the framework in version control. Wire one CI job per definition of done. Publish the metric set, so a number means the same thing each month.

The detections to build come from the shortlist in section 4.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

4. First steps

REMEMBER

- Build detection zero first: an alert for every log source that stops sending events.
- Inventory your log sources before you write detections, with each one's owner, retention, parsing, and normal event rate.
- Score the backlog before you build anything, then build the top 10.
- A candidate with telemetry readiness 0 becomes a log source request.
- Wait on coverage targets until you know your telemetry ceiling.

The temptation with a new detection program is to open the SIEM and start writing rules. I'd do one boring thing first. A rule is only as good as the logs under it. Until you know which logs you have, who owns them, and whether they're still arriving, you're writing detections for an environment you're guessing at. So the first detection here is the least exciting one I can offer, an alert that fires when a log source goes quiet.

Your path depends on what you already have. Do you have a log source inventory? A list of every source feeding the SIEM: its owner, its retention, whether it is parsed and enriched, its usual gap between events, and whether a silence alert covers it. Do you have the last 12 months of incidents, each tagged with the ATT&CK techniques the attacker used? How many engineering hours a week go to building detections, from the first draft through validation? Leave out triage and tuning.

If you have no inventory, build one, ship DET-0000 for every source, and then score. If you have an inventory and no incident history, take threat relevance from public prevalence lists and sector reporting. Score every candidate, and build the readiness 4 or higher ones first. At 3 the fields still need parsing, and below 3 the source needs collection or retention work, so research would send those candidates back. Ship the top 10. If you have both, score threat relevance 5 for every technique in your own incident history, and ship the top 10. The top 10 is the shortlist; you build down it at your work in progress limit: one detection at a time for a single engineer, and for a team the capacity rule. Work on no more detections at once than your weekly hours divided by the hours one detection takes from hypothesis through validate. Use 8 hours per detection until you've measured your own.

Three plans on this site cover the start, each for a different reader. This section picks the first detections for any team. Section 3.4 gives the first month for a team of one, of two to four, or of five or more. Section 11.3 gives the first 90 days to someone handed a function that already runs live detections. Its first month adds no new detections, because the live set comes first.

DET-0000, Log source silence: Alert when a source in the log inventory has sent nothing for three times its usual gap between events, and never sooner than 1 hour. Every other detection depends on its data arriving. A source covered by this alert is one step closer to a readiness score of 5.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

4.1 Build a baseline

The Build a baseline tool does the first pass of this for you. The baseline builder turns an environment into a first set of detections. Four questions each end as a list of log sources, because what is collected decides what can be detected. From the sources, ATT&CK's own analytics say which techniques are ready to build. For each ready technique, the library's best rule that your SIEM can run becomes a candidate, and the scorer's formula puts them in order. The baseline takes the best rule for every ATT&CK tactic the environment supports before it fills the rest in that order. The result is a backlog to start from, with DET-0000 first for every source, and a list of what the environment cannot see. Each environment it asks about ticks the log sources ATT&CK's analytics name for it:

Environment	Platforms in scope	Log sources it ticks
On-premises Windows and Active Directory	Windows, Identity Provider	12 sources, such as WinEventLog:Security, WinEventLog:System, WinEventLog:Application
Linux servers	Linux	10 sources, such as auditd:SYSCALL, auditd:EXECVE, auditd:PATH
macOS endpoints	macOS	7 sources, such as macos:unifiedlog, macos:endpointsecurity, macos:auth
AWS	IaaS	10 sources, such as AWS:CloudTrail, CloudTrail:Signin, CloudTrail:GetCallerIdentity
Azure and Entra ID	IaaS, Identity Provider	10 sources, such as azure:activity, azure:audit, azure:policy
Google Cloud	IaaS	5 sources, such as gcp:audit, gcp:iam, gcp:config
Kubernetes and containers	Containers	10 sources, such as kubernetes:audit, kubernetes:apiserver, kubernetes:events
Microsoft 365	Office Suite, SaaS	10 sources, such as m365:unified, m365:audit, m365:exchange
Google Workspace	Office Suite, SaaS	5 sources, such as gcp:workspaceaudit, Google Admin Audit, saas:googleworkspace
SaaS applications	SaaS	8 sources, such as saas:audit, saas:auth, saas:api
VMware ESXi	ESXi	9 sources, such as esxi:hostd, esxi:vpxd, esxi:auth
Network perimeter and devices	Network Devices	12 sources, such as networkdevice:Firewall, networkdevice:Flow, networkdevice:IDS

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

A library rule is a candidate for a technique when the technique is ready and your SIEM can run the rule. Running it means the rule is in the SIEM's language, or the translator converts it into that language; the library records which languages each rule converts to. On Sentinel, a translated rule counts by the table its KQL reads, either SecurityEvent or one of Defender's Device tables. A Device table needs Microsoft Defender for Endpoint ticked among your tools. On LogScale, it counts only when the translation reads Falcon's events, which needs CrowdStrike Falcon ticked among your tools. For Google SecOps, the translator writes YARA-L only for endpoint rules: process, registry, file, network, and DNS events on Windows, Linux, and macOS. For another SIEM only Sigma rules are candidates, to convert with that SIEM's own Sigma tooling. A translated rule also has to keep the rule's logic. The translator keeps a query's filter and drops the stages after it. A rule that filters again after counting, or joins another table, would match far more in translation: a password spray rule would match every failed login. Those rules are offered only in the language they were written in. One rule per technique: a rule in your SIEM's own language first, then the highest robustness draft, then the one that converts to the most languages, then the one from the source with the most rules. A second rule for the same technique is taken only when it reads a different log source. The rows cover the ATT&CK tactics before they go by score. Every tactic with a ready technique and a rule that runs here gets its best rule first, two each at a baseline of 40 or more and three at 60 or more. The rest go down the ranking, and no tactic takes more than a fifth of the rows while another still has a rule to add. Without that, discovery commands, which a third of ATT&CK's groups run, take many of the top slots, and ransomware falls below the cut. A tactic left with no rule is named with the reason: nothing ready, or nothing that runs on these sources. DET-0000 comes first for every log source you tick, before any candidate. Each library rule's log source is read from where its source writes it: a Sigma rule's logsource, a Splunk rule's data_source field, an Elastic rule's index patterns, the tables a KQL query reads, a Panther rule's log type, a Google SecOps rule's folder. Each maps to the ATT&CK log sources that record it. A rule runs in your environment when one of those sources is ticked, and a KQL rule that joins two tables needs one source for each. Rules written for one EDR's own fields, such as CrowdStrike's or SentinelOne's, run only when that EDR is ticked among your tools. A rule that reads another vendor's own table or a threat intelligence list has no ATT&CK source, so the baseline leaves it out. What the page does not know, it says: Your assets, so impact is 3 everywhere. Your incidents, so threat relevance is ATT&CK group usage. Your alert volume, so run cost is 3 everywhere. Whether a source's fields are parsed, so readiness is 5 for every collected source; lower it in the map or the record when they are not.

The baseline downloads as kit starters, one DET folder per rule at the intake stage, so the first backlog arrives in the repository already scored on everything but impact and run cost. A coverage map with your log sources ticked opens the baseline with them through Export, Build a baseline.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

5. The lifecycle

REMEMBER

- A detection leaves a stage only when its definition of done is met, and every definition of done is a count someone can check.
- Research ends with field names read from a real event and a 7-day count of how often the behavior happens normally.
- Deploy with alerting off, and switch alerting on only after 7 days at or under the alert volume threshold.
- Every tuning change is a pull request that goes back through validate.
- A retired detection keeps its record and gets a retirement note. Any replacement starts at intake with a link to that note.

I grouped the stages onto three lines. The Backlog line carries intake, hypothesis, and research; the Build line carries build and validate; the Live line carries deploy, tune, and retire. A tuning change loops back to validate as a pull request. The replay runs the changed rule against its fixtures, so an exclusion that blinds it to the attack fails the build. A retired detection's replacement loops back to intake with a link to the retirement note, so the next hypothesis starts from why the old one was pulled.

Stage	What it is for	Definition of done, counted	Owner	Metrics
1. Intake	Write down a request for a new detection with enough context that it can be scored.	6 scores, each with an evidence line, and 1 computed score	Detection lead	intake_to_score_days, backlog_age_days
2. Hypothesis	Write one sentence saying what the detection will catch, which log source it needs, and what it will miss.	1 hypothesis sentence, 1 named log source, 1 named blind spot	Detection engineer	readiness_at_hypothesis
3. Research	Produce the events the rule will be written against, and confirm they exist in your logs. Then count how often they appear during normal business.	1 emulation run, fixtures saved, every field checked against a real event, 7 days of baseline counts, lab torn down	Detection engineer	research_hours_per_rule

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	What it is for	Definition of done, counted	Owner	Metrics
4. Build	Write the detection in Sigma and the tests that prove it.	lint passes, 1 conversion per target, 1 event it must match, 1 event it must ignore	Detection engineer	percent_with_tests
5. Validate	Prove the detection fires on what it should catch, stays silent on what it should ignore, and still matches the query a reviewer approved.	fixtures pass on every replay engine, every converted query approved (matching its approved copy on a re-validation), 1 approval by someone other than the author (one-person team: the author after 24 hours)	Reviewer, a second engineer	first_pass_validation_rate, review_turnaround_hours
6. Deploy	Put the detection into production with alerting switched off, leave it running for 7 days, and switch alerting on only if the alert count stayed at or under its threshold.	7 days live with alerting off, alert volume at or under its threshold, then alerting on	Detection lead	deploy_lead_time_days, post_deploy_replay_failures, detection_latency_median
7. Tune	Keep the share of true alerts above the agreed floor as the environment changes.	precision checked every 30 days, running query matches the approved copy at each check	Detection engineer with the SOC or triage lead	precision_30d, triage_minutes_median, minutes_per_true_positive, upkeep_minutes_per_detection
8. Retire	Remove detections whose alerts cost more analyst time than they are worth.	status deprecated, alerting off, record kept, retirement note written, replacement linked if one exists	Detection lead	rules_retired_per_quarter, rule_debt

The metrics column leaves out the 15 metrics that measure the program as a whole: robustness_weighted_coverage, incident_detection_rate, silent_source_hours, priority_technique_gaps, misses_by_cause, open_detection_fixes_age, false_close_rate, time_to_scope_hours, time_to_contain_minutes, notifications_on_time, playbook_coverage, playbook_minutes_saved_per_alert, playbook_actions_unattended, playbook_actions_reverted, and incidents_without_followup.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Appendix A has the full definition of done wording, the mistake that usually stalls each stage, and where a detection goes when a stage's count can't be met.

6. Prioritization

> **REMEMBER**
>
> - Every number in a score needs an evidence line a reviewer can check.
> - Never type a score by hand. The scorer writes it, and CI fails any score that disagrees with the formula.
> - A telemetry readiness of 0 blocks the candidate, and you fix that with a log source request.
> - Mandates due within 90 days go first, in order of due date.
> - Scores in the same 0.2 band count as tied, and the cheaper detection goes first.
> - A candidate the live set already covers scores half, unless it catches a variant the live detection misses.
> - Publish the weights, so everyone can see why the backlog is in its order.

6.1 The inputs

Input	Scale	Question
threat_relevance (T)	1 to 5	How recently, and how close to you, has this technique been used?
impact (I)	1 to 5	What does the attacker reach if the technique succeeds?
robustness (B)	1 to 5	How hard is it for an attacker to slip past this detection using the data you have? Score it with the five Summiting the Pyramid levels: 1 matches something the attacker can change in seconds, 5 matches something the technique cannot work without.
telemetry_readiness (R)	0 to 5	Is the data this detection needs already in the SIEM, and in good enough shape to use?
build_effort (E)	1 to 5	How many hours of work from writing the hypothesis to passing validation?
run_cost (C)	1 to 5	How many analyst minutes a day will this detection consume? Expected alerts per day multiplied by the typical minutes to triage one. For a detection that runs as scheduled SQL, add the warehouse compute it burns, converted the same way your finance team converts it.

Appendix B has the wording for every level of every input.

Each input comes with one line saying where its number came from: an incident id, a report section, a query, or a ticket. CI rejects a record missing any of them.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

6.2 The formula

```
value = wT·T + wI·I + wB·B
cost  = wE·E + wC·C
score = (value / cost) × (R / 5)^k × M
```

A readiness of 0 sets the score to 0 and marks the detection blocked. M is the marginal coverage factor from section 6.3. Scores round to two decimals. With every weight at 1 and k = 1, scores run from 0.06 to 7.50. The shape comes from Weighted Shortest Job First in SAFe, which divides value by size. I added readiness as a factor because you can't build anything on missing telemetry, and the marginal factor so a detection is scored on the coverage it contributes. The 0.8 and 0.5 behind that factor are defaults I chose so a stronger replacement still outranks most new work and a duplicate falls below it. Nothing derives them. If your program disputes them, change them in score.py and say why in the pull request.

Six one-to-five judgments carry noise, so the order sorts scores into fixed bands 0.2 wide (2.40 to 2.59, 2.60 to 2.79, and so on). The width is a chosen default until a team measures how far its scorers disagree. Scores in one band count as tied, and the one with the lower run cost goes first. The band is the rule, so two scores on either side of a band edge are ordered even when they differ by less than 0.2.

6.3 Additions to an existing program

Scored one at a time, the eighth detection for a covered technique scores the same as the first for an uncovered one. The marginal coverage factor fixes that by scoring a detection on what it adds to what already fires.

It comes back	Factor	When	What to do about it
first	×1.0	No live detection names this technique.	Full value. This is coverage the program lacks.
stronger	×0.8	The technique has a live detection, and this candidate scores higher on robustness than the best one live.	Most of the value. It replaces something brittle, so plan the retirement of what it supersedes in the same change.
equal	×0.5	A live detection already covers the technique at the same robustness level or higher.	Half. Build it only if it catches a variant the live one misses, and say which variant in the hypothesis.

The pipeline computes it from the live set, comparing ATT&CK technique and robustness level. It is written into the record like the score; a hand-typed value fails the build. With no live set to compare against, every candidate counts as first.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

A candidate that comes back stronger ships in the same change as the retirement of what it supersedes, because leaving both live doubles the alerts for one technique. A candidate that comes back equal is a duplicate until somebody writes down the variant it catches that the live one misses. If the hypothesis can't name that variant, close the candidate.

6.4 Presets

Preset	Change	Use when	Where it goes wrong
balanced	none	The default. One score for the whole backlog.	Only as good as the six numbers, so every number needs its evidence line.
threat_led	wT=2	You have your own incident history or strong reporting about your sector.	Ranks by what attackers do, so it can pull effort toward techniques your logs can't see yet.
asset_led	wI=2	You know which systems matter most and how an attacker would reach them.	Only works with a current asset inventory.
telemetry_led	k=2	A new program that needs early wins from the data already in the SIEM.	Favors what your logs already show and can leave the biggest threats uncovered.

A detection required by a regulation or an audit carries a mandate with the framework, the requirement number, and a due date. If the due date is within 90 days, it goes to the front of the queue, ordered by date. Everything else is ordered by score, highest first; ties go to the detection with the lower run cost. A mandated candidate with telemetry readiness 0 is still blocked; the mandate moves its log source request to the front of that list instead. Mandated detections come with budgets and deadlines of their own.

6.5 A worked example

DET-0001, Kerberoasting (T1558.003), scores 4, 4, 4, 4, 2, 2 on the balanced preset: value 12, cost 4, scaled by 4/5, score 2.40. DET-0002, a scheduled task created by a non-admin user, scores 3, 2, 3, 3, 2, 4: value 8, cost 6, scaled by 3/5, score 0.80. The gap comes from run cost and readiness. Under threat_led the two score 3.20 and 1.10; under telemetry_led, 1.92 and 0.48. The order doesn't change under any of the three.

6.6 The floor, and when a detection has to go

A detection's floor is its typical triage minutes divided by the program cap on analyst minutes per true positive. The cap is 120 by default. A detection that takes 30 minutes to triage must be right at least 25% of the time; one that takes 6 minutes, 5%.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Any one of these triggers proposes retirement, and the lead makes the call:

1. Its true-positive share stayed below its floor for two 30-day periods in a row.
2. It fired at least 10 times in 180 days and was never right.
3. The log source it needs has been missing for 30 days straight, so its readiness fell to 0. Suspend it with alerting off and raise a log source request, and retire it only if the source owner declines to restore the source.
4. The technique or the system it protects is gone, for example because the platform was decommissioned.
5. Another live detection covers the same technique at an equal or higher robustness level and catches everything this one catches. Retire the weaker one in the same change that ships the stronger one.

A detection that hasn't fired at all in 365 days gets a review before anyone retires it, because either its data stopped or its hypothesis was wrong.

7. Worked examples

> **REMEMBER**
>
> - Work a score in order: value, cost, their ratio, the readiness factor, then marginal coverage.
> - A cheap detection still loses when an attacker can evade it by renaming a tool.
> - Retire a detection when a trigger fires, and write the retirement note before you switch it off.

The examples below take DET-0001 to DET-0005 through the decisions one at a time. DET-0101, the password spray detection whose record is on the site, appears only in the ordered backlog. The environment details are illustrative, and the repository's own scorer computes every number when this paper is generated, so the numbers can't drift from the code.

7.1 Scoring one candidate

DET-0001 detects Kerberoasting (ATT&CK T1558.003). Each of its scores answers one question and carries one line of evidence. The evidence line makes the score arguable, because a reviewer can dispute its source.

Input	Score	The question it answers	The evidence line on the record
threat relevance (T)	4	How recently, and how close to you, has this technique been used?	sector ISAC report, Q2 2026, section 3

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Input	Score	The question it answers	The evidence line on the record
impact (I)	4	What does the attacker reach if the technique succeeds?	service account escalation path to tier-0
robustness (B)	4	How hard is it for an attacker to slip past this detection using the data you have? Score it with the five Summiting the Pyramid levels: 1 matches something the attacker can change in seconds, 5 matches something the technique cannot work without.	event 4769, RC4 ticket encryption, non-machine account
telemetry readiness (R)	4	Is the data this detection needs already in the SIEM, and in good enough shape to use?	DC security logs parsed, no enrichment
build effort (E)	2	How many hours of work from writing the hypothesis to passing validation?	6 hours, comparable to DET-0007
run cost (C)	2	How many analyst minutes a day will this detection consume? Expected alerts per day multiplied by the typical minutes to triage one. For a detection that runs as scheduled SQL, add the warehouse compute it burns, converted the same way your finance team converts it.	10 analyst minutes per day, estimated before any 7-day count

On the balanced preset, where every weight is 1, the arithmetic goes like this:

1. Add the value side. value = 4 + 4 + 4 = 12.
2. Add the cost side. cost = 2 + 2 = 4.
3. Divide value by cost. 12 / 4 = 3.00.
4. Apply the readiness factor. Readiness is 4 out of 5, so the factor is (4/5) = 0.80. 3.00 × 0.80 = 2.40.

A score of 2.40 only means something when you compare it with the rest of the backlog. On the balanced preset, scores run from 0.06 to 7.50, and the median of the six example records in this section is 1.50. This candidate sits above the middle, so it's worth building before most of what's waiting. The scorer writes the score into the record, and CI fails a score typed in by hand.

7.2 Two candidates in a different order

DET-0002 detects a scheduled task created by a non-administrator (T1053.005). It's a reasonable detection, and it still ranks below DET-0001 for three reasons: a smaller value side, a higher run cost, and lower readiness.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Input	DET-0001 Kerberoasting	DET-0002 Scheduled task
threat relevance (T)	4	3
impact (I)	4	2
robustness (B)	4	3
telemetry readiness (R)	4	3
build effort (E)	2	2
run cost (C)	2	4
value	12	8
cost	4	6
score	2.40	0.80

1. Add the value side. value = 3 + 2 + 3 = 8.
2. Add the cost side. cost = 2 + 4 = 6.
3. Divide value by cost. 8 / 6 = 1.33.
4. Apply the readiness factor. Readiness is 3 out of 5, so the factor is (3/5) = 0.60. 1.33 × 0.60 = 0.80.

Run cost and readiness make most of the gap. The run cost evidence on DET-0002 reads "80 analyst minutes per day", which scores 4; on DET-0001 it reads "10 analyst minutes per day, estimated before any 7-day count", which scores 2. Both are built on parsed logs, but DET-0001's source is the better prepared of the two.

Changing the weights doesn't rescue it. Each preset leans the score a different way, and the order stays the same under every one:

Preset	DET-0001	DET-0002	Order
balanced	2.40	0.80	unchanged
threat_led	3.20	1.10	unchanged
asset_led	3.20	1.00	unchanged
telemetry_led	1.92	0.48	unchanged

That's why I wrote the weights down. When somebody argues for a different lean, you change one number and the backlog re-sorts in front of them.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

7.3 A candidate the score blocks

DET-0004 detects a program download to a PLC from an engineering workstation (ATT&CK for ICS T0843). It's the highest-value candidate in the sample: impact 5, robustness 5, threat relevance 4. Its value side adds to 14, the highest in the sample, and it scores 0.00.

Telemetry readiness is 0: no OT network monitoring log source ingested. A readiness of 0 sets the score to zero and marks the record blocked. A detection can't alert on telemetry nobody collects, so the work item becomes a log source onboarding ticket. You argue for it as an onboarding ticket, with the detection attached as the reason.

Once you ingest and parse the log source, readiness moves to 4. The same value and the same cost (build effort 5, run cost 1) give 14 / 6 × 0.80 = 1.87, third by score in this backlog. Nothing else about the candidate changed.

7.4 A candidate the live set already covers

DET-0005 detects the same Kerberoasting technique as DET-0001, by matching the name of the tool that requests the ticket. It's cheap, at two hours to build and four analyst minutes a day. On its inputs alone it scores 3.60, better than DET-0001, because it costs so little.

Input	Score	The evidence line on the record
threat relevance (T)	4	same sector ISAC report as DET-0001, Q2 2026, section 3
impact (I)	4	service account escalation path to tier-0, same as DET-0001
robustness (B)	1	matches the requesting process name, a filename the attacker renames in seconds (Summiting level 1)
telemetry readiness (R)	4	DC security logs parsed, no enrichment
build effort (E)	1	2 hours, the query is a filter on an existing search
run cost (C)	1	4 analyst minutes per day from the 7-day baseline

The record is scored for the day DET-0001 goes live, because that's when the comparison matters. In the example set no live record covers T1558.003 yet, because DET-0001 is still at build, so DET-0005's record carries the marginal state by hand. In a repository with a live set the pipeline compares the candidate against it, finds T1558.003 covered at robustness 4 against this candidate's 1, and marks it equal. That halves its score to 1.80. A low cost doesn't make up for low robustness. If an attacker renames the binary, this detection is worth nothing, while the behavioral detection that's already live keeps firing.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

The marginal factor is there for this case. Without it, a backlog fills up with cheap variants of what already works. Each one scores well on its own, and the coverage chart climbs while nothing new gets detected.

That doesn't make the candidate automatically wrong. If somebody can write down a variant this one catches and DET-0001 misses, it earns its place at half score. If nobody can, close it.

7.5 A detection somebody else's deadline decides

DET-0003 detects an administrator logging in to the cardholder database outside a change window. On the numbers it scores 1.20, mid-table. It still goes first, because it carries a mandate: PCI DSS 4.0, requirement 10.4.1, due 2026-11-30. Measured from 2026-09-19, the as-of date the example is computed on, that's 72 days away, inside the 90-day window, so it sorts to the front by due date.

The scorer orders the whole six-record backlog like this. Live records stay in the list so every candidate is compared against the same set:

Order	Id	Detection	Score	Status	Why it sits there
1	DET-0003	PCI cardholder DB admin login outside change window	1.20	mandated	mandate due 2026-11-30
2	DET-0101	Password spray against Entra ID sign-in	2.60	in_progress	by score
3	DET-0001	Kerberoasting	2.40	in_progress	by score
4	DET-0005	Kerberoasting, RC4 ticket request by a known tool name	1.80	scored	by score
5	DET-0002	New scheduled task by non-admin user	0.80	live	by score
6	DET-0004	OT engineering workstation PLC program download	0.00	blocked	readiness 0

Keep mandated work in the same backlog with its reason recorded, so the time it takes shows up in the same place as everything else.

7.6 Retiring a detection

Retirement is arithmetic, then a check against the triggers. The figures in the two cases below are illustrative inputs, so use the method with your own measurements.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Measure the detection over 30 days: how many alerts it raised, how many were true positives, and how many minutes a typical triage took. Its floor is the triage minutes divided by the cap on analyst minutes per true positive, 120 by default. Compare precision with the floor in two consecutive 30-day windows. One window below the floor triggers nothing, and two in a row is a retirement trigger.

Case	Alerts, 30 days	True positives	Precision	Triage mins	Its floor	Mins per true positive	Verdict
Keeps its place	60	22	36.7%	25	20.8%	68 min	clears the floor
Retires	900	27	3.0%	14	11.7%	467 min	below the floor, second window in a row

The detection that stays is expensive per alert and still clears its floor. Its 60 alerts for 22 real findings cost 1500 analyst minutes, about 68 minutes per true positive and inside the 120-minute cap. The detection that retires is cheap per alert and costly in total. Its 900 alerts at 14 minutes each come to 12,600 minutes, or 210 analyst hours for 27 findings.

Then check the triggers. Any one of them proposes retirement:

1. Its true-positive share stayed below its floor for two 30-day periods in a row.
2. It fired at least 10 times in 180 days and was never right.
3. The log source it needs has been missing for 30 days straight, so its readiness fell to 0. Suspend it with alerting off and raise a log source request, and retire it only if the source owner declines to restore the source.
4. The technique or the system it protects is gone, for example because the platform was decommissioned.
5. Another live detection covers the same technique at an equal or higher robustness level and catches everything this one catches. Retire the weaker one in the same change that ships the stronger one.

When one fires, decide between tuning and retiring. Tune when the noise has one identifiable source you can exclude without losing the behavior, such as a service account, a backup window, or a scanner. Retire when the only way to quiet it is to narrow it until it no longer detects the thing it was written for. If you can't say which exclusion would fix it, you're retiring it.

To retire it, set the Sigma status to deprecated, disable the alert action, and keep the record and its history. Write a note saying which trigger fired and what you measured. If the technique still matters, its replacement enters intake with a link back to that note. That's the second loop in the lifecycle.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

A detection that hasn't fired at all in 365 days gets a review before anyone retires it, because either its data stopped or its hypothesis was wrong. The silence alert on its log source tells you which.

8. Emulation

> **REMEMBER**
>
> - Write the plan and get a second person's approval before you provision anything.
> - Start from a believable foothold, with only the permissions an attacker would have at that point.
> - Read field names from the raw event, because a field name taken from memory produces a detection that never fires.
> - Never write a fixture by hand. Save each one from an emulation run, a research query, or an incident's missed events, and scrub credentials, tokens, and personal data out of it before you commit it.
> - Record missing or late telemetry as a finding, and turn it into a log source request.
> - Tag every resource and identity with the run tag. Destroy attacker-created leftovers before you tear down the provisioned infrastructure.
> - In a cloud sandbox, Stratus Red Team marks every call with its run's uuid. Filter the telemetry by the mark, and never write the rule on it.

Emulation is how a detection gets its events. On an endpoint you can often detonate a sample and read the log. In cloud, SaaS, and identity there's no sample. You model the victim environment and the starting identity, provision a lab, run the behavior from that identity, and read what landed in the logs.

The research stage produces the fixtures you test a detection against and the field names you write it from. Both come out of an emulation run. For most endpoint techniques a scripted test does the job, and Atomic Red Team, from Red Canary, has one for many ATT&CK techniques. Cloud, SaaS, and identity techniques usually need the planned run described below.

8.1 Starting a run

Mode	The question it answers	What you start from	What to watch
Intelligence-driven	Can the behavior described in public research be reproduced?	A report, advisory, campaign writeup, or a CTID emulation plan	Stay faithful to the source. Don't invent victim details that make the run easier.
Detection-driven	Does a live detection fire on the behavior it claims to catch?	An existing detection	The detection names the API calls and fields. You infer identity, permissions, and environment, so flag those as assumptions.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Mode	The question it answers	What you start from	What to watch
Coverage-driven	Which techniques have no detection today?	An ATT&CK technique or log source gap	The gap goes straight to intake as a candidate.
Hypothesis-driven	Could an adversary realistically abuse this service or permission?	A threat model of a service or an IAM pattern	This start needs the most judgment. Model what an adversary would do from a stolen identity.

8.2 Scope

Scope	What it covers	Effort	Use it for
Atomic	One technique, one signal question.	Hours	Proving a single call lands in the log with the fields you expect.
Micro	Two or three chained behaviors, for example discover, stage, exfiltrate.	Days	Detections that depend on sequence or on an artifact an earlier step creates.
Full	End to end, across planes and surfaces.	Weeks	Reproducing a published campaign, or testing a whole tactic chain before a board-level claim.

The lifecycle is the same at every scope. Scope changes the source material and how much of the victim environment has to be real.

8.3 Plan before you provision

Answer these, then get a second person to approve them:

1. State the objective, and what the source says and leaves out.
2. Describe the threat model: what the adversary is trying to achieve.
3. Describe the victim model, with its resources, controls, identities, and network paths.
4. Name the starting identity, and what the attacker has to earn later.
5. Write the operational flow: each call, its actor, its target, the expected result, and the log it should land in.
6. Estimate the cost if metered services are involved.
7. Name the second person who will approve the plan before anything is provisioned.

Keep the plan with the detection, because it records why you ran each command.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

8.4 Keep the run realistic

- Start from the lowest privilege the hypothesis assumes: stolen long-term credentials, an assumed role, or compromised compute. An admin session is a believable start only for a hypothesis that begins with one, and the record has to say that.
- Use only the permissions that stage would have. Granting AdministratorAccess to make the run succeed invalidates the telemetry.
- Respect dependencies. Earlier actions enable later ones, and the artifacts they create are part of the trace.
- Keep the failures. A denied call during recon is often the exact event a detection should chase.
- No 0.0.0.0/0 ingress rules, even in a lab.
- Account for cloud timing. IAM is eventually consistent. Poll for readiness, and give logs a settle window before calling telemetry missing.

Give every run a unique tag, a name plus a short random id. Put it in resource names, identity names, and session names, so you can filter telemetry to the run and find leftovers during teardown. A resource tag alone won't appear in an ARN.

8.5 Verify the telemetry

1. Decide before the run which log should capture each step, and confirm that log is switched on. Data-access logging often ships off.
2. Open the raw event and read the field names from it. Names taken from memory or documentation are often wrong, and then the detection can't fire.
3. Look for what is late, partial, or absent as carefully as for what arrived.
4. Ask the last question: does this data distinguish the attacker from normal activity?

A technique that leaves no useful trace is still a result, so record it. Score the candidate readiness 0 and turn it into a logging request.

8.6 Test coverage against the exercise

Replay the run's events against the detections you already have, and read what fires and why:

Outcome	What it means	What to do
Fires for the right reason	The detection matched the behavior it was written for.	Nothing. Record the run id in the detection's Validation section.
Fires on a different link in the chain	Something alerted, but on a different step.	Decide which step you meant to catch. The gap may still be open.
A different detection fires by accident	Coverage that depends on a detail the attacker can change.	Treat as uncovered. Score the candidate.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Outcome	What it means	What to do
A mapped detection never touches this telemetry	Inventory coverage. The heatmap is green and the log is silent.	Retire or rewrite the detection.

The most expensive outcome is a mapped detection that never touches the telemetry it claims to cover, because it was counted as coverage and passed an audit.

Not every gap becomes a detection. Pick the response that fits:

- Write a new detection.
- Tune an existing one, naming the exclusion.
- Request a logging change, because the data is absent or partial.
- Hunt, because the behavior is too rare or too variable to alert on.
- Run a wider emulation, because the question is still open.

8.7 Teardown, and the run report

1. Treat the lab as a state machine. Don't redeploy on top of a half-dead environment.
2. Destroy attacker-created leftovers first: backdoor identities, keys, grants, topics. Infrastructure as code doesn't know about them.
3. Then destroy what the code provisioned.
4. Turn off any logging that was enabled only for the run.
5. Verify the teardown by run tag, and let the check fail loudly.

Write the run up as a short engagement report:

- Objective, scope, and source material
- Threat model and ATT&CK mapping
- Victim model and assumptions, each with the date it was true
- Operational flow: action, API call, identity, result
- Observations, including denials and identity progression
- Telemetry that appeared, and telemetry that did not
- Coverage outcome per detection that fired
- Infrastructure summary and teardown status
- Recommendation: new detection, tune, logging change, hunt, or more research

Date the environmental claims. When a detection gets noisy six months later, somebody will ask what normal meant when you wrote it. If nobody can answer, the comfortable move is to raise the threshold until it goes quiet.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

8.8 Automation

Automate:

- Drafting the threat and victim model from a source
- Provisioning and tagging
- Executing with the right identity
- Tracking orphans
- Checking fields against real events
- Searching existing coverage
- Writing the run up and verifying teardown

Keep with the engineer:

- Whether the scenario is realistic
- Whether the assumptions are true
- Whether existing coverage is equivalent
- Whether the behavior is worth detecting at all

Adapted from Elastic Security Labs, "Cloud Threat Emulation on Autopilot: Context is Everything", Terrance DeJesus and Bryan Porras Blanch, 21 September 2026. The plan-first shape comes from the MITRE Center for Threat-Informed Defense Adversary Emulation Library.

8.9 Practical emulation in the cloud with Stratus Red Team

Stratus Red Team, from Datadog, is Atomic Red Team for the cloud. It is one Go binary, under the Apache 2.0 license, that carries its attack techniques. Each is one step an attacker has been documented taking, with its prerequisites, its detonation, and its cleanup. In this lifecycle a technique is an atomic run (8.2), and it fits the detection-driven and coverage-driven starts (8.1). Its warm-up provisions, so the plan of 8.3 and its second approver come first, as for any run.

At commit df3316f of 2026-11-04 it carried 103 techniques: 46 for AWS, 25 for GCP, 15 for Azure, 8 for Kubernetes, 7 for Entra ID, 2 for EKS. Every technique names its ATT&CK tactic, and 16 also name an ATT&CK technique id. 89 provision prerequisites with Terraform, 61 can revert their detonation, and 36 ship with a detonation log, a real event from one of the project's runs. The catalog those counts come from is published beside this site as emulation/stratus.json, and the weekly library job rereads it from the source.

One command set covers six platforms. A team with an AWS account, an Azure subscription, and a Kubernetes cluster runs the same research stage on each, with the same record lines and the same teardown check. What changes per platform is how you authenticate, which log receives the detonation, the field that names the action, and where the run's mark lands.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Platform	Techniques	You authenticate with	The detonation lands in	The field that names the action	Where the run's mark la
AWS	46	AWS credentials in the environment or a profile, for example through aws-vault	CloudTrail	eventSource and eventName	userAgent, as stratus-re by the run's uuid, on the Terraform calls and the Resource names carry th characters, and every Te is tagged StratusRedTea
GCP	25	gcloud auth application-default login, and GOOGLE_PROJECT	Cloud Audit Logs, Admin Activity	protoPayload.methodName	requestMetadata.callerS with the same prefix and names carry the short id
Azure	15	az login, and AZURE_SUBSCRIPTION_ID	The Activity Log	operationName.value	The SDK sends the uuid id, and the Activity Log r agent. The run's resourc resource names carry th they are the mark.
Entra ID	7	az login	Entra ID audit logs	The activity type, such as Add application	As for Azure. The applica and users the run create name or the short id in th names.
Kubernetes	8	The current kubectl context; the tool does not create a cluster	The API server audit log	verb and objectRef.resource	userAgent, with the same The names of the pods, bindings carry the short
EKS	2	AWS credentials and the cluster's kubeconfig; the tool refuses a cluster that is not EKS	The control plane audit log, in CloudWatch, once audit logging is switched on	As for Kubernetes	As for Kubernetes.

The run, step by step, on one AWS technique. Each step names the command and the line of the record it fills.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Step	Command	What happens	What the record gets
Choose	`stratus list --platform aws`	The list has one platform's techniques with their tactics, and stratus show prints a technique's warm-up and detonation as steps. For an AWS technique with a published detonation log, read the log first. The field names then come from a real event before anything is provisioned.	The technique id, and the field names, in the Technical Context.
Plan		The plan of 8.3, with the technique's warm-up and detonation as its operational flow. The tool runs as the identity you authenticate, so that identity is the starting identity, with the permissions the hypothesis assumes and no more. A second person approves.	Plan approved by: who, and when.
Set the run tag	`export STRATUS_RED_TEAM_CORRELATION_ID=$(uuidgen)`	The tool makes a uuid for every run and puts it in the mark. Setting it yourself records it before the run, and keeps the run's state apart from another run of the same technique.	Run tag: the uuid.
Warm up	`stratus warmup aws.defense-evasion.cloudtrail-stop`	Terraform creates the prerequisites, named with the tool's prefix and the short id. Nothing attacks yet, and stratus status reads WARM.	Nothing yet.
Detonate	`stratus detonate aws.defense-evasion.cloudtrail-stop`	The technique's calls run as the authenticated identity, each with the mark. Give the log its settle window before calling telemetry missing; CloudTrail delivers in minutes.	The time of the detonation.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Step	Command	What happens	What the record gets
Read the events	`a query on the log for the mark, in the run's window`	Open the raw events and read the field names from them. The detonation's events are the positive fixtures. The warm-up's calls carry the same mark and make the first negative fixtures: the same identity and window, a different action.	Fixtures in tests/events.jsonl, scrubbed of account ids, principals, and addresses, each with the run's uuid as its fixture_source.
Revert, then clean up	`stratus revert aws.defense-evasion.cloudtrail-stop && stratus cleanup aws.defense-evasion.cloudtrail-stop`	Revert undoes the detonation's side effect where it has one. Cleanup destroys the prerequisites, and cleanup with --all cleans every technique that can be cleaned.	Nothing.
Verify the teardown	`stratus status, then the same query after the cleanup`	Status must read COLD. The log must show the deletes with the mark, and a search by the short id and the StratusRedTeam tag must return nothing.	Teardown verified: the date, and the query that returned nothing.

The event a rule is written from, as the project published it for aws.defense-evasion.cloudtrail-stop, with the account, key, principal, user, and address scrubbed the way a fixture is before it is committed:

```
{
  "eventSource": "cloudtrail.amazonaws.com",
  "eventName": "StopLogging",
  "eventTime": "2024-07-31T13:06:24Z",
  "eventType": "AwsApiCall",
  "awsRegion": "<region>",
  "readOnly": false,
  "requestParameters": {"name": "stratus-red-team-ct-stop-trail-buykxbqejv"},
  "responseElements": null,
  "sourceIPAddress": "<address>",
  "userAgent": "stratus-red-team_c97089f1-1ae3-4ecc-b006-f5e8fd0f2571",
  "userIdentity": {
    "type": "IAMUser",
    "arn": "arn:aws:iam::<account>:user/<user>",
    "accountId": "<account>",
    "accessKeyId": "<key>",
```

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

```
    "principalId": "<principal>",
    "userName": "<user>"
  }
}
```

The rule is written on eventSource and eventName, and on nothing the run put there. The trail name and the user agent are the run's, and a rule that matched on either would fire on nothing real. The kit's lint fails such a rule.

The four lines the record's Technical Context needs after such a run, with made-up names:

```
Plan approved by: R. Second, 2026-10-16, for aws.defense-evasion.cloudtrail-stop in the
sandbox account, from an IAM user with cloudtrail:StopLogging and nothing wider.
Run tag: 6c1f0a2e-7b3d-4c58-9e21-5f0d8a7b4c13. Resources stratus-red-team-ct-
stop-6c1f0a2e-*; the mark is stratus-red-team_6c1f0a2e-7b3d-4c58-9e21-5f0d8a7b4c13 in
userAgent.
Baseline: StopLogging events in production CloudTrail, 2026-10-09 to 2026-10-15: 0
rows.
Teardown verified: 2026-10-16. stratus status reads COLD; DeleteTrail and DeleteBucket
were seen with the mark; a search for the short id and the StratusRedTeam tag returned
nothing.
```

What the tool leaves to you:

- Point it at a sandbox account, subscription, project, or cluster, and never at production. Its own documentation says so, and the plan's approver is the person who checks the account id in the plan.
- Never write the rule on the mark. The kit's lint fails a rule whose values contain the tool's mark or the record's run tag, because such a rule fires on nothing real.
- Pick the identity. The tool runs as whoever you authenticated, and 8.4 says that is the lowest privilege the hypothesis assumes. The record names the identity and its permissions.
- Detonate once unless the technique is idempotent. 59 of the 103 techniques can be detonated again without a revert. The rest need stratus revert first, and --force on one that has a side effect leaves two side effects to clean.
- The published detonation logs are the project's, from its own runs, and for AWS only. They give you the field names early. The fixture still comes from your run, in your tenant, with your identity.
- The list is the tool's. A technique it lacks still goes through the planned run of 8.3, or becomes a contribution to the tool.
- The warm-up has a bill. Some techniques launch instances or create a cluster's worth of resources, so the plan's cost estimate covers the warm-up as well as the detonation.
- A config file at ~/.stratus-red-team/config.yaml adds a prefix and tags to the AWS resources, so teams sharing a sandbox can tell their runs apart.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

No technique has been detonated from this repository. The commands and flags here were run offline against a binary built from the source: list, show, and each command's help. The fields and the marks come from the source and from the published detonation logs, and the outside checks page lists this as untested.

9. Detections in SQL, and the variables that tune them

A detection that runs as scheduled SQL against a data warehouse goes through the same lifecycle, with the same definitions of done, score, and floor. Its thresholds move into declared variables, and every run costs compute.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

9.1 Detection variables

Most tuning changes a number: a threshold moves, a lookback widens, or an account joins an allowlist. Written inline in the query, each of those is a logic change. You re-review it, re-test it, and re-convert it for every target, and the reason for the old value gets lost. Declared on the record, each value keeps its evidence and its history. The kit's lint checks every declaration today. It doesn't substitute the values into a query or replay at the bounds yet, so a tuning change edits the rule and variables.yml in the same pull request.

Kind	Type	Example	What it tunes
threshold	integer or decimal	`min_failed_logons, default 10, bounds 5 to 50`	How much of the behavior has to happen before it is worth an alert.
window	interval	`lookback_minutes, default 60, bounds 15 to 1440`	How far back the query reads. It sets both the sensitivity and the warehouse bill.
allowlist	list of strings, or a reference to a table	`allowed_service_accounts, default empty, sourced from a governed table`	The named exclusions. Empty by default, so an exclusion is always a visible act.
scope	list of strings	`in_scope_domains, default every domain the log source covers`	Which part of the estate the detection runs against, for staged rollout and for OT segments that need their own thresholds.

Kind	Type	Example	What it tunes
severity_cutoff	integer	`page_above, default 3, bounds 1 to 5`	What reaches a human immediately against what waits in a queue.

1. Every variable is declared once on the record with a type, a default, and, for thresholds and windows, bounds. Nothing reads a bare number out of the query.
2. The default carries one line of evidence, the same standard the six scoring inputs carry.
3. A variable change is a pull request against the record. Nobody edits it in the SIEM, so every threshold change stays in the detection's history.
4. A threshold needs one fixture that still fires at the upper bound and one that stays silent at the lower bound, so a value that works only at its default fails. The kit doesn't replay at the bounds yet, so the reviewer checks each bound against the fixtures.
5. Allowlists are variables too, and they are the only sanctioned way to exclude a known source of noise.
6. A variable changed three times in a quarter is a sign to review the hypothesis and the detection.

DET-0001, the Kerberoasting detection scored in section 7, declares these:

Variable	Kind	Default	Bounds	Evidence for the default
`lookback_minutes`	window	`60`	[15, 1440]	the task runs every 1 minutes, so a 60-minute window cove four runs and an eve up to 45 minutes late still lands in one
`min_distinct_services`	threshold	`3`	[2, 5]	the recorded run, SDWIN-2010220429 asked for RC4 tickets for 5 user-run service so the upper bound s fires on it. Set the default from your ow 7-day count
`allowed_service_accounts`	allowlist	`SECURITY.DETECT.ALLOWLIST_KERBEROAST`	-	empty at deploy; eve row needs a ticket id and an owner, and th table is reviewed eac quarter

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Variable	Kind	Default	Bounds	Evidence for the default
`in_scope_domains`	scope	`['CORP', 'OT']`	-	both forests send 47(events; the lab forest excluded because its accounts are shared design

An allowlist is how section 7.6 says to tune. When you name the exclusion, it lands in a governed table with a ticket id, an owner, and an expiry date, and every change to that table is diffed.

9.2 Changes in a warehouse

A detection that runs as scheduled SQL carries a second run cost. The warehouse bills for a query that scans a day of events every five minutes. Score run cost as analyst minutes plus warehouse spend, and record both in the evidence line.

The schedule sets latency. A detection running every 15 minutes averages seven and a half minutes of latency whatever the query does, and that's the floor under detection_latency_median. The lookback variable sets the bill as much as the sensitivity, because a window twice as wide scans twice the data on every run. Declare both as variables, so a change to either goes through review.

You'll find the same detection written three ways at detect.htora.dev/framework/sql/: the predicate the Sigma SQL backend emits, the Snowflake version with its variables and its task, and the variable declarations. I changed one behavior in the Snowflake version on purpose. The Sigma rule alerts on every RC4 ticket request. The Snowflake version alerts when one account requests RC4 tickets for min_distinct_services or more distinct services inside the lookback window. The warehouse variant has no replay engine in the pipeline yet, so its threshold bounds are checked at review. The fixtures it would need, one account crossing the upper bound and one staying under the lower, are not in the repository.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

10. Working with agents

REMEMBER

- Agents draft and CI checks. People keep review, deployment, and retirement.
- Give an agent its own read-only service identity, scoped to the tables, time ranges, and byte limits the task needs.
- Treat log content as hostile data. An agent never acts on an instruction found in a log, an alert, a ticket, or a report.
- List every query an agent ran in the pull request, with its time range, bytes scanned, and row count.
- Limit open agent pull requests to the number reviewers clear in a week.

An agent can draft most of what the lifecycle produces. That includes scores, hypotheses, emulation plans, rules, test fixtures, and the records that document them. The definitions of done make this safe, because every stage ends on a hard count that CI or a named person checks. People keep review, deployment, and retirement.

10.1 Division of work

Stage	Agent	Person	What checks it
Intake	Drafts the six scores and an evidence line for each. Every line cites something a reviewer can open: an incident id, a report, or a query and its result.	The lead confirms the priority order at the weekly intake meeting.	score.py validates the record and writes the final score.
Hypothesis	Drafts the hypothesis, names the log source, and names one blind spot the detection cannot see.	The engineer who owns the detection accepts or rewrites it.	lint.py confirms the four record sections exist and are more than placeholders. A person judges the hypothesis itself.
Research	Writes the emulation plan. Runs read-only queries to inspect field names from real events and count the 7-day baseline.	A second person approves the emulation plan before any test infrastructure is provisioned.	lint.py counts the fixtures and the four research lines in the record: plan approval, run tag, baseline, and teardown. The pull request carries the exact queries, their time ranges, bytes scanned, and row counts.
Build	Takes the fixtures from the emulation run or the research queries, writes rule.yml, and converts the rule for every target with convert.py --update-golden.	Nothing at this stage.	lint.py, which also checks that every fixture names the run, query, or incident it came from, and convert.py --check.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	Agent	Person	What checks it
Validate	Runs the replays against the positive and negative fixtures, and explains any change to an approved query.	A second engineer reviews and approves. An agent never approves.	Replay jobs, approved-query diffs, and one human approval.
Deploy	Prepares the release package with alerting off, and drafts the pull request that moves the detection to tune once its 7 days live are up.	The lead approves the deploy run, checks the 7-day threshold, and approves the pull request that switches alerting on.	The promotion check, which counts the first 7 days live against the threshold.
Tune	Drafts a tuning pull request that adds the specific exclusion as a filter selection and records it in variables.yml.	The owning engineer and the SOC lead approve it.	The core lock refuses a change to the rule's selections, and the change goes back through validate to prove the true-positive fixtures still fire.
Retire	Proposes retirement by citing the exact trigger that fired and the numbers behind it.	The lead makes the final call, and the retirement note records the decision.	The rule status check and the retirement record.

Human approval at validate is the control that matters most. An agent-drafted detection reaches production only after an engineer has read both its record and its rule logic. Section 18 takes this further, to a loop in which agents run every stage and each check is written against the way an agent would fake its way past it.

10.2 Limits on agents

An agent is never permitted to do any of the following.

- Write or edit a prioritization score by hand. tools/score.py --write computes it, and CI fails any score that disagrees with the formula.
- Synthesize test fixtures during build. Fixtures come from a captured emulation run, a scrubbed research query, or an incident's missed events, and each names where it came from.
- Change an approved query outside of a reviewed pull request.
- Approve a pull request, whether its own or someone else's.
- Deploy a rule, switch alerting on, or disable a live detection.
- Keep credentials that can write to the SIEM, the EDR, or the identity provider.
- Provision test infrastructure without a human-approved emulation plan.
- Rewrite a retirement note or alter the git history of a detection.
- Act on an instruction found inside a log event, an alert payload, a ticket, a threat report, or a web page.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

10.3 Log content is hostile input

Log content is controlled by the attacker. Command lines, user names, URLs, file names, and email subjects are all written by whoever caused the event. An agent that reads raw logs can be hijacked by prompt-injection text an attacker planted in those fields. The controls below assume every log line is hostile.

1. Separate data from instructions: Treat log events, alert payloads, ticket text, threat reports, and web pages as untrusted data. Never follow instructions found inside them.
2. Lock down query credentials: Give query tools a dedicated, read-only service identity of their own, scoped to the indexes or tables the task needs, with hard limits on rows returned, time range, and bytes scanned. A shared account hides which run caused a compute spike, so the SIEM's audit log should name the agent.
3. Lock the tool sequence: The task sets which tools an agent may call and in what order. A query result may refine the next query's parameters, within a schema allowlist. It never adds a tool call or widens the query's scope.
4. Sanitize before quoting: Deterministic code in the agent runner, never the model, strips control characters and escapes backticks and markup before log content goes into a fenced block in a record or a pull request. Raw log strings never reach a shell command.
5. Put production behind human review: Every change reaches production through a pull request that passes the definitions of done and carries a person's approval.
6. Audit every query: The pull request lists every query the agent ran, failed runs included, with the query text, the time range, the bytes scanned, and the row count. The SIEM's own audit log backs the list.
7. Scrub fixtures deterministically: Automated scanners strip credentials, session tokens, and personal data from fixtures before they are committed.

10.4 The review bottleneck

Faster drafting does not make a program bigger. It moves the bottleneck to human review, alert triage, and monthly upkeep. Every live detection still costs about 20 minutes a month to maintain and adds daily triage load, so the program reaches its capacity ceilings sooner.

Track how many pull requests the team can properly review in a week. Cap the number of open agent pull requests at that limit.

The scored backlog still decides what gets built. An agent pulls work from the top of the backlog and introduces new candidates only through intake.

10.5 Three metrics for agent work

Drafting speed measures compute. These three metrics show whether agent-drafted rules survive human review and production.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Unit	Aggregation	Definition
`intake_to_review_ready_hours`	hours	median per month	Hours from a candidate entering intake to a pull request that passes every automated check and waits only on human review.
`agent_draft_acceptance_rate`	percent	per month	Share of agent-drafted pull requests approved at first review, without a rewrite by the reviewer or a second round of changes.
`agent_detection_survival_90d`	percent	per quarter	Share of agent-drafted detections deployed 90 or more days ago that remain live and at or above their precision floor.

A high acceptance rate says the drafts are healthy only while 90-day survival stays high. Acceptance rising while survival falls means reviewers are waving noisy rules through. A third number helps when the first two disagree: the share of an agent's pull request lines the reviewer had to rewrite before merging.

10.6 The files in the repository

AGENTS.md sits at the root of the reference repository and on the site's downloads page. It gives any agent runner the lifecycle, the commands for each stage, and the boundaries above. The script tools/agents_md.py generates it from framework.json, and lint fails when the file is out of date, so the instructions an agent reads match the definitions of done CI enforces. CLAUDE.md points Claude Code at the same file. The pull request template carries the review checklist and a line that discloses agent-drafted work.

11. Operating rhythm

REMEMBER

- Run intake weekly, live review monthly, and coverage and capacity quarterly, and review detections within 5 working days of each incident.
- Every role needs an owner, and one person can own several.
- Check that the reviewer and SOC lead roles have owners. Gaps in those two take the longest to show.
- Spend the first 30 days on the inventory, detection zero, and measurement, before you write new detections.
- Report the incident detection rate even when it's low.
- A playbook gets a record, the eight stages, and 14 days in propose mode before it acts. A person approves every containment.
- Every incident ends in a candidate, a tuning pull request, or a log source request, and the report names the ones that ended in nothing.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

The framework says what good looks like. Three standing meetings and a review after every incident keep the backlog scored, noisy rules tuned or retired, and leadership informed. Running a program also takes an owner for every role and the templates on the Templates and cadence page.

11.1 Meetings

Meeting	Time box	Who is in it	What it decides	What it produces
Intake and order, weekly	30 minutes	Detection lead, the engineers, whoever brought a candidate	What enters build this week, and which candidates are blocked on telemetry	A re-ordered backlog with every new candidate scored and every score carrying its evidence lines
Live review, monthly	60 minutes	Detection lead, the SOC or triage lead, the engineer who owns each detection under discussion	What gets tuned with a named exclusion, what gets retired, what gets left alone	Tuning pull requests and retirement notes, each naming the trigger that fired and the number behind it
Coverage and capacity, quarterly	90 minutes	Detection lead, SOC lead, the platform or data owner, the manager who owns the budget	Next quarter's telemetry asks, whether the weights still match the threat picture, and which ceiling to spend against	The quarterly report: the numbers, the ask, the lowest maturity dimension with the work that raises it, and next quarter's OKRs
Detection review after an incident, within 5 working days of every incident	45 minutes	Incident responder who ran it, detection lead, the engineer who will build what comes out	Which live detections get a fix carrying the incident's events as a fixture, which candidates enter intake with what threat relevance, and which rules put live by hand during it enter intake	Intake items backed by your own incident with dates, the strongest evidence a candidate can have. A row in incidents.csv says who found the incident first and, when no live detection did, gives its miss code.

Each meeting prevents a specific failure:

Meeting	What happens when it is skipped
Intake and order	The loudest request wins, and blocked candidates sit in the queue looking like engineering work.
Live review	Nothing leaves production. Analysts absorb the cost until they stop reading alerts.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Meeting	What happens when it is skipped
Coverage and capacity	The program grows until a ceiling breaks it, and you find out from a missed incident.
Detection review after an incident	The most valuable intake source in the organization goes unread, and the same technique works twice.

None of these meetings needs a tool. The weekly takes half an hour. On a one-person team you spend it alone with a written backlog, which beats carrying the order in your head.

11.2 Roles

On a small team one person owns several roles, and section 3 says which ones collapse at each size.

Role	Owns	Decides	Stages it owns
Detection lead	Intake, the backlog order, the metric set, and the quarterly report	What is built next and what the weights are	intake, deploy, retire
Detection engineer	The hypothesis, the detection, the record, the fixtures, and the score's evidence lines	How the behavior is matched, and what the detection will not catch	hypothesis, research, build
Reviewer, a second engineer	The approval on the change, and the argument with the blind spots section	Whether it ships	validate
SOC or triage lead	The triage minutes, the alert outcomes, and half of the tuning decision	What the analysts can absorb, which sets the triage ceiling	tune
Incident response lead	Investigation, containment, and the notification path	When an alert becomes an incident, what gets contained, and who is told by when	none; receives what triage escalates, and sends every confirmed incident to the detection review
Platform or data owner	The log source inventory, the silence alerts, parsing and retention	What gets onboarded and when, which sets telemetry readiness	none; supplies readiness for research

What goes wrong when nobody owns a role:

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Role	What goes wrong with no holder
Detection lead	Priority becomes whoever asked most recently, and no number is reported the same way twice.
Detection engineer	Detections exist without records, and nobody but the author can triage or change them.
Reviewer, a second engineer	Every detection carries its author's assumptions about the environment into production.
SOC or triage lead	Run cost is a guess, the precision floor has no data behind it, and nobody notices alert fatigue until analysts leave.
Incident response lead	Confirmed incidents never reach detection engineering, so the same technique works twice, and reporting deadlines depend on memory.
Platform or data owner	Readiness scores are fiction, and detections are built on log sources that stopped last month.

Check the reviewer and the SOC lead first, because their absence takes the longest to notice. Without a reviewer, detections reach production built on an assumption the author didn't know they were making. Without a SOC lead, nobody measures run cost, and the backlog order becomes a guess.

11.3 The first 90 days

This is for you if you've just been handed the function, with or without a team. A team starting from nothing begins with section 4 instead.

Days 1 to 30: See what you have

- Inventory the log sources: source, owner, retention, parsed yes or no, normal event rate.
- Ship the silence alert for every source that matters. This is detection zero.
- List the live detections and find the ones nobody has looked at in a year.
- Pull the last 12 months of incidents and tag each with the technique the attacker used and with whether a live detection alerted on it first.

Report at the end of the block: The inventory, the live count, and the share of incidents a detection caught first.

Refuse for now: Don't write new detections yet, and don't promise coverage percentages.

Days 31 to 60: Put the work in order

- Score the open backlog on the six inputs, with an evidence line each.
- Put the definitions of done into your ticket template so a detection can't close on judgment alone.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

- Start the weekly intake meeting and the post-incident detection review.
- Measure precision and triage minutes for the 10 noisiest live detections.

Report at the end of the block: The ordered backlog, the first precision numbers, and the first retirement candidates.

Refuse for now: Don't buy a platform to fix a process problem, and don't adopt the CI pipeline before the records exist.

Days 61 to 90: Make it repeat without you

- Build the first three detections through every definition of done, records and fixtures included.
- Retire what the triggers say to retire, with a note each.
- Run the monthly live review for the first time.
- Score the maturity dimensions and publish the lowest one with the work that raises it.

Report at the end of the block: The quarterly report: coverage weighted by robustness, the three ceilings against the live count, the maturity score, and one ask.

Refuse for now: Don't report a detection count as progress, and don't commit to a coverage target before you know the telemetry ceiling.

11.4 Common failure patterns

None of these failures takes a bad engineer. Each one is the reasonable thing to do on a busy day, which is why they keep happening. The table says what each looks like, what it costs, and what replaces it.

Pattern	What it looks like, and why it happens	What it costs	What replaces it
The console edit	A detection is changed directly in the SIEM to quiet an alert during a shift. It takes 30 seconds, and nobody reviews it.	The change has no author, no reason, no test, and no way back. The next replay fails and nobody knows why.	Deployment from the repository only; console edits reverted or alarmed. Tune through an allowlist variable.
The coverage heatmap	A grid of ATT&CK techniques shaded green wherever a detection exists. It's the easiest artifact to produce and the easiest to present.	Seven brittle detections and one solid detection both show green, with no way to tell which is which.	Report coverage weighted by robustness beside the raw count, and report the gap between them.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Pattern	What it looks like, and why it happens	What it costs	What replaces it
The vendor pack	400 out-of-the-box detections enabled at once because they came with the platform. It looks like instant coverage and costs nothing up front.	Alert volume past the triage ceiling in a week, and no record, score, or owner for any of them.	Enable them against the alert budget, scored like anything else, in batches you can measure.
The unowned alert	An alert fires into a queue that nobody is accountable for reading. The detection shipped before the response steps were written.	Triage minutes are spent and no decision comes out of them.	Deploy only detections whose record has response steps.
The hero detection	One engineer's detection that only that engineer can interpret. The detection is clever and the record feels like overhead.	It goes unread on their leave and unmaintained after they move on.	11 sections, and a reviewer who has to triage from the record alone.
Tuning that becomes narrowing	Each exclusion shaves the detection until it matches only the original sample. Every individual exclusion is defensible in the moment.	A detection that still reports as live coverage and can no longer catch the technique.	Name the exclusion as a variable and count them. When a detection needs a new exclusion every month, propose its retirement at the monthly meeting.
The silent source	A log source stops and the detections on it keep reporting as healthy. Nothing alerts on absence, and dashboards only show what arrived.	Coverage is reported for a technique nobody could detect that month.	Detection zero, and readiness that drops automatically when a source dies.
The redefined metric	Precision means one thing this quarter and another the next. The number was inconvenient, and the definition was never written down.	Every trend line tracks the definition.	One written definition per metric, one owning stage, and a change to a definition treated as a release.
The god-mode emulation	The technique is run from an admin session because that session was already open. It's faster, and the commands succeed the first time.	The events carry an identity no attacker would have. Detections written from them fire in the lab and never in production.	Start from the foothold the scenario allows. Use only the permissions that stage would have.
The remembered field name	The detection is written from documentation or from memory of the schema. The engineer has read that log 100 times.	A detection that is almost right, never fires, and reports as coverage until an incident proves otherwise.	Read every field name off a real event from the emulation run.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Pattern	What it looks like, and why it happens	What it costs	What replaces it
The steered agent	An agent reading logs follows an instruction that an attacker planted in a command line or a user name. Nothing told the agent that log content is data.	The agent queries, writes, or opens pull requests for the attacker, with the program's own credentials.	Read-only credentials, tool calls taken only from the task, and every change through a pull request a person approves.

11.5 Miss codes

The detection review after an incident names the cause of every miss with one code. Counted over a quarter, the codes show leadership where misses come from and which owner fixes them.

Code	What happened	The fix	Owner
`no_log_source`	The behavior happened where no collected log source records it.	A log source request to the platform owner, and the technique's candidates rescored with telemetry readiness 0.	Platform or data owner
`source_silent`	The log source is collected, but it sent nothing while the incident ran.	A silence alert for the source in DET-0000, with the source's usual gap measured.	Platform or data owner
`field_unparsed`	Events arrived, but a field the detection needs was missing or unparsed.	A parsing fix, and a lower telemetry readiness for the source until it lands.	Platform or data owner
`no_detection`	The telemetry was there, and no live detection covered the behavior.	A candidate at intake, with the incident as its threat-relevance evidence and its events as the positive fixture.	Detection lead
`detection_broken`	A live detection covers the behavior and stayed silent, because of a wrong field, a changed format, or a query that drifted from its approved copy.	Fix this detection. The missed events become a positive fixture, and the change goes back through validate and its 7-day count.	Detection engineer
`tuned_out`	An exclusion added in tuning suppressed the alert.	Narrow the exclusion. The missed events become a positive fixture, so the next tuning change has to keep them.	Detection engineer and SOC lead

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Code	What happened	The fix	Owner
`alert_late`	A detection alerted, but after a person or another source had already found the incident.	Shorten the detection's latency through its schedule, its lookback, or its place in the queue.	Detection engineer
`alert_unworked`	The alert fired and nobody opened it in time.	Cut the load or raise the triage ceiling: retire noisy detections, or route this one to a faster queue.	SOC lead
`alert_closed`	A person or an agent opened the alert and closed it as benign.	Fix the record's response section, add the case to triage calibration, and count it in the false-close rate.	SOC lead

Record one code for each incident or exercise that a live detection did not catch first. When more than one applies, take the first break along the path from the log source to the analyst, because a fix further along cannot work until the earlier one lands. The repository keeps the record in incidents.csv, one row per confirmed incident or exercise. Lint checks every row against this list, and tools/metrics.py counts the codes per quarter for the reporting page. One row per confirmed incident or exercise. The found_by column says detection only when a live detection alerted first, and then miss_code stays empty. The fix column names the detection id or the log source request, and fix_validated_on is the date the fix passed validate.

11.6 Quarterly OKRs

Each quarter, the coverage and capacity review sets at most three objectives, each raising one maturity dimension by one level. Every objective carries two kinds of key result: the work that earns the next level, and a metric that moves because of that work.

- Raise at most three dimensions in a quarter, each by one level. A level counts only when every statement at it is true, so two levels in one quarter is rare.
- Start from the lowest dimension, because the program's level is its lowest dimension.
- Write each metric key result as a baseline and a target. With no baseline yet, the key result is to measure it this quarter.
- Use a KRI as a key result only when it is past its tolerance today, and set the target at the tolerance.
- Score last quarter's objectives at the same review, before setting the next ones.

The roadmap plays the same rule forward. Each quarter, the lowest dimensions below the year's target rise one level, up to three a quarter. Re-run it every quarter, because a quarter's real results change the next one. The Plan OKRs and a roadmap tool drafts both from a maturity score. Each maturity dimension names an owner and the metrics that show it moving:

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	Owner	Metrics that show it moving
Telemetry	Platform or data owner	`readiness_at_hypothesis`, `silent_source_hours`
Backlog and order	Detection lead	`intake_to_score_days`, `backlog_age_days`, `priority_technique_gaps`
The detection record	Detection engineers	`triage_minutes_median`, `minutes_per_true_positive`
Validation	Detection lead	`percent_with_tests`, `first_pass_validation_rate`, `review_turnaround_hours`
Deployment	Detection engineering, with the platform owner	`deploy_lead_time_days`, `post_deploy_replay_failures`, `detection_latency_median`
Upkeep and retirement	Detection lead and SOC lead	`precision_30d`, `upkeep_minutes_per_detection`, `rule_debt`, `rules_retired_per_quarter`
Measurement	Detection lead	`robustness_weighted_coverage`, `incident_detection_rate`, `misses_by_cause`, `open_detection_fixes_age`
Triage	SOC or triage lead	`triage_minutes_median`, `false_close_rate`
Investigation	Incident response lead	`time_to_scope_hours`, `misses_by_cause`
Containment	Incident response lead	`time_to_contain_minutes`
Communication	Incident response lead, with legal and the CISO's office	`notifications_on_time`

11.7 Regulatory requirements

A detection and response program already keeps most of the records a security regulation or an audit asks for: the logs it collects, the detections it reviews, the incidents it records, and the notifications it sends. The crosswalk names, for each control, the records that answer it, so an audit request becomes a list of files.

Answer each control with a record the program already keeps, and add a record only when none answers. An auditor, or counsel for a regulation, decides whether a record meets the control. A notification clock starts at the event its rule names, and the program records that time with the notification.

The crosswalk maps 59 controls from 9 regulations and standards. NIST's wording is quoted; the rest is paraphrased from the source, except where the last column says otherwise:

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Regulation or standard	Version	Controls	Read in
NIST Cybersecurity Framework 2.0	CSWP 29, 26 February 2024	16	the source
ISO/IEC 27001:2022, Annex A	2022	10	control titles only
PCI DSS	4.0.1, June 2024	10	the source
SOC 2 Trust Services Criteria	2017, points of focus revised 2022	6	a secondary copy
DORA, Regulation (EU) 2022/2554	Applies from 17 January 2025, with Delegated Regulation (EU) 2025/301 for report timing	3	the source
NIS2, Directive (EU) 2022/2555	Directive of 14 December 2022, applied through each member state's law	3	the source
US SEC cybersecurity disclosure rules	Form 8-K Item 1.05 and Regulation S-K Item 106	2	the source
HIPAA Security Rule and Breach Notification Rule	45 CFR Part 164 as in force; the 2025 proposed rule is not final	4	the source
NYDFS Cybersecurity Regulation, 23 NYCRR 500	As amended 1 November 2023	5	the source

Most controls ask for the same few records, so the program keeps each once:

Record	Where it is	Asked for by
Case records	the case system	NIST CSF, ISO 27001, PCI DSS, SOC 2, DORA, NIS2, SEC, HIPAA, NYDFS
Detection records	detections/*/ads.md and rule.yml	NIST CSF, ISO 27001, PCI DSS, SOC 2, DORA, NIS2, HIPAA, NYDFS
Notification log	notifications.csv	NIST CSF, ISO 27001, SOC 2, DORA, NIS2, SEC, HIPAA, NYDFS
Log source inventory	inventory.csv	NIST CSF, ISO 27001, PCI DSS, SOC 2, DORA, HIPAA, NYDFS
Incident log	incidents.csv	NIST CSF, ISO 27001, PCI DSS, SOC 2, DORA, SEC, HIPAA
Detection zero	detections/DET-0000/	NIST CSF, ISO 27001, PCI DSS, DORA, HIPAA, NYDFS

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Record	Where it is	Asked for by
Alert dispositions	the case system	NIST CSF, ISO 27001, PCI DSS, SOC 2, NIS2, HIPAA
Validation replays	detections/*/tests/ and the CI log	ISO 27001, PCI DSS, SOC 2, DORA, NIS2
Emulation runs	the Technical Context section of each ads.md	NIST CSF, PCI DSS, SOC 2, DORA, NYDFS
Metrics report	site-data/metrics.json, from tools/metrics.py	NIST CSF, SOC 2, DORA, NIS2, SEC
Precision checks	the 30-day precision review, which a person runs today	ISO 27001, SOC 2, NIS2
Maturity score and OKRs	the maturity and OKR tools	NIST CSF, NIS2, SEC
Backlog scores	detections/*/score.yml	NIST CSF, ISO 27001
Review approvals	the pull request history	ISO 27001, SOC 2
Retirement notes	the Retirement record section of each ads.md	PCI DSS

The reporting clocks run from an event the regulation names. Each notice goes in notifications.csv with its clock and the time the clock started. Lint computes each deadline and fails a hand-typed one that disagrees, and metrics.py reports notifications_on_time each quarter. A late notice to a regulator or a customer turns that KRI red. Business days count weekdays and skip no holidays, so the computed deadline is never later than the real one. National law sets the details of NIS2 reporting, and counsel decides which clocks apply to an incident.

Report	Due	Clock
DORA initial notification	4 hours after the incident is classified as major	`dora_initial`
DORA initial notification, outer limit	24 hours after the entity becomes aware of the incident	`dora_initial_awareness`
DORA intermediate report	72 hours after the initial notification is sent	`dora_intermediate`
DORA final report	1 month after the latest intermediate report is sent	`dora_final`

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Report	Due	Clock
NIS2 early warning	24 hours after the entity becomes aware of a significant incident	`nis2_early_warning`
NIS2 incident notification	72 hours after the entity becomes aware of a significant incident	`nis2_notification`
NIS2 incident notification by a trust service provider	24 hours after the provider becomes aware of a significant incident	`nis2_notification_trust`
NIS2 final report	1 month after the incident notification is sent	`nis2_final`
Form 8-K Item 1.05	4 business days after the company determines the incident is material	`sec_8k`
Form 8-K amendment with information missing from the first filing	4 business days after the missing information is determined or becomes available	`sec_8k_amendment`
HIPAA notice to affected individuals	60 days after the breach is discovered	`hipaa_individuals`
NYDFS notice to the superintendent	72 hours after the company determines a cybersecurity incident occurred	`nydfs_notice`
NYDFS notice of an extortion payment	24 hours after the extortion payment is made	`nydfs_extortion`
NYDFS reasons for an extortion payment	30 days after the extortion payment is made	`nydfs_extortion_reasons`

Some clocks bend in cases the table can't show:

- `dora_initial`: When a deadline falls on a weekend or bank holiday, Delegated Regulation 2025/301 moves it to noon of the next working day for most entities. The tools do not, so their deadline can only come earlier.
- `dora_initial_awareness`: Log this clock beside dora_initial only when the incident was classified major within 24 hours of awareness. A later classification leaves dora_initial alone, at 4 hours.
- `nis2_notification`: Trust service providers have 24 hours, on the nis2_notification_trust clock.
- `nis2_final`: For an incident still being handled at the one-month mark, a progress report is due then, and the final report one month after the handling ends.
- `sec_8k`: The materiality determination itself must come without unreasonable delay after discovery.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

The kit's audit_pack.py reads the records the repository keeps and writes an index for the regulations named, with each record found, empty, or missing. It lists the records kept elsewhere, case records above all, for a person to export. The Map regulatory controls tool shows the same crosswalk in a browser.

11.8 Response playbooks

A playbook is the automation that runs when a detection fires: it enriches the alert, decides, and acts or proposes an action. It gets the same treatment as a detection. It has a record, it moves through the eight stages with a countable definition of done at each, it runs in propose mode before it acts, and a trigger retires it. A playbook that only exists in the SOAR console is a live detection with no record.

Playbooks serve detections, so a playbook names the detection ids or the alert class it runs on, and it enters intake only when that detection is live. The agent rules of section 10 cover a playbook that calls a model; the approval points below apply either way.

The record lives at playbooks/PB-NNNN/playbook.md and carries these sections:

Section	What it holds
Serves	The detection ids or the alert class the playbook runs on, and the alert volume per day it will see.
Trigger	The alert field values that start it. Nothing else does, so an alert the trigger does not name goes to a person.
Enrichment	Each lookup, its source, and the field it fills. A lookup that fails leaves its field empty and says so, and the playbook goes on.
Decision	The rule that turns the enriched alert into one outcome: close as benign, escalate, or act. Every branch is written down, and the default branch is escalate.
Action	What it does on each outcome, at which tier of the ladder, and the exact call it makes.
Approval point	Which actions wait for a person, who that person is, and how long the action waits before it expires unacted.
Rollback	The call that undoes each action, tested once against a real object, and who may run it.
Manual baseline	The steps an analyst took by hand before the playbook, timed over at least 20 alerts: median minutes, and the outcome each one got.
Fixtures	One past alert per outcome, with the enrichment results as they were, and the outcome the analyst recorded.
What it gets wrong	The alert the decision rule will misread, with its name. A playbook that closes alerts has at least one.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

What a playbook may do comes in four tiers. Each tier up changes more outside the alert, and each needs more of a person:

Tier	What the playbook does	Who approves
1. Enrich	Adds context to the alert: asset owner, user's role, prior alerts, reputation, the detection's record.	None. Nothing changes outside the alert.
2. Route	Sends the alert to a queue, a person, or a ticket, and sets its severity from the enrichment.	None, once the routing rule has run in propose mode.
3. Close	Closes an alert as benign with a code, and records why.	No person for a detection rated impact 1 to 3 once agreement is above the floor. A person every time for impact 4 or 5.
4. Contain	Changes something outside the alert: disables an account, isolates a host, blocks an address, revokes a token.	A person, always, and the action expires unacted after the time the record sets. A reversible action on an asset the record names as expendable, such as a test host, may skip the approval.

A playbook moves through the same eight stages as a detection. Where a stage's definition of done differs, this table says how; the rest is Appendix A:

Stage	What differs for a playbook	Definition of done, counted
1. Intake	The candidate names the detection or alert class it serves, the alerts per day it would see, and the ladder tier it wants. Scored on the six inputs like a detection, where build effort covers the integrations and run cost covers the actions it would take.	6 scores with an evidence line each, 1 detection id or alert class, 1 tier
2. Hypothesis	One sentence: on this alert, with these lookups, this rule gives this outcome, and it will misread this case.	1 sentence, 1 named outcome, 1 named misread
3. Research	An analyst works at least 20 of the alerts by hand and times each one. The steps they took become the enrichment list and the decision rule; the minutes become the manual baseline.	20 alerts timed, each with its steps and its outcome
4. Build	The playbook is written as code in the repository, where it can be reviewed and replayed. Each lookup has a fixture, each outcome has a fixture, and the rollback runs once against a real object.	1 fixture per outcome, 1 fixture per lookup, 1 rollback run

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	What differs for a playbook	Definition of done, counted
5. Validate	Replayed against the last 30 days of closed alerts for the alerts it serves. Its outcome agrees with the analyst's recorded outcome at or above the floor, it proposes no close on an alert later confirmed malicious, and a second person approves.	30 days of alerts replayed, agreement at or above the floor, 0 closes on a confirmed alert, 1 approval by someone other than the author
6. Deploy	Runs for 14 days in propose mode: it enriches, decides, and writes what it would do, and a person does it. Acting switches on only if agreement stayed above the floor for the whole 14 days and no proposed action was refused on an alert rated impact 4 or 5.	14 days in propose mode, agreement at or above the floor on every day, 0 refused proposals on an impact 4 or 5 alert
7. Tune	Every refused or reverted action is a tuning pull request that changes the decision rule and adds the alert as a fixture. The playbook stays in propose mode until it merges.	1 pull request per refused or reverted action, with the alert as a fixture
8. Retire	A trigger fires, the owner decides, and the record keeps a retirement note. The playbook's actions switch off before its code is removed.	1 trigger named, 1 retirement note, actions off before removal

The agreement floor is 0.95. The share of alerts on which the playbook's outcome matched the analyst's. The default is a chosen starting point that no team has measured yet. Set it lower for tier 1 and 2 playbooks, which cost a person a minute when wrong, and never below 0.99 for a playbook that closes alerts on its own.

Four triggers propose a playbook's retirement, and the owner decides, as with a detection:

- `agreement`: Agreement under the floor for two months running. Back to propose mode first; retirement if the tuning pull requests do not raise it.
- `orphaned`: Every detection it serves is retired. The playbook retires with the last one.
- `idle`: No run in 90 days. Either the alert stopped, which is a detection question, or the trigger no longer matches, which is a playbook bug. Both end at retire or tune.
- `reverted`: A tier 4 action reverted by a person. The playbook drops to propose mode that day, and the live review decides whether it comes back.

Four metrics, defined in Appendix C with the rest. The kit does not compute them yet; the SOAR's own run log and the alert outcomes supply the counts.

Metric	Unit	Aggregation	Definition
`playbook_coverage`	percent	per quarter	Share of alerts in the quarter whose detection has a live playbook at tier 1 or higher.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Unit	Aggregation	Definition
`playbook_minutes_saved_per_alert`	minutes	median per quarter	The manual baseline minutes on the playbook's record, minus the analyst minutes an alert takes with the playbook running, per alert class.
`playbook_actions_unattended`	count	per month, by tier	Actions a playbook completed with no person in the loop, counted by ladder tier. A tier 4 count above zero names the expendable asset the record allows.
`playbook_actions_reverted`	count	per quarter	Playbook actions a person refused in propose mode or undid after they ran. Each one is a tuning pull request.

11.9 After an incident, and who carries the pager

Every confirmed incident changes the backlog or a detection. The detection review names what it changes, and the quarterly report counts the incidents that changed nothing.

- Every confirmed incident gets a row in incidents.csv within 5 working days, with found_by filled in and, when no live detection caught it first, one miss code.
- Every incident produces at least one of: an intake candidate carrying the incident's events as its positive fixture, a tuning pull request on the detection that fired late or stayed silent, or a log source request to the platform owner. The row's fix column names it.
- An incident that produced none of these within 30 days counts in incidents_without_followup. The quarterly report lists each one with the responder who ran it.
- A rule put live by hand during the incident enters intake the same week, with the incident as its evidence. It stays live until its candidate passes validate or the live review retires it.
- The responder brings the timeline; the detection lead brings the alert record. Neither writes the other's part.

The SOC's rotation answers alerts. A second, lighter rotation answers for the detection pipeline itself. A silent log source or a red build on main is not an alert, so nobody is paged for it unless someone is named.

What pages	Who answers	Within
A DET-0000 silence alert on a log source rated readiness 3 or higher	The platform or data owner, with the detection engineer on rotation in copy	The source's usual gap, as DET-0000 measures it

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

What pages	Who answers	Within
A red build on the default branch, or a conversion that no longer matches its approved copy	The detection engineer on rotation	The next working day; nothing deploys until it is green
A live detection whose alert count passes its hourly limit	The detection engineer on rotation, with the SOC lead	One hour, and alerting switches off on its own before anyone answers
A tier 4 playbook action waiting for approval	The incident response lead or the named approver	The time the playbook's record sets; the action expires if nobody answers

One engineer carries the pipeline pager for a week at a time. A one-person team carries it always, which is one reason the silence alert's floor is an hour and not a minute.

12. Handling objections

Bring this into a working program and people will push back, usually in the first meeting. These are the arguments I'd expect, with my answer to each. Some of them are at least partly right, and for those my answer is to agree.

The objection	Usually from	The answer
"Our SIEM ships hundreds of rules. Why write our own?"	The platform owner, usually in month one	Keep them. Nobody is asking you to rewrite a vendor's work out of pride. Score the vendor pack like anything else, enable it in batches you can measure against the alert budget, and retire what doesn't clear its floor. Switch on a whole pack at once and you've made a coverage claim nobody can check and signed your analysts up for alert volume nobody agreed to.
"Writing a record for every detection is overhead we cannot afford."	The engineer who is already behind	For detections already live, start with four of the 11 sections: goal, categorization, blind spots, and response. That takes 30 minutes a detection, and most of it is remembering what you already know. Lint wants every section written from validate on, so keep these short records outside the kit's detections folder until the other seven are written. For new detections, you write the rest as they move through the stages. Skipping the record only moves the cost. The next person re-derives the intent from the query, and the responder at 3 a.m. has no response steps and a strong opinion about whoever wrote the rule.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

The objection	Usually from	The answer
"Our MSSP or vendor manages detections for us."	The manager who signed the contract	Then the definitions of done and the metrics become contract terms. Ask for the record, the fixtures, the precision per detection, and the retirement list. A provider who can't produce precision per detection is selling you alert volume with no measure of quality. Your analysts still spend the alert budget, whoever wrote the rules.
"We already report ATT&CK coverage. This duplicates it."	Whoever built the heatmap	Keep reporting it and add the weighted number beside it. If the two are close, the heatmap was honest and you've lost nothing. If they're far apart, the gap is the finding, and it's much nicer to find it yourself than to have a post-incident review find it for you.
"Scoring is false precision. Six numbers do not capture the judgment."	The most experienced engineer in the room, and they have a point	The score records the judgment, so somebody other than its author can inspect it. The value is in the evidence line under each number, and that's what a reviewer argues with. The order treats differences under 0.2 as ties for the same reason. I picked that width as a default, because nobody has measured how far two scorers disagree, me included.
"We cannot retire detections. Audit will ask why coverage dropped."	The compliance lead	A retirement with a note is a stronger audit position than a detection nobody has looked at in a year. The note says which trigger fired, what you measured, and what replaced it. If an auditor is going to object to something, it should be counting a detection that fired for 180 days and was never right as coverage.
"This is too heavy for a team of two."	The team of two	Then take the four practices section 3 gives a team of two to four: the scored backlog, the definitions of done, the detection record, and the retirement triggers. Defer the full metric set, but keep precision and triage minutes per detection, because the retirement triggers need them. Keep the review too. One of you builds and the other reviews, so neither of you approves your own detection.
"Our detections are in the SIEM console. Moving to code is a six-month project."	Anyone who has tried it	Section 3 puts detection as code last for teams under five engineers. Records, scores, definitions of done, and retirement all work with the console. What code adds is a build that enforces the definitions of done, including on the day a deadline makes skipping them look reasonable.
"Nobody will attend four more meetings."	Everybody	I don't like meetings either. The four take about three hours and 40 minutes a month for a small team: three standing ones, plus 45 minutes after each incident. The three standing ones replace something you're already doing badly: an ad-hoc request queue, a tuning conversation in chat, a quarterly slide nobody can source. The post-incident review is the only new one, and it produces the intake items with the strongest evidence.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

The objection	Usually from	The answer
"Our telemetry is a mess. We should fix that first."	The person who is right	I agree, and the framework makes that answer visible. Every candidate gets a readiness score, a zero blocks the build, and the backlog then shows which log source unblocks the most value. That list is the telemetry business case, and you get it as a side effect of doing the detection work.

12.1 Objections that are partly or wholly right

The objection	What it gets right
"Writing a record for every detection is overhead we cannot afford."	That's true for a program of 10 detections that one person owns and will keep owning. It stops being true the first time that person takes a vacation.
"Scoring is false precision. Six numbers do not capture the judgment."	The objection is right about precision. A score of 2.40 and a score of 2.52 are the same score, which the band rule states and the sort implements.
"Our detections are in the SIEM console. Moving to code is a six-month project."	Migrating is a project with its own plan and budget. Do it when hand-converting queries becomes the bottleneck, around 200 live detections.
"Our telemetry is a mess. We should fix that first."	Telemetry work outranks detection work whenever telemetry readiness blocks more than about a third of the scored backlog.

13. Maturity levels

> **REMEMBER**
>
> - Claim a level only for what is true today at that level and every level below it. A pilot counts for nothing.
> - Your overall level is your lowest dimension, never the average.
> - Use evidence someone outside the team could check.
> - Work on the lowest dimension first. The Score your maturity tool lists the dimensions in that order.

The maturity model scores a program at five levels on each dimension: seven for detection, and four for response that a team scores only if it owns response. Your overall level is your lowest scored dimension and never the average, so a strong pipeline can't make up for a missing inventory.

Every cell is an evidence test somebody outside the team could run, so a level is a claim an auditor can check and a self-assessment can't inflate.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Level	Name	What is true at this level
0	Ad hoc	Detections exist. Nothing about them is written down in a place a second person can find.
1	Repeatable	Every detection has a record and a defined way in. Two people would build the next one the same way.
2	Measured	The program counts its own work: definitions of done carry numbers, the backlog carries scores, alerts carry outcomes.
3	Enforced	Machines enforce the definitions of done. A skipped step fails the build.
4	Self-correcting	Measurement drives change: detections retire themselves on evidence and coverage claims can be falsified.

Score a level only if every statement at that level and below is true today. A pilot earns no credit. The gap between your lowest dimension and your highest is the work queue.

13.1 The dimensions

Detection

Detection dimensions cover how the program finds attacks: what it collects, what it builds, and how it keeps its detections honest.

Dimension	The question it answers	0. Ad hoc	1. Repeatable	2. Measured	3. Enforced	4. Self-correcting
Telemetry	Do you know what you collect, and do you know within a day when a source stops?	No inventory. A source can stop and nobody finds out until a detection is needed.	An inventory exists with an owner per source, refreshed at a stated interval.	Every source has a measured normal rate, and readiness is scored 0 to 5 on the backlog.	The silence alert covers every source in the inventory and fires to a real destination.	New detections cannot reference an uninventoried source, and readiness drops automatically when a log source stops.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	The question it answers	0. Ad hoc	1. Repeatable	2. Measured	3. Enforced	4. Self-correcting
Backlog and order	How is the order of detection work decided, and could you defend it to a skeptic?	Order is whoever asked most recently, or whoever asked loudest.	A single written backlog exists, ordered by hand.	Every candidate carries six scores, six evidence lines, and a computed score.	Scores are computed by the pipeline; a hand-typed score fails the build. Mandates carry their framework, requirement, and due date.	The backlog re-scores on a schedule and the order changes without anyone arguing about it.
The detection record	What does a responder have in front of them at three in the morning?	The query, and whatever the author remembers.	Each detection names its goal and its technique.	Records carry goal, technique, log sources, and response steps.	Records carry all 11 sections, the 10 ADS sections plus the retirement record, and the pipeline rejects a detection missing any of them.	Records are versioned with the detection, and every change to the logic changes the record in the same commit.
Validation	What proves a detection fires on the behavior, and that somebody other than the author agreed?	It went live because it looked right in the console.	Each detection is spot-checked by hand against one true event.	Every detection has a true-positive fixture and a false-positive fixture stored with it.	Fixtures replay in CI on every change, and a second engineer's approval is required to merge.	Fixtures and golden query diffs run for every target, and a failing replay blocks the release.
Deployment	How does a detection get from a text editor into production, and can you replay how?	Edited directly in the SIEM console.	Detections live in version control and are copied into the SIEM by hand.	Deployment is a documented, repeatable procedure with a change record.	The pipeline deploys; console edits are reverted or alarmed.	Every release is reproducible from a commit, and a rollback is one revert.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	The question it answers	0. Ad hoc	1. Repeatable	2. Measured	3. Enforced	4. Self-correcting
Upkeep and retirement	What removes a detection, and how long does a dead one keep costing you?	Nothing. Detections accumulate.	Detections are reviewed when somebody complains about the noise.	Precision and triage minutes are measured per detection every month.	Each detection has a floor, and a detection under its floor for two windows is proposed for retirement on the record.	The precision check runs on a schedule and proposes retirements; replacements enter the backlog with a link to the retirement note.
Measurement	What numbers does the program report, and would two people compute them the same way?	Detection count, when asked.	A handful of numbers are reported without written definitions.	Every reported number has one written definition and one owning stage.	The pipeline produces the build-side numbers, and coverage is reported weighted.	Numbers change decisions on the record, and a claim about coverage can be falsified by pointing at a test.

Response

Response dimensions cover what happens after an alert fires. Score them only if the team owns response. A team that hands confirmed incidents to another group leaves them out, and its level is its lowest detection dimension.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	The question it answers	0. Ad hoc	1. Repeatable	2. Measured	3. Enforced	4. Self-correcting
Triage	Who works each alert, how long does it take, and would two analysts close it the same way?	Alerts are worked by whoever sees them, and nothing records how they closed.	Every alert goes to one queue, and each closed alert carries an outcome.	Every alert closes with a code from a fixed list, and triage minutes and precision are measured per detection.	Analysts calibrate each month on the same sampled alerts, and an alert closed as benign that proves malicious gets the miss code alert_closed.	A weekly sample of closed alerts measures the false-close rate, and each wrong close changes a record or a runbook.
Investigation	When an alert is real, how fast does someone establish what happened, and could a second person repeat the work?	Investigations happen in chat and in people's heads.	Every escalated alert opens a case with an owner and a timeline.	The most frequent alert types have playbooks, and the hours from escalation to a known scope are measured.	Each case records its queries and their time ranges, and a second person can repeat the investigation from the case alone.	Every confirmed incident reaches the detection review, and its events become fixtures or intake items with a miss code.
Containment	Once something is confirmed, how fast is it stopped, and who is allowed to stop it?	Containment depends on who is online and what they can reach.	The containment actions, and who may take each one, are written down.	The common actions are scripted, and the minutes from confirmation to containment are measured.	Actions that can be undone are approved in advance in playbooks, and each one is logged with who took it.	Every containment action is tested each quarter, and a failed test is fixed before an incident needs it.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	The question it answers	0. Ad hoc	1. Repeatable	2. Measured	3. Enforced	4. Self-correcting
Communication	Who hears about an incident, how soon, and does the program meet its reporting deadlines?	Leadership and regulators hear about an incident when someone remembers to tell them.	A notification list exists for each severity, with every deadline that applies.	Every notification is timestamped and measured against its deadline.	The notification path is rehearsed each year, and a late notification is reported with its cause.	Reports are drafted from the case record, and no notification missed its deadline in the last year.

13.2 Reading your own score

Score each dimension on what's true today, then take the lowest. A program at Enforced on deployment and Ad hoc on telemetry is an Ad hoc program with good deployment. Its coverage numbers may describe detections reading a log source that stopped last month.

Raise your lowest dimension first. Each dimension names the next action from where you stand, and the site's Score your maturity tool prints that list for the levels you select.

Dimension	To reach Measured	To reach Self-correcting
Telemetry	Record whether each source's fields are parsed, and its normal event rate.	Fail the pipeline when a detection cites a source that is not in the inventory.
Backlog and order	Score each candidate on the six inputs, with one evidence line per input.	Re-score the standing backlog on a schedule, so stale relevance falls away.
The detection record	Add the log sources it reads and the response steps for the analyst.	Version each record with its detection, so a change to the logic changes the record in the same commit.
Validation	Capture that event as a fixture stored next to the detection.	Diff the converted query for each target against an approved copy, so a conversion change cannot pass silently.
Deployment	Write the deployment step down so two people do it the same way.	Make every release reproducible from a commit, with a rollback that is one revert.
Upkeep and retirement	Measure precision and triage minutes per detection over 30 days.	Automate the check so the report arrives without anyone running it.
Measurement	Write one definition per metric, with its unit and how it aggregates.	Tie the numbers to decisions: what gets built next, what gets retired.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Dimension	To reach Measured	To reach Self-correcting
Triage	Close every alert with a code from a fixed list, and measure triage minutes per alert.	Sample a share of closed alerts every week, and send each wrong close back to the detection's record.
Investigation	Write a playbook for each of the most frequent alert types, naming the queries to run.	Send every confirmed incident's events to the detection review, as fixtures and intake items.
Containment	Script the common actions, and measure the minutes from confirmation to containment.	Test each containment action every quarter against a test host or account.
Communication	Record when each notification went out, and measure it against its deadline.	Draft the regulator reports from the case record, so the timeline is written before the clock starts.

Each detection dimension maps to one of the practices in section 3, so your level also tells you which practice to adopt or deepen next. Countable definitions of done is the one practice without a dimension of its own. I left it out because a definition of done is how every other dimension moves from Repeatable to Enforced.

14. Scaling

> **REMEMBER**
>
> - Analyst triage time, engineering upkeep time, and telemetry each cap a program, and the smallest cap sets your size.
> - Each live detection costs about 20 minutes a month to maintain, on top of its triage minutes.
> - Replace the 20-minute default with your own number after a quarter of measuring.
> - Tuning and retirement raise the triage ceiling for less than hiring analysts does.
> - A candidate blocked on telemetry needs a log source request before it needs an engineer.

Triage minutes cap how many alerts analysts can absorb, engineering minutes cap how many detections stay current, and telemetry readiness caps what you can build at all. The smallest of the three sets the size of your program.

14.1 The ceilings

Ceiling	What it limits	How it is computed
Triage ceiling	How many live detections can the analysts absorb?	`(analysts x triage hours each per day x 60) / (alerts per detection per day x minutes per alert)`

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Ceiling	What it limits	How it is computed
Maintenance ceiling	How many live detections can the engineers keep current?	`(engineers x upkeep hours each per week x 60 x 52 / 12) / upkeep minutes per detection per month`
Telemetry ceiling	How many detections can be built at all?	`count of techniques whose log sources are all at readiness 3 or above; one detection per technique until you have measured your own ratio`

I count both recurring ceilings in minutes, because minutes are what run out. The maintenance figure defaults to 20 minutes per live detection per month. That covers the precision check, one look at the tuning queue, and fixing the query after a field changes. Replace it once you've measured a quarter.

Triage ceiling

- Cut alerts per detection: tighten the logic, exclude one named source of noise, or raise the threshold.
- Cut minutes per alert: enrich the alert with the context the analyst looks up by hand.
- Group related alerts into one case, so 10 alerts cost one triage.

Common mistake: Hiring analysts is the most expensive way to raise this ceiling.

Maintenance ceiling

- Generate the query for every target from one source rule.
- Let the precision check propose retirements, so upkeep shrinks the estate.
- Template the detections that differ only by a field or a threshold.

Common mistake: Nobody books the recurring cost when the detection ships, so this ceiling goes unnoticed until upkeep falls behind.

Telemetry ceiling

- Onboard first the source that unblocks the most scored candidates.
- Fix sampling, truncation, and short retention. That moves a source from readiness 2 to 3, where this ceiling starts counting it; parsing the fields then takes it to 4.
- Ship the silence alert, so a source that dies doesn't take its detections with it.

Common mistake: Treating a blocked candidate as engineering work keeps it in the detection backlog. It belongs on the log source request list until someone onboards the data.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

14.2 Breaking points by size

Live detections	What still works	What breaks first	The change that reaches the next order of magnitude
About 10	A spreadsheet, a shared folder, and one person's memory.	Nobody can say why a detection exists once its author is on leave.	Write the record: goal, technique, blind spots, response.
About 50	Records, a written backlog, manual deployment.	Alert volume becomes a rostering problem, and tuning changes lose their history.	Put the definitions of done in CI and start measuring precision per detection.
About 200	CI, scores, a metric set.	No one person knows the estate. Near-duplicates appear, and hand-converting queries per platform stops scaling.	One source rule per detection, converted to every target by the pipeline; ownership assigned per platform.
About 1000	Generated queries, owners, automated checks.	Coverage claims become unfalsifiable and the maintenance ceiling binds before the triage one.	Automate retirement, decouple alert volume from detection count by grouping into cases, and report coverage weighted by robustness.

At every step, coordination between people fails before the detections do. Each move swaps something one person remembers for a file, a definition of done, or a generated artifact.

14.3 Work that doesn't scale

- Counting detections. 200 detections that match filenames add up to the same weighted coverage as 40 at robustness 5, and each of the 200 still costs upkeep every month.
- One query per platform maintained by hand. Every new target multiplies the estate by the number of SIEMs.
- Triage headcount. Alert volume grows with the estate; analyst hours grow with the budget cycle.
- Review by the author. It's the first check dropped under deadline.

Review is the cheapest of these to keep. A second engineer's review costs 20 minutes, and it's the only check that catches an assumption the author didn't know they were making.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

15. Metrics

REMEMBER

- Define each metric once, and treat a change to a definition as a release.
- Report robustness-weighted coverage beside the raw detection count.
- Measure precision for each detection against that detection's own floor.
- The incident detection rate is the share of confirmed incidents a live detection caught first.
- Ingestion delay and the schedule set the floor under detection latency.

The program measures its health with 31 metrics. Each has one definition and one owning stage, except the 15 that belong to the program as a whole: robustness_weighted_coverage, incident_detection_rate, silent_source_hours, priority_technique_gaps, misses_by_cause, open_detection_fixes_age, false_close_rate, time_to_scope_hours, time_to_contain_minutes, notifications_on_time, playbook_coverage, playbook_minutes_saved_per_alert, playbook_actions_unattended, playbook_actions_reverted, and incidents_without_followup. The agent metrics in section 10 sit outside this set. The repository produces the build-side ones from score.yml and the file layout. The pull request system and the backlog's entry dates supply the durations. The rest come from what analysts record about each alert and the engineering hours logged on each pull request. The definitions are in Appendix C.

The Check coverage and alert load tool opens on a chart that puts the raw detection count beside robustness-weighted coverage. On a 30-detection sample program run through the repository's own metrics code, Execution has the most detections and Credential Access has the most weighted coverage. A detection mapped to more than one tactic is counted under each, so the detection column adds up past 30, and the sample includes two ICS tactics, Evasion and Impair Process Control:

Tactic	Live detections	Robustness-weighted
Execution	7	3.2
Persistence	5	3.0
Privilege Escalation	5	3.0
Credential Access	4	3.4
Stealth	3	1.6
Command and Control	3	0.8
Defense Impairment	2	1.4
Impact	2	1.4

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Tactic	Live detections	Robustness-weighted
Lateral Movement	2	1.4
Initial Access	2	1.2
Discovery	2	1.0
Exfiltration	2	0.8
Evasion	1	1.0
Impair Process Control	1	1.0

The coverage numbers answer different questions, so report the one that answers the question asked:

Number	The question it answers	How it is computed	Where it tops out
Detection count	How many live detections name a technique in this tactic?	One per live detection, counted under every tactic it maps to.	Never. A brittle detection adds to it as much as a strong one.
Robustness-weighted coverage	How much evasion-resistant work sits in this tactic?	Each live detection counts as its robustness divided by 5, summed per tactic.	Never. Five detections at robustness 1 equal one at robustness 5.
Weighted share	What share of the matrix is covered, and how well?	Each technique scores its strongest live detection's robustness divided by 5, averaged across every technique.	100%, when every technique has a detection at robustness 5.
Covered count	How many techniques are covered at all?	A technique counts once its strongest live detection is at robustness 4 or 5.	The technique count of the pinned ATT&CK release.

Both per-tactic numbers keep growing, which is what makes a bare detection count easy to inflate. Both per-technique numbers cap at the matrix, so a weighted share of 40% is a figure a board can compare quarter to quarter. The ATT&CK coverage workbook lists every Enterprise ATT&CK 20.1 technique and sub-technique and computes all four. The site's coverage app runs the same rules in your browser, keeps your team's data on your own machine, and exports to Excel, PNG, or PDF. It also reads the log sources each of ATT&CK's analytics names. A technique counts as ready to build only when one analytic has every log source it reads collected. From that the app reports which uncovered techniques you could

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

build today, which have some of the telemetry, which are blind, and which log source would make the most of them ready. For every technique it also says how many of the detection library's open-source rules are tagged with it, and links to them.

15.1 Checks from outside

Four of the site's claims are checked against something other than its author, and the numbers are republished with each run. MITRE's own ATT&CK library reads the same STIX bundle the site parses; on 2026-11-06 the two agreed on every technique, platform, analytic, and data component in ATT&CK 20.1. D3FEND's own API returned the same count of defenses as the coverage map for 5 of the 5 techniques checked on 2026-10-20. The library's robustness draft is run against the 72 SigmaHQ rules that MITRE's CTID scored by hand for Summiting the Pyramid. It matches the hand score on 55 and stays at or under it on 66. The library's Windows Sigma rules run against 107 recorded attacks from OTRF's Security-Datasets; of the 1,695 rules with a recording of their technique, 139 fired on it. The site's evidence page lists what each check shows, what it doesn't, and which claims remain opinion: the score model, the lifecycle's numbers, the maturity levels, and the ceilings.

15.2 Performance and risk

A KPI says how well the program does its work, and the team sets a target for it. A KRI says how exposed the program is, and the team sets a tolerance for it: the level at which someone has to act. The tolerances here are defaults to replace once a quarter of the team's own numbers exists. Of the 31 metrics, 10 are KRIs:

KRI	Amber	Red
`detection_latency_median`	over 15 minutes	over 60 minutes
`rule_debt`	over 10%	over 20%
`silent_source_hours`	over 4 hours for a source behind a live detection	over 24 hours
`priority_technique_gaps`	over 20%	over 40%
`misses_by_cause`	the same code twice in a quarter	the same code in two quarters running
`open_detection_fixes_age`	over 30 days	over 60 days
`false_close_rate`	any confirmed wrong close in a quarter	a wrong close on a detection rated impact 4 or 5
`notifications_on_time`	any internal notification late	any regulator or customer deadline missed

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

KRI	Amber	Red
`playbook_actions_reverted`	any reverted action in a quarter	a reverted tier 4 action, or a refused close on an impact 4 or 5 alert
`incidents_without_followup`	any incident in a quarter	the same responder twice, or any incident in two quarters running

Section 19 adds `ai_false_close_rate` as a KRI for a program that lets an agent close alerts. The quarterly report lists every KRI against its tolerance, red ones first.

15.3 ATLAS and D3FEND

ATT&CK covers attacks on hosts, networks, identities, and cloud services, and two other MITRE knowledge bases extend it. ATLAS 2026.09 lists 120 techniques and 88 sub-techniques against AI systems, and 44 of the 208 are adopted from ATT&CK. ATLAS names no log sources, so the coverage map can't score its techniques. A score.yml accepts an ATLAS id beside an ATT&CK one, and the kit checks both. ATLAS ties 48 of its techniques to a mitigation that, by this site's reading, detects an attack in progress, most often AI telemetry logging, which records a deployed model's inputs and outputs. Collect that log first, because a detection for prompt injection or for exfiltration through an inference API has nothing to read without it. ATT&CK 20.0 added direct and indirect prompt injection too, and a detection for a technique both list carries both ids.

D3FEND 1.6.0 links 387 current Enterprise and ICS techniques to the defenses that counter them, through an artifact both touch, such as a process, a file, a network flow, or a credential. Of the 156 defensive techniques it links, 56 detect. The coverage map lists those for each technique by the artifact they analyze, so a hypothesis can start from a named analysis such as process spawn analysis. D3FEND 1.6.0 maps to ATT&CK 19.0. The extract moves the 22 ids ATT&CK has since revoked to their replacements, and the techniques ATT&CK added after 19.0 have no D3FEND links yet.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

16. Detection as code

REMEMBER

- If the live rule and the repository disagree, the repository no longer tells you what runs. Alarm on the difference.
- Write the logic once in Sigma and let the build produce the query for each target. A hand-edited query can never be regenerated.
- Commit the converted query for every target. The diff is what catches a converter upgrade changing your production logic.
- Prove each detection on a real engine with a fixture it must match and one it must ignore. A query that converts cleanly can still match nothing.
- Give the build two credentials: read-only for queries, and a deploy credential no pull request build can read.
- Your git host, your CI system, and your SIEM will all be replaced before your detections are. The rule file and the record are the parts you keep.

Detection as code makes the repository the program's source of truth and the SIEM a copy of it. Whether a program has it comes down to one question: can you read the repository at a commit and say what ran in production that day, thresholds included? Storing YAML in version control gets you part of the way, because a file can still be copied into a console by hand and edited there. The rules below close the rest of the distance. The last column names what proves each rule, and it says where the kit's check is still planned.

Rule	What goes wrong without it	What proves it
Keep one directory per detection, named for its id, carrying the rule, the record, the score, the variables, the fixtures, and the approved queries.	Logic, intent, and tests are reviewed in three places, so a reviewer approves a query change and misses the record that no longer describes it.	Lint fails when the directory is missing a file the detection's stage requires.
Write the logic once in Sigma, and generate the query for every target in the registry from it. Never edit a generated query.	The same detection becomes a different detection on each platform, and a SIEM migration means rewriting every one of them.	Convert regenerates each target from rule.yml, so a hand edit shows up as a diff on the next build.
Commit the converted query for each target beside the rule, and fail the build when a conversion stops matching its approved copy.	A converter upgrade changes what runs in production, and the new query reaches the SIEM before anybody has read it.	The golden diff compares every conversion with its committed copy on each pull request.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Rule	What goes wrong without it	What proves it
Pin the converter, every query backend, and the ATT&CK release the technique ids are checked against.	A dependency that floats rewrites a production query with no commit behind it, and a detection that stopped matching is the first sign.	Each backend is capped below its next major version, and lint rejects a technique id outside the pinned map.
Declare every threshold, window, allowlist, and scope on the record with a default, bounds, and an evidence line.	Tuning edits the query, so each change needs a full re-review and the reason for the old number is gone.	Lint rejects a variable with no default or no evidence, and a threshold or window with no bounds. Substituting each value into the queries is planned.
Commit one event the detection must match and one it must ignore, and replay both on a real engine.	A query that converts cleanly can still match nothing, and the proof of it arrives as an incident the detection should have caught.	The replay job loads the fixtures into each engine and compares the hits with tests/expected.yml.
Scrub credentials, tokens, internal host names, and personal data out of a fixture before it is committed.	A captured event carries a live secret into version control, and git history keeps it after the fixture is cleaned.	A secret scan on every pull request, with a line in the review checklist as the backstop. The kit adds no scan of its own, so turn on the repository host's secret scanning.
Deploy from the build, make the production console read-only, and alarm on a live rule that no longer matches the repository.	An edit made in the console during a shift leaves production running a query no test has ever covered.	A daily check that compares each running query with its approved copy and fails when they differ. The kit's tools/running.py does it for Splunk, along with the schedule, the alert condition, and whether alerting is on. On another SIEM, compare them by hand at each precision check.
Route every change that reaches production through a pull request approved by somebody other than its author.	Under deadline pressure the checks get skipped, and a week later nobody can say which detections skipped them.	Branch protection and required status checks on the default branch, with an owner named for the detections directory.
Deploy a new detection with alerting off, count seven days of alerts, and switch alerting on only at or under the threshold on the record.	A detection reaches the analyst queue before anybody has counted what it costs, and the first measurement is the complaint from the SOC.	The deploy job packages a detection at deploy with alerting off; its vetting and install steps are placeholders a team fills in for its own stack. The pull request that moves it to tune runs a check that counts its first seven days live against the threshold, and merging that pull request switches alerting on.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Rule	What goes wrong without it	What proves it
Retire inside the repository: set the rule status to deprecated, switch alerting off, and keep the record and the retirement note.	A deleted detection takes its coverage claim, its blind spots, and the reason it retired away with it.	Lint checks the Sigma status against the stage, and the record keeps its retirement record section.

Most checks in the third column are the reference pipeline's jobs. An approval and an owner on the detections directory are repository settings, because a build can't enforce a rule about itself. The secret scan and the 30-day precision check are jobs I haven't written yet; today the review checklist covers the first and a person runs the second.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

16.1 A common pipeline

The parts differ by platform, and the order rarely does. Events land in object storage, a notification tells a runtime they arrived, the runtime runs the queries the build deployed, and the alerts reach a queue an analyst works. What the analyst records about each alert is what sends the next tuning change back to the repository. The steps below name the AWS parts; the Azure and GCP equivalents sit in the same order.

Step	What happens	Example parts on AWS	What the repository controls
1. Collect	Each log source writes its events where it can, in its own schema and on its own delay.	CloudTrail, GuardDuty, VPC flow logs, EKS audit logs, and whatever the identity provider and the EDR export	A row per source in the inventory, with the normal event rate DET-0000 alerts against.
2. Land in object storage	Events arrive as compressed JSON objects, partitioned by date, kept long enough to replay a detection across them.	An S3 bucket per account or per source, with versioning on and retention set by its bucket policy	Nothing yet. Research reads field names and the 7-day baseline out of this store.
3. Announce the new objects	One notification per object tells the next step that events arrived, so nothing has to poll the store.	S3 event notifications into SQS, SNS, or EventBridge, with a dead-letter queue behind them	Nothing. The age of the oldest message deserves the same silence alert a log source gets.
4. Normalize the fields	Vendor field names are mapped into one set, so a rule written once resolves against every source.	OCSF or ECS names, applied by the ingest function, an Iceberg table, or the SIEM's own parsers	One field map per target, written from real events, so a conversion resolves the names the store has.

Step	What happens	Example parts on AWS	What the repository controls
5. Run the detection	The query runs on each object as it arrives or on a schedule, and emits an alert when it matches.	A Lambda or Panther rule per event, a scheduled search in Splunk, Sentinel, or Elastic, or scheduled SQL in Athena or Snowflake	The rule itself. CI converts rule.yml for this target, diffs it against the approved copy, and deploys it.
6. Raise the alert	The alert carries the detection id, a link to the record, the severity, and the events that matched.	SNS or EventBridge into a case manager such as Jira, ServiceNow, or TheHive	The Response section of the record, linked from the alert, which is what an analyst reads at 3 a.m.
7. Triage in the SOC	An analyst works the alert, records a disposition and the minutes it took, and closes it.	Case management, with the disposition stored in a field somebody can count	Nothing, and every number that decides this detection's future comes out of that field.
8. Tune or retire	Precision over 30 days and minutes per true positive go back to the detection, and a change to either becomes a pull request.	A scheduled report out of case management, read at the monthly review	A tuning pull request that adds one named exclusion, or a retirement proposal with the trigger that fired.

Whatever runs at step 5, the rule file stays the same. A streaming runtime evaluates one event at a time, which suits an object store that notifies on arrival. A scheduled search in a SIEM reads a window of indexed events. Scheduled SQL in a warehouse reads the same window out of a table. What changes per target is the conversion, and section 9 covers what the warehouse variant adds on top.

Steps 1 to 5 are infrastructure, so each one has a name on every cloud:

Step	On Azure	On Google Cloud
1. Collect	Azure Activity log, Entra ID sign-in logs, and Defender for Cloud alerts	Cloud Audit Logs and Security Command Center findings
2. Land in object storage	Blob Storage, or Data Lake Storage Gen2	A Cloud Storage bucket
3. Announce the new objects	Event Grid on Microsoft.Storage.BlobCreated, into Storage queues or Service Bus	Pub/Sub notifications for Cloud Storage, or an Eventarc trigger
4. Normalize the fields	A Function or a Data Factory job writing the agreed names	A Dataflow or Cloud Run job writing the agreed names

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Step	On Azure	On Google Cloud
5. Run the detection	A Function, or a scheduled analytics rule in Microsoft Sentinel	A Cloud Run function, or a scheduled query in BigQuery

The pipeline above is a shape, and what fills it is whatever the team already runs. The tables below say what each choice gives this job and where it stops. I read each one against the vendor's own documentation on 2026-10-06. They are documented mechanisms and not products I run, and the names move fast enough that you should check again before you build on one.

16.2 Where the detections live

A detection repository asks four things of its host: a default branch nobody can push to, a check that must pass before a merge, a named owner for the detections directory, and a scan that stops a credential reaching the history. Every host below does at least three.

Host	Protected branch and required checks	Owner for a directory	Secret scanning on a push
GitHub	Rulesets, or the older branch protection rules	CODEOWNERS, required through the pull request rule	Push protection, paid on a private repository
GitLab	Protected branches, with pipelines that must succeed	CODEOWNERS, with approval enforced on Premium and above	Secret push protection, on Ultimate only
Bitbucket Cloud	Branch restrictions and merge checks	.bitbucket/CODEOWNERS, added as default reviewers	Nothing at push time, so add a pipeline step
Bitbucket Data Center	Branch permissions and the required builds check	.bitbucket/CODEOWNERS, added as reviewers	On by default, but it reports after the push lands
Azure Repos	Branch policies for reviewers, build validation, and status checks	Automatically included reviewers on a path filter, with no CODEOWNERS file	Push protection through the paid Advanced Security add-on
Gitea and Forgejo	Protected branches with required approvals and status checks	CODEOWNERS; Gitea can require the owner's approval and Forgejo can only request it	Nothing
Google Secure Source Manager	Branch rules with required approvals and checks	.securesourcemanager/CODEOWNERS	Rejects the push, with a documented skip option

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Two hosts are closed to new customers, so neither should start a detection repository: AWS CodeCommit since July 2024, and Cloud Source Repositories since June 2024.

The build needs two credentials and no more: a read-only query credential scoped to the indexes a baseline check reads, and a deploy credential only the deploy job can read. Keep the deploy credential out of pull request builds, because a pull request from a fork runs code a stranger wrote.

16.3 What runs the checks

The reference pipeline in section 17 declares its jobs for GitHub Actions: lint, convert, score, and a replay matrix with a service container per engine on every change, then deploy after a merge, and a promotion check on the pull request that switches a detection's alerting on. Any CI system can carry them. What differs is where the jobs are declared, whether a runner can sit inside the network where the SIEM runs, and what a build started by a stranger is allowed to read.

CI system	Where the jobs are declared	Runners you can own	What a build on a fork's pull request reads
GitHub Actions	Files under .github/workflows	Self-hosted runners, per repository or organization	No secrets until a maintainer approves the run
GitLab CI/CD	.gitlab-ci.yml	Self-managed runners, per group or per project	Protected variables only where the project allows them
Azure Pipelines	azure-pipelines.yml	Self-hosted agent pools	Nothing, until an administrator turns fork secrets on
Jenkins	Jenkinsfile	Agents are the whole model	What the fork trust setting allows, and Collaborators is the recommended one
CircleCI	.circleci/config.yml	Container runners and machine runners	What the context restrictions allow
Buildkite	A pipeline file the agent uploads	Every agent is yours	Whatever your own secret store hands the plugin
Bitbucket Pipelines	bitbucket-pipelines.yml	Self-hosted runners	Secured variables, masked in the log
Tekton, with Pipelines as Code	Pipeline runs under .tekton	Your own cluster	What the repository policy allows, after an ok-to-test comment
Concourse	A pipeline file set with fly set-pipeline	Workers are yours	Only what the pipeline's own credential manager gives it

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Woodpecker CI, TeamCity, Argo Workflows, Dagger, and the beta successor to Drone all do the same job. Ask any of them the same four questions: where the jobs are declared, whether a job can run beside a SIEM that has no public address, what a stranger's pull request can read, and whether the repository can refuse a merge on the answer.

16.4 What ships the rule into the platform

Delivery takes one of two shapes. The build pushes, calling the platform's API after a merge, which is simple and leaves the platform with no idea the repository exists. Or the platform pulls, reconciling against the repository on its own, which closes the drift question and gives you one more system to operate. Of the platforms in 16.5, only Microsoft Sentinel offers the pull shape itself. Everywhere else the pull side means Argo CD or Flux plus a controller that speaks the vendor's API.

How the rule ships	What it reaches	What to watch
The platform's own repository connection	Microsoft Sentinel, from GitHub or Azure Repos	Five connections per workspace, and deleting a file leaves the rule running
Terraform or OpenTofu	Sentinel, Elastic, Splunk saved searches, Datadog, Sumo Logic, and Google SecOps	A rule edited in the console drifts until the next apply silently overwrites it
The vendor's content CLI	Elastic detection-rules, Splunk contentctl-ng, Panther's upload tool, SecOps Content Manager	Upload adds and replaces, so a deleted detection needs a delete call of its own
A REST call from the build	Elastic's import endpoint, Sekoia's rules catalog, Exabeam, Devo	The script is yours to maintain, and the API host is often pinned to one region
GitOps reconciliation with Argo CD or Flux	Anything expressed as a Kubernetes resource	Neither reaches a SaaS rule without a controller that speaks the vendor's API
Packages pulled onto the node, as falcoctl does for Falco	Rules shipped as OCI artifacts	Follow mode installs new versions on its own, with no deploy step in between

16.5 The platform on the other end

Sigma covers a detection whose logic is a match on fields inside one event, which is most of them. A correlation across events, a stateful sequence, or a scheduled warehouse query goes in as its own query file, with the same record, score, variables, and fixtures beside it. Review happens on that file, because no conversion can regenerate it.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Platform	Query language	The rule in version control	What deploys it
Splunk Enterprise Security	SPL	One YAML file per detection, built into an app	contentctl-ng, then the Admin Config Service on Splunk Cloud
Microsoft Sentinel	KQL	Bicep or ARM for a scheduled alert rule, or YAML converted into it	A repository connection, an az deployment, or Terraform
Microsoft Defender XDR	KQL	A custom detection rule	The Bicep security extension, still in preview
Elastic Security	KQL, Lucene, EQL, and ES|QL	One TOML file per rule	The detection-rules CLI, through the detection engine API
Google SecOps	YARA-L 2.0 over UDM	Rule text beside the state that says whether it runs	The rules API, Content Manager, or the Terraform resource
CrowdStrike Falcon Next-Gen SIEM	Its correlation rule query	A Terraform resource per rule	The CrowdStrike Terraform provider
CrowdStrike Falcon LogScale	LogScale query	A package: a manifest with alerts, parsers, and queries beside it	Installed per repository or view, so a fleet rollout is many installs
Panther	Python, or Simple Detections YAML	A .py and .yml pair per detection	The upload command, with an API token the build carries
Datadog Cloud SIEM	Its own query syntax	A Terraform resource per rule	The Datadog Terraform provider
Sumo Logic Cloud SIEM	Its rule expressions	Terraform, with a different resource per rule shape	The Sumo Logic Terraform provider
Sekoia.io	Sigma, used natively	The Sigma rule itself	A post to the rules catalog, scripted by you
Grafana Loki	LogQL	A ruler rules file	lokitool, or the ruler API
IBM QRadar	AQL, plus the rule engine	A content export	The content management tool; the SaaS product reached end of life in April 2026
A warehouse: Snowflake, Databricks, BigQuery, Athena	SQL	A .sql file beside the schedule that runs it	A task or alert, a scheduled query, or a scheduler rule

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Sigma reaches these platforms unevenly, and the gap is worth measuring before a team commits to one rule format. Sekoia reads Sigma directly. Splunk, Elastic, OpenSearch, Kusto, Loki, CrowdStrike, Panther, and SQL all have a converter that works. Google SecOps converts only rules whose fields already have a UDM name, and the QRadar converter that still builds is the one IBM maintains. Correlation across events converts for fewer targets again, so a rule that counts or sequences is portable in theory more than in practice.

Section 17 takes the pipeline I run against this framework's own example detections, job by job.

16.6 Where this pipeline breaks

What breaks	What you see	The check that finds it
The notification queue wedges	The bucket keeps filling, every log source reports healthy, and nothing has run for six hours.	Alert on the age of the oldest message, the way DET-0000 alerts on a silent source.
A normalization change renames a field	Every rule still converts and every build still passes, and the live queries match nothing.	Field names come from real events at research, and re-validation replays the fixtures against the current schema.
Somebody deploys by hand during an incident	The live rule and the approved copy diverge, so the detection's tests prove nothing about what runs.	The daily running check, which compares the running query with its approved copy.
Retention is shorter than the longest window	You can't re-measure a detection before tuning it, so the tuning argument comes down to opinion.	Set retention from the longest window any detection declares, and check it again when a window moves.
Alert delivery fails	The failure is itself an alert that nobody receives, and the queue looks quiet because it is empty.	A dead-letter queue behind the alert topic, with what lands in it counted as a program metric.
A retired detection keeps running	The rule is gone from the repository and still live on the platform, because an upload adds and replaces but never deletes.	Make the delete its own deploy step, and count live rules against the repository at each review.
A notification fires before the object is finished	The runtime reads a partial file, so the detection misses the events that were still being written.	Subscribe to the event that marks the write complete, which on Data Lake Storage Gen2 means FlushWithClose.

These failures make no noise, with one exception: the retention gap announces itself on the day you need the data. A detection that throws an error gets noticed; a detection that runs, matches nothing, and reports success stays unnoticed for as long as nobody asks.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

17. The reference pipeline

REMEMBER

- Copy the repository layout and replace the example detection with your own.
- The scorer writes every score, and CI fails any score that disagrees with the formula.
- Convert once, commit the approved copies, then review every difference after that.
- Lint also fails when AGENTS.md no longer matches framework.json.
- The Splunk replay leg needs only its container's password and HEC token. The promotion check needs a read-only Splunk account in the siem-read environment, and the two daily checks need the same account in siem-read-daily. The deploy job needs the deploy credential in the production environment.

If you're using the repository as a template, replace the example and point the secrets at a Splunk Cloud stack. Each detection is a directory. It carries rule.yml (Sigma), ads.md (the record), score.yml (the inputs, an evidence line for each, and a CI-written score), the fixtures, and the approved copies of the converted queries.

Stage	Sigma status	CI jobs	Artifact
1. Intake	no rule file yet	score	score.yml with stage set to intake
2. Hypothesis	no rule file yet	lint	The Goal, Categorization, Strategy Abstract, and Blind Spots and Assumptions sections of ads.md
3. Research	no rule file yet	lint	The Technical Context section of ads.md; the emulation plan and run report; tests/events.jsonl
4. Build	experimental	lint, convert	rule.yml, tests/events.jsonl, tests/expected.yml
5. Validate	test	lint, convert, replay (one leg per engine), golden-diff, review	Passing CI run; approved pull request
6. Deploy	stable	deploy, promote-check, field-check (scheduled), running-check (scheduled)	The pull request that moves the detection from deploy to tune, carrying the promotion check's count for its first 7 days live
7. Tune	stable	lint (core lock, on pull requests), field-check (scheduled), running-check (scheduled), precision-check (scheduled, planned)	A tuning pull request that cites the alert ids that prompted it

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	Sigma status	CI jobs	Artifact
8. Retire	deprecated	deploy (the next package leaves it out)	A retirement note in ads.md: reason, date, and the id of the replacement detection

The definitions of done call these tools in the repository:

Tool	Jobs it backs	What it does
lint	lint	Checks the layout and parses the detection. It confirms the id and tag agree, the Sigma status matches the stage, the ADS sections are present, and every named fixture exists. It also validates the score record, rejects any ATT&CK id outside the pinned map, and fails when AGENTS.md no longer matches framework.json.
convert	convert, golden-diff	Turns each Sigma rule into all 13 targets in targets.yml, and fails when any output differs from its approved copy.
score	score	Computes each record's score and sorts the backlog.
metrics	score	Adds to the score job's metrics the numbers only the file layout, incidents.csv, and notifications.csv can tell.
package	deploy	Builds a Splunk app from the approved queries. A detection at deploy runs with alerting off, one at tune runs with it on, and a retired one is left out, which takes it out of production.
replay	replay, promote-check	Loads the fixtures into a real instance of each replay engine (SQLite, Splunk, Kusto, Elasticsearch, ES|QL, OpenSearch). On the pull request that moves a detection to tune, it counts the first 7 days live against the threshold.
history	promote-check	Reads from git when each detection changed stage and when it went live, so no record carries a date by hand.
fields	field-check	Once a day, counts the events from each live rule's log source and how many carry each field the rule reads. It fails a source that sent nothing and a field that was empty in every event.
running	running-check	Once a day, compares each live rule's saved search in Splunk with what the package writes from main. It fails a changed query, schedule, or alert condition, a disabled search, alerting at the wrong stage, and a search in the app with no live detection behind it.
next	none; run by hand	Runs the checks a detection's current stage needs, in one command, and prints what is still missing, the count the stage must reach, and the step that moves the detection on.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Tool	Jobs it backs	What it does
intake	none; run by hand	Prints the weekly intake meeting's agenda from the records and git: new candidates, the ones blocked on telemetry, the queue in scored order, the work in flight, and what has not moved in a month.

The precision-check job is planned and not yet written. The other seven targets (EQL, Loki, LogScale, NetWitness, Panther, Microsoft Defender XDR advanced hunting, Google SecOps) have no public test instance. Their conversions stay drafts until someone runs them on a real one. Separate upkeep tools reshape fixtures for each engine, refresh the pinned ATT&CK map, and regenerate the agent instructions.

I cap each query backend below its next major version in requirements.txt. A major release can change the query a backend emits, and the golden check catches that. When pysigma-backend-sqlite 2.0.0 renamed the table placeholder, it failed the check on DET-0001. To move a cap, you open a pull request that updates the approved copies in the same change.

These have run on the example: lint, convert with the golden check on all 13 targets, score, metrics, packaging with one detection at deploy and one at tune, and the promotion check against a stubbed search. The two daily checks ran against a real Splunk in a container. The field check ran on the fixtures with fields renamed, blanked, and removed, and the running check on the packaged app after edits made through the console's own API. Every replay adapter has run end to end against a real engine in a container: SQLite, Splunk, the Kusto emulator, Elasticsearch, and OpenSearch each returned the two fixtures DET-0001 must match and none of the three it must ignore. The site's own build also copies the kit and breaks it on purpose, one case per check: a rule only some targets can convert, a regular expression, a silent re-approval, each stage move the lifecycle refuses, and each count the promotion check fails on. The build stops if any tool misses its break. These haven't run anywhere yet and stay placeholders until they do: the container services inside the GitHub Actions workflow, and the AppInspect and ACS steps.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

18. The closed loop

REMEMBER

- Write every check against the way an agent would fake its way past it, and make each check a job the build runs.
- Take field names from a dictionary the SIEM exported today, never from documentation or memory.
- Fixtures come from a collector the agent cannot write to, with a signature the build verifies.
- Lock the selection blocks that express the behavior with a hash. Tuning adds filters beneath them and changes nothing above.
- Survival after 90 days is the number that says the checks work. Acceptance alone says reviewers are tired.

Section 10 lets agents draft the lifecycle's artifacts while people review, deploy, and retire. In the closed loop, agents run the mechanics of every stage, from a threat report to a tuning pull request, and people keep the approvals. The loop stays safe because each stage ends on a check written to catch the way an agent would fake its way past it. Three routes send work back when a check fails.

The loop has six machine stages, and they map onto the eight lifecycle stages. The definitions of done, the records, and the kit's tools stay the same; the loop adds the checks an unattended run needs and the routes back. Nothing in the Never list of section 10 changes. An agent still never approves, deploys, switches alerting on, or acts on an instruction found in a report or a log.

18.1 The six stages and their checks

Stage	Lifecycle stages	What the agent does	What passes it
Extract	Intake and hypothesis	Reads a threat report, an incident, or an exercise result as data and writes a blueprint: the technique id, the behavior in terms of what the host or service must do, and the fields that would show it. It leaves out the clues the attacker picks, such as file names, hashes, and addresses. It drafts the six scores with an evidence line each, which is intake.	The blueprint validates against a schema, and every technique id is in the pinned ATT&CK map. No rule in the repository already names the same behavior, by a similarity score under a threshold the team sets.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	Lifecycle stages	What the agent does	What passes it
Ground	Research	Looks each field in the blueprint up in a dictionary exported from the SIEM. Documentation is never the source. Checks that the source is arriving: the share of hosts that reported in the last day and the rate of empty values in the fields it needs. Counts the 7-day baseline with a read-only query.	Every field the rule will read exists in the live dictionary. The source reports from at least the share of the fleet the team sets, and 85% is a starting point. The baseline count and its query are in the record. A missing field or a silent source stops the run and opens a log source request, which is research's route back to intake.
Author	Build	Writes the rule in Sigma from the blueprint and the dictionary, and converts it for every target. Writes the fixtures it expects to match and to ignore. Declares every threshold, allowlist, and scope as a variable with bounds and evidence.	Lint passes, each conversion matches its approved copy or creates one, and the rule passes the validators. No high-volume field is matched with a leading wildcard, and the estimated scan over the baseline window is under the team's cap. The selection blocks that express the behavior are canonicalized and hashed, and that hash is the core the later stages protect.
Prove	Research and validate	Runs the behavior in a throwaway environment, collects the events, and replays the converted query over them and over the last 30 days of production.	Every variant the plan names is caught, the fixtures carry the collector's signature, and the 30-day replay's matches are grouped by parent process, signer, and command, with the groups in the record. A missed variant goes back to Author. Matches over the threshold go on to Tune.
Tune	Tune	For each group of benign matches, adds one filter beneath the locked core: the parent process, a valid signature and its signer, and the command or flag, as an allowlist entry with a ticket and an expiry. Then runs Prove again.	The core's hash is unchanged. The diff adds filter selections and nothing else. Each filter names at least three conjunctive attributes, one of them a signature where the source carries one, and no path wildcard. Recall is still complete, and the 30-day matches are under the threshold. A rule with no historical matches at all skips tuning, because an exclusion on a rule that never fired removes something nobody has seen.
Ship	Validate, deploy, and the first week live	Opens one pull request carrying the rule, the conversions, the signed fixtures, the replay counts, the hash of every artifact, and every query it ran with its time range and row count. After the merge it packages the rule with alerting off, and seven days later opens the pull request that switches alerting on, with the week's count attached.	Branch protection requires every check above plus one approval by a person, and the deploy run needs the lead's approval. Its opening seven days run with alerting off and at or under the threshold. A rule that fires more than its hourly limit after alerting is on goes back to alerting off and to Tune.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

18.2 How each check can be gamed, and what stops it

Stage	How an agent games it	What stops it	Tools
Extract	The report is the attack surface. Text planted in a page, a ticket, or a log can tell the agent to skip a check or weaken a rule.	The extractor produces structure and nothing else. It runs with no tool but the schema validator, hidden text and markup are stripped before it reads, and the hostile-input rules of section 10 apply to everything it sees.	A model call with a JSON schema for its output; the pinned ATT&CK extract for ids; an embedding or tag match against the catalog for duplicates.
Ground	An agent invents a field name that reads well, or picks a source that is configured but not arriving, and the rule converts cleanly and matches nothing.	The dictionary comes from a scheduled job the agent does not run, exported daily from the SIEM's own schema. The agent's query credential is read-only and scoped to that export and the baseline count.	Splunk tstats and fieldsummary over the data models, Microsoft Sentinel's table schema and getschema, Elasticsearch's field capabilities, OCSF for the neutral names; the framework's inventory.csv for the sources and their silence alert.
Author	A rule that matches the blueprint's ephemeral clues, the file name or the hash, passes every syntax check and is worthless a week later.	The robustness draft scores the rule from its observables, the way the library's sources page describes. A draft at level 1 fails the build unless the record says why that is enough.	pySigma and its validators, sigma-cli check, the backends in the kit's target registry; the SIEM's job statistics for bytes scanned; the kit's lint, convert, and golden diff.
Prove	The agent writes the fixtures it needs, or picks the replay window that returns the count it wants.	The sandbox and its collector run under a credential the agent does not have and sign what they emit. The replay window is fixed by the job. The fixtures are read from the signed archive, never from the pull request.	Atomic Red Team through Invoke-Atomic, Stratus Red Team for cloud accounts, and Caldera for chains. Throwaway hosts from Terraform or Ludus, with Sysmon and nxlog or Winlogbeat. OTRF's Security-Datasets when there is no lab. The SIEM's own search over 30 days, or a lakehouse query over OCSF, and Chainsaw or Hayabusa for EVTX offline.
Tune	The cheapest way to zero false alarms is to silence the rule: widen a filter, drop a selection, or exclude a directory.	The lock, the additive-only diff, and the re-run of recall. The exclusion policy is a lint rule the build runs.	The kit's declared variables and allowlist entries; a lint that diffs the parsed rule between base and head; the same replay as Prove.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	How an agent games it	What stops it	Tools
Ship	A reviewer who trusts the bundle approves in a minute, and the bundle is the agent's own account of itself.	The bundle links to artifacts that other jobs produced, under hashes the reviewer can check, and the metrics below count how many approvals still look right after 90 days. That number says whether review is still review.	GitHub Actions job summaries and branch protection, the kit's promotion check, and the rule APIs of Sentinel, Splunk ACS, and Elastic. A scheduled alert-rate query that switches alerting off. The case tool's dispositions, such as Sentinel's incident classification or a Splunk ES notable status, as the webhook that drafts a tuning pull request.

18.3 The routes back

Route	When it is taken
Prove to Author	A variant in the plan that the rule misses sends the blueprint and the missed event back to Author. The next draft is written against the event the rule did not match.
Prove to Tune	Matches over the threshold in the 30-day replay go to Tune with the groups already made, and the tuned rule goes through Prove again before it ships.
Live to Tune	An analyst's false-positive disposition, an alert rate over the hourly limit, or a daily field check that finds a renamed column each opens a tuning pull request. The detection stays live until it merges, unless the rate already switched alerting off.

18.4 What people keep

- Approve every pull request. Branch protection enforces it, so it never depends on habit.
- Approve the deploy run and the pull request that switches alerting on.
- Decide every retirement, as in section 6.
- Set the thresholds the checks use: the share of the fleet a source must report from, the scan cap, the exclusion policy, and the hourly alert limit. Write each one down with the reason.
- Read the 90-day survival number every month, and sample the approvals when review time per pull request falls under a few minutes.

18.5 What the kit has and what it lacks

Lint on the record, the fixtures, and the stage moves. Conversion with an approved copy per target, and replay on five engines. Declared variables, and the promotion check that reads a detection's live date from git. The core lock and the exclusion policy, in tools/corelock.py. The daily field check in tools/fields.py fails when a live rule's log source sends nothing or a field the rule reads is empty in every event. A person opens the tuning pull request from the failed

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

run. The daily running check in tools/running.py fails when a live rule's saved search in Splunk differs from what the deploy job packaged, is disabled, or alerts at the wrong stage. AGENTS.md as the policy an agent runner reads, the hostile-input rules, and the four research lines the record must carry.

The extractor, and the daily export of the whole field dictionary for Ground to draft from. The scan estimate, and the robustness draft as a build check. The sandbox runner and its signing collector, and the 30-day replay job. The alert-rate check that switches alerting off, and the disposition webhook. Each is one job in the pipeline, and the rollout below adds them in the order that pays first.

18.6 The pieces, specified

Each check above is a job. These are the six pieces the loop rests on, specified closely enough to build. The kit carries two of them, the core lock and the exclusion policy; the other four are the team's to build.

Piece	What it is
The field dictionary	One table a scheduled job writes daily: source, table, field, type, the share of events in the last 24 hours in which the field was non-empty, and the share of hosts that reported. Ground reads this and nothing else for field names. A field under the team's non-empty share is treated as absent.
The core lock	Every selection in the detection block whose name does not start with filter, plus the condition with its 'and not filter' clauses removed. The two are canonicalized, with keys and value lists sorted and values stripped, and hashed with SHA-256. Tuning must leave the hash unchanged. The kit's tools/corelock.py computes it and fails a pull request that changes it on a live rule.
The exclusion policy	A filter is one map of at least three fields that must all be true, such as the parent process, the signer or original file name, and the flag or path. No value may replace a whole directory with a wildcard. Removing a filter widens the rule and goes back through validate. Each filter cites a ticket and an expiry in the record's allowlist variable.
The signed fixture	The collector in the sandbox writes each run's events to an archive with a manifest listing every file's SHA-256, and signs the manifest with a key the agent never sees. Prove reads fixtures from that archive only. The build verifies the signature before it counts a match as recall.
The pull request bundle	The rule, the approved copy for every target, the signed fixture archive's hash, and the recall result per variant. With them, the replay window and its counts grouped by parent and signer, the core hash, the scan estimate, and every query the agent ran with its time range and row count. Branch protection requires the jobs that produced each item.
The alert-rate check	A scheduled query every few minutes counts each live rule's alerts in the last hour against the hourly limit on its record. Over the limit, the job switches the rule's alerting off, records the hour's count, and opens a tuning pull request with the matches grouped. Alerting comes back on only through the promotion pull request, as for a new rule.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

18.7 What to measure

Target	How to measure it
First-time pass rate per check	The monthly share of runs that pass each check on the first try. A check that always passes checks nothing, and one that never passes is set wrong.
Trips around the loop per rule	Median count of returns to Author and to Tune before a rule merges. A rising number points at the extractor or the dictionary before it points at the model.
Hours from intake to review-ready	The intake_to_review_ready_hours metric of section 10, which the loop should move from days to hours. Report the median; the best run flatters.
Acceptance and survival	agent_draft_acceptance_rate and agent_detection_survival_90d from section 10. Survival is the number that tells you the checks work; acceptance alone tells you reviewers are tired.
Review minutes per pull request	Measured from the pull request's timeline. When it falls below a few minutes, sample a tenth of the approvals and re-review them.
Alerting switched off by the rate check	Count per month, with the rule and the cause. Each one is a tuning pull request the loop should have opened earlier.
Renames the field check caught	Count per month. Each one is a detection that would have gone silent.

18.8 Rolling it out

Phase	Add	Done when
Days 1 to 30	The repository, lint, conversion with approved copies, the daily dictionary export, and the scan cap. No model yet.	No rule reaches a pull request with a field the dictionary lacks, and no query over the scan cap merges.
Days 31 to 60	The extractor, then Prove: the sandbox or the recordings, the signing collector, and the 30-day replay.	Every pull request carries signed fixtures, a recall result, and a replay count.
Days 61 to 90	Tune with the core lock and the exclusion policy, the promotion check, the alert-rate check, and the disposition webhook.	A false-positive disposition becomes a tuning pull request without a person starting it, and the core hash of every live rule matches its approved copy.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

18.9 One pass, by hand

No agent has run the loop. To see whether its checks work on real events, I walked the six stages by hand. The tools were the kit's, and the events were the 107 recorded attacks the library already runs its rules against. The procedure was LSASS memory dumped through comsvcs.dll, T1003.001, the example the reference architecture behind this section was built around. The table shows what each stage did and what came back.

Stage	What was done	What came back
Extract	Read the procedure. rundll32.exe loads comsvcs.dll and calls its MiniDump export against the process id of lsass.exe. The library name and the export are durable; the attacker can also name the export by its ordinal, 24. The dump file's path and name are the attacker's choice, so they were left out.	One behavior and two candidate observables: the command line of the rundll32 process, and the call trace of its access to lsass.exe. Both map to T1003.001, which the offline ATT&CK copy confirms is a real id.
Ground	Counted, in the recordings, the two event types the candidates read, and how many of each had its field filled.	Sysmon event 1, process creation: 1,515 events in 95 of the 107 recordings, with CommandLine filled in every one. Sysmon event 10, process access: 249,519 events in 103 recordings, with CallTrace filled in every one. Both fields are safe to build on.
Author	Wrote the rule as Sigma: process creation where CommandLine contains comsvcs and one of MiniDump, #24, #+24, or '24 followed by a space. Computed its core hash and its robustness draft, and converted it to SQL with the kit's SQLite backend.	Core hash b251dfc760654ed9. The same rule with its keys and its list of values in another order gives the same hash. The robustness draft is level 2, text from the attacker's own tool, because the attacker types the command line. SigmaHQ's rule for the same procedure, a49fa4d5, reads the call trace of the process access, and it drafts at level 2 as well.
Prove	Ran both rules over all 776,654 events in the corpus, grouped by recording. The recording of this procedure is SDWIN-201018195009, Lsass Memory Dump via Comsvcs.dll, 184 events.	The candidate fired on 1 event, in that recording, and on nothing else in the corpus. SigmaHQ's rule fired on 2 events, in the same recording, and on nothing else. The match is the fixture: rundll32.exe started by powershell.exe as WORKSTATION5\wardog, with the MiniDump command line quoted below. Recall against the one recording of this procedure is 1 of 1.
Tune	Looked for noise to exclude.	There was nothing to exclude. The corpus has no benign recording, so zero false positives here says nothing about a real fleet. The exclusion policy was tested another way. The kit self-test adds a one-field filter, a filter that wildcards a directory, and a change to a live rule's selections, and the core lock refuses each.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	What was done	What came back
Ship	Nothing. No SIEM received the rule, no pull request was opened, and no seven days ran.	Most of the bundle the pull request would carry exists: the rule, the SQL conversion, the one fixture, the replay counts per recording, the core hash, and the queries below with their row counts. The signature on the fixture does not, because no collector outside the author's reach produced it.

The one event the candidate matched, quoted the way a record quotes log content, as data in a fenced block:

```
"C:\Windows\System32\rundll32.exe" C:\windows\System32\comsvcs.dll MiniDump 756 C:
\Users\wardog\AppData\Local\Temp\lsass-comsvcs.dmp full
```

The queries the pass ran, listed the way the pull request template asks for them:

Query	Over	Rows
Sysmon event 1 with CommandLine filled, by recording	all 107 recordings	1,515 in 95 recordings
Sysmon event 10 with CallTrace filled, by recording	all 107 recordings	249,519 in 103 recordings
The candidate rule, converted to SQL	all 776,654 events	1, in SDWIN-201018195009
SigmaHQ a49fa4d5, converted to SQL	all 776,654 events	2, in SDWIN-201018195009
Any command line containing comsvcs	all 776,654 events	2, both in SDWIN-201018195009

The other four T1003.001 recordings, Mimikatz twice, Task Manager, and direct syscalls, each set off 8 to 10 library rules, and none of those rules was a comsvcs rule. One rule covers one procedure, and a technique's recall is the sum over its procedures. Extract should emit one candidate per procedure a report describes, and a coverage count by technique overstates what one rule covers.

The pass shows that ground, author, prove, and the core lock work on real events with the kit's tools, and that the replay has a corpus to run on today. It does not show that a model can do extract or author unassisted, or how the exclusion policy fares against a tuning agent. It says nothing about the rule's alert rate on a real fleet. The outside checks page lists the pass and these gaps.

The loop multiplies what section 10 already says about the review bottleneck. Cap open agent pull requests at what reviewers clear in a week, and let the scored backlog decide what gets drafted.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

A 30-day replay over every candidate is the loop's largest bill. Run it over a sample of hosts first, and over the whole fleet only for a rule that passed.

One pass has been measured, by hand and against recordings, in 18.9. The targets above are what a team should expect to see, and the outside checks page lists them as untested.

19. Choosing AI for detection and response

> **REMEMBER**
>
> - Adopt AI where a bad output costs one review, such as drafting into a pull request, before uses where it costs an incident.
> - Count an agent's benefit in analyst minutes saved and its price in alerts it wrongly closed, and report the two together.
> - Run an AI triage agent in shadow mode on the team's own alerts, and replay every confirmed incident through it before it closes anything.
> - Ask for all four counts on the team's own alerts, with the number of real attacks. A share of alerts closed hides every attack the agent missed.
> - Ask any product where its rules go, which events prove them, and who switches alerting on. Grade it by its weakest answer.
> - Treat a new model, prompt, or vendor release as a new agent that goes back through shadow mode.

An AI detection and response program pays for itself in analyst and engineering minutes. It fails when it misses an attack it was trusted to catch. Every choice in this section weighs the minutes an AI use saves against the evidence that it misses nothing. The same tests apply whether the AI ships inside a SIEM, comes from a separate vendor, or is assembled from open-source parts.

Section 18 says what a safe automated detection pipeline looks like. This section turns that design into a buying and building decision. It covers which uses to adopt first, which path fits a team, what to ask any product, and how to prove an AI triage agent before it closes an alert.

19.1 Where AI earns its keep

Each use below saves a kind of minute the scaling ceilings of section 14 count, and each can break something the lifecycle exists to catch. The order is the order to adopt them in.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Use	Minutes it saves	What it can break	What proves it	When to adopt it
Drafting detections and records	Engineering minutes spent writing a rule, its record, and its conversions.	A rule keyed on one tool's fingerprint, a field the logs do not carry, or a test event the model wrote.	The closed loop's checks in CI, and the share of drafts still live at 90 days.	First. Every draft goes through a pull request, so a bad one costs one review.
Translating rules between platforms	Engineering weeks in a SIEM migration.	A translation that reads the same and matches different events, as with case sensitivity or a renamed field.	The same fixtures replayed in the old engine and the new one, with identical matches.	First, during any migration.
Enriching and summarizing an alert	Analyst minutes per alert, which raises the triage ceiling.	A summary that leaves out the deciding fact, or states one the data does not show.	A weekly sample in which analysts mark each summary right, incomplete, or wrong.	Second. It only reads, and the analyst still decides.
Triage verdicts that close alerts	More analyst minutes than any other use.	It closes a real attack, the costliest failure the program can have.	Shadow mode on the team's own alerts, with every confirmed incident replayed through it, as 19.4 describes.	Third, one alert class at a time, after shadow mode passes.
Tuning proposals	Upkeep minutes spent on noisy rules.	An exclusion broad enough to blind the rule.	The core lock and the exclusion policy of section 18, with the true-positive fixtures still firing.	Once the core lock runs in CI.
Response actions	Minutes to contain an intrusion.	A wrong action that causes an outage, or one an attacker steered with text planted in a log.	A playbook the team wrote for each action, a log of every action taken, and the count of actions people reverted.	Last. Without a person's approval, only actions that can be undone in one step.

19.2 Build, buy, or assemble

Path	It fits when	Watch for	Start with
Use the AI in the SIEM or XDR the team runs	One platform carries most of the telemetry, and the team is small.	Agent actions logged only in the vendor's console, verdicts that cannot be replayed, and per-unit pricing that grows with use.	The investigation assistant, read-only. Then triage in shadow mode.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Path	It fits when	Watch for	Start with
Buy an AI SOC product	Alert volume is past the triage ceiling, and the telemetry spans several tools.	Write credentials into the team's tools, accuracy claims with no method, and verdicts that cannot be exported.	Shadow mode on 30 days of the team's own alerts with known outcomes.
Assemble from open models and MCP servers	Logs cannot leave the team's tenant, or the team already runs detection as code.	The team owns the evaluation, the defenses against planted instructions, and the on-call rota for the agent.	A drafting agent that opens pull requests into the kit's CI and does nothing else.
Build the closed loop of section 18	The detection program is itself the product, as at a managed security provider or a team with its own platform group.	The checks cost more than the model. Budget the field dictionary and the signed fixtures before the prompts.	The 90-day rollout in 18.8.

A team can mix the paths by function. A common split takes triage from the SIEM vendor, builds drafting around the team's own repository, and keeps response with people until the action log has a quarter of history.

19.3 What to ask any product

Each question comes from a check in section 18. A product that passes all of them behaves like the closed loop, whoever built it. The Grade an AI product tool records the answers and does the arithmetic.

Area	Ask	A passing answer	A red flag
Extract	What happens to instructions hidden in a threat report, a ticket, or a log line?	Untrusted text reaches only a model with no tool access, and the vendor shows a test of it.	The model has been trained to ignore them.
Ground	Where do the field names in a generated rule come from?	From the team's own data, queried read-only when the rule is drafted.	From documentation, a schema the vendor ships, or the model's memory.
Author	Where does a generated rule go when it is done?	Into a pull request in the team's repository, through the team's CI.	Straight into the SIEM, or into a vendor-side library the team cannot diff.
Prove	Which events prove that a generated rule fires?	Captured events from the team's environment or a named emulation run, stored beside the rule.	Events the model wrote, or no test events at all.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Area	Ask	A passing answer	A red flag
Tune	Can the agent change what a live rule catches?	Only by adding a narrow, named exclusion that a person approves. The detection logic itself is locked.	It adjusts live rules on its own to cut noise.
Ship	Who switches alerting on for a new rule?	A named person, after a period with alerting off and an alert count at or under its threshold.	The agent, or anyone with access to the console.
Triage	Which alerts can the agent close without a person, and who decided that?	Classes the team chose after shadow mode, each with a measured false-close rate.	Any alert above a confidence score the vendor set.
Investigate	Can every query the agent ran be seen outside the vendor's console, with its time range and cost?	Yes. The SIEM's own audit log records them under the agent's identity, or the product exports them.	Only a written summary of the investigation.
Respond	What can the agent do without a person's approval?	A short list of actions that can be undone, each in a playbook the team wrote.	Anything an analyst could do.
Access	What credentials does the agent use?	Its own identity, read-only and scoped, with write access only for the listed actions.	An administrator's API key, or an analyst's account.
Data and model	Where do the team's logs go, and can the model be pinned?	A stated region, no training on customer data, and a pinned model version whose changes are announced.	The model changes without notice.
Accuracy	What were the four counts in the trial, real attacks escalated and closed and benign alerts escalated and closed?	All four counts for each alert class, from the team's own labeled alerts during the trial, with the number of real attacks stated.	A share of alerts closed, an agreement rate, or a percentage from an unnamed benchmark or other customers.

The grade is the weakest answer among the questions that count. Any failed question keeps that function with people until it passes.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

19.4 Proving an AI triage agent before it closes anything

An agent that closes alerts goes through the steps a detection takes before its alerting goes on. It runs where it can do no harm, its record is counted, and a person promotes it.

1. Run the agent on every alert for 30 days with no power to close one. It records a verdict, and analysts decide as before without seeing it until they have.
2. Replay the alerts of every confirmed incident and purple-team exercise from the last year through the agent. Each one must come back escalated. The number replayed sets the smallest miss rate the trial can show, as 19.7 explains.
3. Count the agent's four cells for each alert class: real attacks it escalated or would have closed, and benign alerts it escalated or would have closed. Count analyst minutes per alert with and without its summary too.
4. Promote one class to auto-close when it shows no wrong closes over at least 100 alerts and every replayed incident came back escalated. The team can raise either number.
5. After promotion, analysts review a random 5% of each class's auto-closed alerts every week.
6. One confirmed miss sends the class back to shadow mode. So does a new model, prompt, or vendor release, because each makes it a different agent.

19.5 Metrics for AI in detection and response

Metric	Unit	Aggregation	Definition
`ai_false_close_rate`	percent	per month	Share of alerts later confirmed malicious, by an incident, an exercise, or the weekly sample, that the agent closed or would have closed.
`ai_verdict_agreement`	percent	per month	Share of shadow or sampled alerts on which the agent's verdict matched the analyst's.
`ai_minutes_saved_per_alert`	minutes	median per month	Analyst minutes per alert without the agent's summary, minus minutes with it, on the same alert class.
`ai_actions_reverted`	count	per month	Response actions the agent took or proposed that a person then undid or refused.
`ai_cost_per_true_positive`	currency	per month	The agent's spend on licenses, model tokens, and query compute, divided by the true positives it worked on.

The triage ceiling of section 14 rises as minutes per alert fall, so minutes saved is the agent's benefit. The false-close rate is its price, and a report shows the two side by side.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

19.6 What ships today

These tables list what each maker documents, read on 2026-10-17 from product documentation, release notes, model cards, and repositories, with DefendBench read on 2026-10-27. Each row maps a product to the loop's stages and lists the controls its maker documents. Grading is left to the questions in 19.3 and the scorecard tool.

Read side by side, the documents show the following.

- Every SIEM and XDR platform read for this table ships AI for triage or investigation, and most let a person approve each response action.
- Google's detection engineering agent and Panther's builder test the rules they write against events or test cases a model generated. Prophet backtests against the customer's own history, which counts a rule's noise. None of the products that write rules documents a test against a recorded run of the attack, the event section 18's Prove stage asks for.
- No AI SOC product or platform publishes all four counts of its triage record from production. Qevlar publishes a false-negative rate without saying what it is out of, and 19.7 compares what each maker publishes.
- Several MCP servers can change the tenant as soon as they run, so the access question in 19.3 applies to the open-source path too.
- The one public benchmark that scores generated rules covers Azure and KQL. DefendBench scores rules on six SIEMs and keeps its tasks private. No defense against planted instructions reports a test on log fields.

AI in the SIEM or XDR

Google's detection engineering agent and Panther's builder are the two platforms here that test the rules they write.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Security Copilot with Sentinel and Defender XDR (Microsoft). The triage agent is generally available for user-reported phishing and in preview for some identity and cloud alerts. The Sentinel MCP server is in preview, with some tools generally available.	Ground, author, triage, investigate, respond with a person	The triage agent classifies Defender alerts and resolves the ones it judges false positives, leaving true positives open. A hunting assistant writes and runs KQL, and a migration tool maps Splunk and QRadar rules to Microsoft's own.	The agent runs under its own Entra identity with scoped roles, and its actions go to Purview audit logs. Analyst feedback becomes lessons in its memory, which an administrator can review or reject.	learn.microsoft.com (1), learn.microsoft.com (2), learn.microsoft.com (3), learn.microsoft.com (4), learn.microsoft.com (5)
Gemini in Security Operations (Google Cloud). The triage agent is generally available. The detection engineering agent and event simulation have been in preview since 2026-08-18.	Extract, author, prove, ship, triage, investigate	A triage agent investigates SIEM alerts and returns a verdict with a confidence and a summary. A detection engineering agent, run from MCP clients, drafts YARA-L where no existing rule matches the intelligence, and tests rules against simulated events.	The user approves or rejects each draft rule. Simulated events are machine-generated, ingested live under a SIMULATION label the caller must add, and counted against quota. Agent calls go to Cloud Audit Logs.	docs.cloud.google.com (1), docs.cloud.google.com (2), docs.cloud.google.com (3), docs.cloud.google.com (4), docs.cloud.google.com (5)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Charlotte AI (CrowdStrike). Detection triage has been available since 2025-02-13. The product documentation needs a customer login, so this row rests on public pages.	Author, triage, investigate, respond	Detection triage gives each endpoint detection a priority, a true or false positive call, and a recommended action. Named agents write correlation and YARA rules and translate Splunk queries into CrowdStrike's query language.	It takes no automated response by default, and each action can run on its own or wait for approval, set per workflow. Responses respect the user's permissions. The public pages do not say where audit logs or data are kept.	crowdstrike.com (1), crowdstrike.com (2), crowdstrike.com (3)
Cortex XSIAM with Cortex AgentiX (Palo Alto Networks). AgentiX has been available in XSIAM since 2025-10-28. Agentic response and AI data-model rules are in preview.	Ground, author queries, triage, investigate, respond	Agents plan and run playbooks, scripts, and commands on cases, write and run XQL, and summarize cases. An AI feature proposes field mappings for raw data from sampled logs.	Sensitive actions need a user's explicit approval, and an audit dataset names the agent and the user behind each action. EU and US tenants pick the model per chat. Prompts stay in region and are not used for training.	cortex-docs.paloaltonetworks.com (1), cortex-docs.paloaltonetworks.com (2), cortex-docs.paloaltonetworks.com (3), cortex-docs.paloaltonetworks.com (4)
Splunk Enterprise Security (Splunk, a Cisco company). Enterprise Security 8.7 was released 2026-09-02. The AI SOC analyst is in the Premier edition.	Author, tune, triage, investigate, respond	A detection builder drafts and tunes SPL detections in the editor, and it cannot save, test, or change a detection's status. An AI SOC analyst triages findings and runs the response actions it is allowed to run.	Each builder suggestion needs Approve or Reject, and each response action is set to run automatically or wait for approval. The MCP server blocks REST writes, and its Enterprise Security tools only read.	help.splunk.com (1), help.splunk.com (2), help.splunk.com (3), help.splunk.com (4)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Elastic Security (Elastic). Agent Builder has been generally available since 9.4, with AI rule creation in the Enterprise tier. Attack Discovery is generally available.	Ground, author, triage, investigate, respond	Drafts an ES|QL rule with an ATT&CK mapping from a description and previews it on live data. Automatic Migration translates Splunk, QRadar, and Sentinel rules, and Attack Discovery groups alerts into attacks.	Rule drafts must be applied by hand, and migrated rules are installed for the user to enable. The customer picks the model connector and can anonymize fields. Everyone using one MCP API key acts with that key's permissions.	elastic.co (1), elastic.co (2), elastic.co (3), elastic.co (4), elastic.co (5)
Purple AI (SentinelOne). Auto-triage has been generally available since April 2025. The product documentation needs a customer login, so this row rests on public pages.	Triage, investigate, respond	Investigates alerts without an analyst, collects evidence, and issues verdicts that can trigger automated responses. No detection-writing feature is documented publicly.	Automated actions run only within policies the customer approved, and each verdict carries an evidence chain. Customer data is not used for training.	sentinelone.com (1), sentinelone.com (2)
Panther AI (Panther Labs). The AI Detection Builder has been in open beta since version 1.118, and automatic closing of low-risk alerts is in closed beta.	Author, prove, ship, triage, investigate	The builder writes Python detections, log types, and test cases, and triage judges each alert a true or false positive.	Write tools wait for a person to accept or reject them, and each decision goes to Panther's audit logs. Builder changes save only when a person clicks Deploy. Claude runs through Amazon Bedrock in the customer's single-tenant AWS account.	docs.panther.com (1), docs.panther.com (2), docs.panther.com (3)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Cloud SIEM SOC analyst agent (Sumo Logic). The help page states no release stage.	Triage, investigate	Investigates each Cloud SIEM insight in the background and labels it malicious, suspicious, benign, or inconclusive, with its key findings.	It takes no remediation or containment action, and its recommended severity applies only when an analyst accepts it. Every tool call is logged to Cloud SIEM. Models run on Amazon Bedrock, with no training on customer data.	sumologic.com (1), sumologic.com (2)

AI SOC products

Products that triage and investigate alerts from the team's existing tools. Qevlar alone publishes a false-negative rate, from a test it ran, and Intezer alone publishes results with a sampling method. Section 19.7 compares what each publishes.

Product	Stages and functions	What it does	Controls it documents	Sources
Dropzone AI SOC Analyst (Dropzone AI).	Triage, investigate, respond	Investigates each alert it receives and writes a conclusion with its evidence. Containment runs when a user starts it, and customer-written Python actions can run without a click.	Read-only access by default. A changelog and a system events log record activity. Models from Anthropic, Azure OpenAI, and others, in AWS's Oregon region, or in the EU on request, with no training on customer data.	docs.dropzone.ai (1), docs.dropzone.ai (2), docs.dropzone.ai (3), dropzone.ai
Socrates (Torq).	Triage, investigate, respond	An agent inside Torq's automation platform that triages and investigates cases and runs response workflows.	Asks a user to confirm sensitive actions and runs only the tools given to it. Torq's audit log records every action with the user who asked. Torq's model or the customer's own subscription, in US, EU, or Japan regions.	kb.torq.io (1), kb.torq.io (2), kb.torq.io (3), kb.torq.io (4)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
AI SOC Analyst (Prophet Security).	Triage, investigate, respond	Investigates every alert with a plan built for that alert, and sends its findings to case management.	Takes only actions the customer approved, each previewed and backtested first. Documents every query and reasoning step. Single-tenant with a bring-your-own-key option, and setup asks for read-only access.	prophetsecurity.ai (1), prophetsecurity.ai (2), prophetsecurity.ai (3)
Intezer AI SOC (Intezer).	Triage, investigate, respond	Triages every alert by combining forensic analysis, such as sandboxing and memory analysis, with AI models, and closes false positives automatically.	Remediation can be automated with a person's explicit approval, and any verdict can be challenged. Experts review a random 5% of alerts each week, stated at 95% confidence with a margin under 2%. Models and residency are not documented.	intezer.com (1), intezer.com (2)
CognitiveSOC (Conifers.ai).	Author, prove, tune, ship, triage, investigate, respond	Agents investigate alerts to a verdict and a remediation plan. Others write, deploy, and tune detections, testing rule changes in a sandbox.	The customer picks a person in the loop, on the loop, or full autonomy. Each conclusion lists the queries run and the data touched. Runs in Azure in a chosen region, and reads from and writes to the customer's tools.	conifers.ai (1), conifers.ai (2), conifers.ai (3)
Qevlar AI.	Triage, investigate, respond when an analyst starts it	An orchestrator plans the steps of an investigation and model agents carry them out. Each produces a verdict, a report, and a containment or tuning recommendation.	Analysts review and start remediation. Every step and every queried observable is visible. Hosted in the EU on Google Cloud, with no training on customer inputs.	qevlar.com (1), qevlar.com (2), qevlar.com (3)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Command Zero.	Triage, investigate	Runs investigations as sequences of curated questions that query the connected sources at the time, without ingesting their data.	Connects through read-only APIs, logs every decision, and replays investigations. Analysts can take over any autonomous run. Model and residency are not documented.	commandzero.ai

AI for detection engineering

Products that apply a model to writing, translating, testing, or tuning rules. None of them publishes an accuracy figure for the rules it writes.

Product	Stages and functions	What it does	Controls it documents	Sources
AI Detection Engineer (Prophet Security). Announced 2026-07-30 for its AI SOC Analyst customers.	Ground in part, author, prove, tune, ship	Maps coverage from the customer's own detections, drafts detections for the gaps, and tunes noisy rules in Splunk, Sumo Logic, or Microsoft Sentinel. Each change is backtested against the customer's recent history.	Suggests by default, and the team approves version-controlled changes before they deploy. Autonomy can be raised per category of work. The model is not named.	prophetsecurity.ai (1), prophetsecurity.ai (2)
Anvilogic.	Author, tune, ship, triage, investigate	A detection layer over Splunk, Snowflake, and Databricks. Its agents plan detections for coverage gaps, write the queries, and deploy them, and users accept or dismiss tuning suggestions.	Approval steps, roles, and budgets limit what agents run, with version history and rollback. Its chat assistant uses OpenAI-hosted models, with no training on raw customer data.	anvilogic.com, public-docs.anvilogic.com (1), public-docs.anvilogic.com (2)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Uncoder AI (SOC Prime).	Extract, author, ship	Writes Sigma rules or indicator queries from pasted threat-report text, translates Sigma into 48 query languages, checks syntax, and suggests ATT&CK tags.	Open-weight models hosted by SOC Prime, an opt-in OpenAI option for some functions, and no training on user data. Its validation checks syntax only, and the user reviews the output.	help.socprime.com (1), help.socprime.com (2), help.socprime.com (3), socprime.com
NARC (Tidal Cyber).	Extract	Turns threat reports, blogs, and incident documents into ATT&CK procedures linked to groups, software, and campaigns.	Analysts review and correct the extractions. Single-tenant with a choice of region on the enterprise tier. The model is not described.	tidalcyber.com (1), tidalcyber.com (2), tidalcyber.com (3)
Detection Rule Validation (Picus Security).	Ground, prove	Reads the live rule inventory from Splunk, Microsoft Sentinel, or QRadar, checks that each rule's log sources send events, and runs attack simulations to see which rules fire. Its AI maps unmapped rules to ATT&CK.	Integrates with read-only access and does not change rules. The customer applies any fix.	picussecurity.com

MCP servers for security tools

Several servers expose tools that change the tenant as soon as they run. Read each tool list, start a server read-only where it offers a flag, and give it an identity of its own, as section 10.3 asks.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
MCP Server for the Splunk platform (Splunk). A Splunkbase app, version 1.3.	Ground, author queries, triage, investigate	Runs SPL searches and saved searches, lists indexes and metadata, and generates or explains SPL. Its Enterprise Security tools list findings, investigations, and response plans.	A search may be refused past 1 minute or 1,000 events, or when it uses a command the server deems unsafe. REST writes are blocked, the Enterprise Security tools only read, and each tool needs a capability an administrator grants.	help.splunk.com (1), help.splunk.com (2), help.splunk.com (3)
mcp-security (Google). Apache 2.0 local servers, plus a managed remote server.	Ground, prove, tune, ship, triage, investigate, respond	Local servers cover the SIEM, SOAR, threat intelligence, and Security Command Center. Some tools change the tenant, such as creating rule exclusions, changing curated rule deployments, and ingesting events.	The remote server needs an IAM role, writes to Cloud Audit Logs, and can screen tool calls with Model Armor. The local servers run under the caller's Google Cloud credentials.	github.com, raw.githubusercontent.com (1), raw.githubusercontent.com (2)
Microsoft Sentinel MCP server (Microsoft). Hosted. In preview, with some tools generally available.	Ground, triage, investigate	Tool collections explore the data lake with KQL, list and fetch incidents and alerts, run Defender advanced hunting, and create and deploy Security Copilot agents.	Uses the caller's Entra identity and roles. The triage and exploration tools only query data, and deploying an agent writes.	learn.microsoft.com (1), learn.microsoft.com (2), learn.microsoft.com (3)
Agent Builder MCP endpoint (Elastic). Generally available since 9.3.	Ground, author, prove, triage, investigate	Serves Agent Builder's tools, among them ES|QL queries, index mappings, security alerts, a tool that writes an ES|QL detection rule from a description, and rule preview.	An API key gives every caller one shared identity, so Elastic advises scoping it to the needed indices and rotating it. Per-user OAuth is available on Serverless only.	elastic.co (1), elastic.co (2), elastic.co (3)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
falcon-mcp (CrowdStrike). MIT. Pre-1.0, maintained by CrowdStrike, and described as not an official product.	Ground, author, tune, triage, investigate, respond	Covers 28 Falcon modules, including detections, hosts, SIEM search, intelligence, exclusions, correlation rules, and SOAR, with tools that create, update, and delete exclusions and run workflows.	A --read-only flag registers only the tools that read. HTTP transports have no authentication unless an API key is set.	github.com
purple-mcp (SentinelOne). MIT. Maintained by SentinelOne and described as not an official product.	Ground, triage, investigate	Asks Purple AI questions, runs PowerQuery over the data lake, and reads alerts, vulnerabilities, misconfigurations, and assets.	Read-only, with one token scoped to an account or a site. It has no built-in authentication, so it belongs off shared networks or behind a proxy.	github.com
mcp-panther (Panther Labs). Apache 2.0.	Ground, triage, investigate	Runs data-lake SQL, lists schemas and log sources, updates alerts, and lists or disables detections.	Uses a scoped Panther API token, and the README advises starting with a read-only one. Disabling a detection is among its tools.	github.com
Wazuh-MCP-Server (gensecaihq, a community project). MIT.	Ground, triage, investigate, respond	Exposes 55 tools for Wazuh, 41 that read and 14 that act, such as blocking an IP, isolating a host, or disabling a user, each with a rollback.	Read-only by default. Write tools need a write scope and a confirmation in production, and each is audit-logged.	github.com
Sumo Logic MCP Server (Sumo Logic). Hosted. Generally available since 2026-07-30.	Ground, author, triage, investigate	Searches logs and alerts, creates and updates dashboards, and works with Cloud SIEM insights and detection rules.	Uses OAuth 2.0, and its tools respect existing permissions. The release note does not say which rule operations are allowed.	sumologic.com

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Open models trained for security

The model cards report knowledge tests such as CTIBench and CyberMetric. None reports a result for writing or translating rules.

Product	Stages and functions	What it does	Controls it documents	Sources
Foundation-Sec-8B and its variants (Cisco Foundation AI). The base model is Apache 2.0. The instruct and reasoning variants use the Llama 3.1 license, with Cisco's changes under Apache 2.0.	Extract, triage, investigate	Eight-billion-parameter Llama 3.1 models trained further on about 5.1 billion tokens of security text, with instruct, 64k-context, and reasoning variants. The cards report CTIBench scores.	The reasoning card says the model is unsuited to autonomous security decisions. The cards recommend an extra safety filter and human oversight.	huggingface.co (1), huggingface.co (2), huggingface.co (3)
Llama-Primus (Trend Micro AI Lab). MIT, plus the Llama 3.1 license.	Extract, investigate	Eight-billion-parameter Llama 3.1 models trained further on 2.77 billion tokens of security text, which Trend Micro publishes, then tuned on about 1,000 security tasks.	The card reports small losses on general benchmarks and has no limitations section.	huggingface.co, arxiv.org
SecureBERT 2.0 (Cisco AI). Apache 2.0.	Extract	A 150-million-parameter encoder trained on threat reports and code, with an entity model that tags indicators, malware, and vulnerabilities in text.	It labels and embeds text and generates none. Its entity labels do not include ATT&CK techniques.	huggingface.co (1), huggingface.co (2)

Open-source detection engineering projects

SigmaOptimizer is the one project here that tests a generated rule against captured logs and a benign baseline, the pattern section 18 asks for.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
SigmAIQ (AttackIQ). LGPL 2.1. Pre-release.	Author	Wraps pySigma with an optional LangChain toolkit for writing and translating Sigma rules with a model, and converts rules to 18 backends.	The README warns that it is pre-release and may have bugs.	github.com
SigmaOptimizer (Yusuke Nakajima). MIT. The original repository is unmaintained, and the author points to a UI version.	Author, prove, tune	Runs a command, captures the Windows and Sysmon logs, and has a model draft a Sigma rule from them. It tests the rule with Hayabusa against those logs and a benign baseline, then refines it.	Windows only. The default benign baseline is one bundled set of Windows 10 client logs.	github.com (1), github.com (2)
Security-Detections-MCP (MHaggis). Apache 2.0, per its README.	Author	Indexes about 8,200 detections from Sigma, Splunk, Elastic, KQL, Sublime, and CrowdStrike, so a model can search them by technique, CVE, or process, find coverage gaps, and open a draft pull request.	Local mode reads the repositories the user points it at. A hosted mode offers read-only tools behind a token.	github.com
TRAM (MITRE Center for Threat-Informed Defense). Apache 2.0.	Extract	Maps sentences in threat reports to ATT&CK techniques with fine-tuned SciBERT models, covering up to 50 common techniques, and lets users annotate and retrain.	Publishes no accuracy figures, and targets ATT&CK v13.	github.com
CTINexus (peng-gao-lab). MIT.	Extract	Uses a model to pull entities and relations out of threat-intelligence text, merge duplicates, and draw a knowledge graph, with OpenAI, Gemini, Bedrock, or a local Ollama model.	Its README gives no evaluation figures.	github.com, arxiv.org

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Atomic Red Team MCP (cyberbuff, a community project). MIT.	Prove	Lets an agent search Red Canary's Atomic Red Team tests, check a test against the schema, and optionally run one to produce telemetry.	Running tests stays off until an environment variable turns it on, and the README says to run them only in isolated machines.	github.com

Benchmarks and recorded attack data

CTI-REALM is the one public benchmark found that scores rules an agent writes from threat reports, and it covers Azure and KQL. Spectrum's DefendBench scores writing, tuning, and repairing rules on six SIEMs, but its tasks are private. No public benchmark found scores tuning, and GUIDE's labels are the nearest public data for triage.

Product	Stages and functions	What it does	Controls it documents	Sources
CTI-REALM (Microsoft Security AI). Evaluation code in inspect_evals. The dataset states no license.	Extract, ground, author, prove	Agents read threat reports, explore the data, and write detections over emulated Linux, AKS, and Azure attacks in a Kusto emulator. Scores weigh both the steps taken and the detections' results.	The authors say it centers on Azure and does not reproduce production-scale telemetry. The best of 16 models reached a reward of 0.637.	arxiv.org, github.com
DefendBench (Spectrum). Announced 2026-10-27. The tasks stay private to keep them out of training data, and Spectrum plans to release the simulated environment.	Extract, ground, author, tune	Agents work 100 tasks in threat research, rule writing, tuning, rule health, and data engineering on Splunk, Sentinel, Elastic, Google SecOps, CrowdStrike, and Databricks. Each task runs in a simulated enterprise built from real environments and passes only if it meets all of its checks. An agent grades some of the checks.	The best of 14 entries, each a model in a coding agent, passed 31 of 100. At 95% confidence that score fits a pass rate from about 23% to 41%, a range that takes in the top five entries. The post gives no pass rate for experts other than each task's author, and no counts of attacks kept or benign events removed on tuning tasks.	spectrum.security (1), spectrum.security (2)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
ExCyTIn-Bench (Microsoft). MIT code. The dataset states no license.	Ground, investigate	Agents answer 7,542 investigation questions by querying 57 log tables drawn from 8 incidents.	Needs about 33 GB of disk, and the best model in the paper reached a reward of 0.606.	github.com, arxiv.org
CyberSOCEval (Meta and CrowdStrike). MIT code, with data licensed per folder.	Extract, investigate	Questions on malware analysis and on reasoning over threat-intelligence reports, built from sandbox output and CrowdStrike reports.	Its paper says current models are far from saturating it.	github.com, arxiv.org
CTIBench (Rochester Institute of Technology). CC BY-NC-SA 4.0.	Extract	Five threat-intelligence tasks, among them mapping CVEs to CWEs, scoring CVSS, and extracting ATT&CK techniques from reports.	The extraction and attribution sets are small, at 60 and 50 items.	github.com, arxiv.org
GUIDE (Microsoft Security Research). CDLA 2.0, per its paper.	Triage, tune	About one million incidents from 6,100 organizations, each labeled true positive, benign positive, or false positive by the customer's analysts.	Its labels are per incident, so it tests triage and leaves rules untested. Microsoft's guided-response paper publishes a full confusion matrix for one region's test split.	arxiv.org, kaggle.com
SIR-Bench (Amazon Web Services researchers). Preprint, 2026-04-13. The authors plan to release the test cases, labels, and replay framework.	Triage, investigate	794 cases built from 129 anonymized incident patterns, replayed in cloud environments for an incident response agent to triage and investigate. 475 are true positives and 319 false positives.	Its results give all four cells for the authors' agent, which missed 14 of 475 true positives and raised 85 false alarms on 319 false positives.	arxiv.org (1), arxiv.org (2)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
ALERT-BENCH (Virginia Tech, Google, and North Carolina State University researchers). Preprint, 2026-10-07. No release of the benchmark is stated.	Triage	Replays the AIT-LDSv2 log dataset through a Wazuh SIEM, producing 225 alerts tied to a multi-stage attack and 1,022 benign ones for an agent to triage.	Reports the false-negative rate beside the share of alerts escalated. Five approaches on GPT-4.1 each missed at least 40.4% of the attack alerts, and the authors' AIDA missed 7 of 225 while escalating 18.4% of all alerts.	arxiv.org (1), arxiv.org (2)
CyberSecEval 4 (Meta). MIT.	Any stage	Measures a model's help with attacks, its false refusals, prompt injection, and code-interpreter abuse, among other risks.	Its prompt-injection tests apply to any agent that reads untrusted text.	github.com
Security-Datasets (OTRF). MIT.	Ground, prove	Recorded attack and benign telemetry from simulated techniques, mapped to ATT&CK. The library on this site replays its rules against these recordings.	Each recording covers one procedure, so a rule's silence on one says little.	github.com
attack_data (Splunk). Apache 2.0.	Ground, prove	Raw attack logs filed by ATT&CK technique, most produced with Attack Range, ready to replay into a SIEM.	More than 9 GB through Git LFS, and a single dataset can be fetched alone.	github.com
BOTSv3 (Splunk). CC0.	Investigate, prove	About 320 MB of indexed Splunk data across about 100 sourcetypes, built for a capture-the-flag in 2018.	It needs the Splunk app versions it was built with.	github.com

Defenses against planted instructions

Each is tested on chat and agent benchmarks. None of the pages read for this table reports a test on instructions planted in log fields, the attack section 10.3 assumes.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Product	Stages and functions	What it does	Controls it documents	Sources
Llama Prompt Guard 2 (Meta). Llama 4 license.	Any stage that reads untrusted text	A small classifier that flags text trying to override instructions. Its card reports recall of 97.5% at a 1% false-positive rate on English text.	It reads 512 tokens at a time, so long inputs must be split, and the card says adaptive attacks can succeed.	huggingface.co
LlamaFirewall (Meta). MIT.	Any stage that reads untrusted text	Combines Prompt Guard, a model that audits an agent's reasoning for goal hijacking, and checks on generated code.	Its paper calls the reasoning audit experimental, and the framework needs a model to run.	github.com, arxiv.org
Meta SecAlign (Meta). Llama 3.1 and 3.3 licenses for the models.	Any stage that reads untrusted text	Llama models fine-tuned to ignore instructions inside untrusted data placed in a separate input role. On the 8B model, attack success on InjecAgent fell from 15.1% to zero.	It works only when untrusted text goes in the input role, and adaptive attacks still succeed at times.	huggingface.co, arxiv.org
CaMeL (Google Research). Apache 2.0. Research code.	Ground, triage, investigate, respond	Takes control flow and data flow from the trusted request, so untrusted data cannot change what the program does, and enforces data-flow rules at each tool call.	Its README calls it a research artifact that likely contains bugs and is unmaintained.	github.com, arxiv.org
NeMo Guardrails (NVIDIA). Apache 2.0.	Any stage that reads untrusted text	Programmable checks on input, dialog, retrieval, tool calls, and output, including jailbreak and injection detection.	The README says its built-in checks may not suit a given production use.	github.com
Agent Scan (Snyk). Apache 2.0.	Before an agent's tools are trusted	Scans MCP server configurations, tool descriptions, and agent skills for injection, tool poisoning, destructive tools, and exposed secrets.	Sends tool names and descriptions, with secrets redacted, to Snyk's API. Scanning can start local MCP servers, so it belongs in a sandbox.	github.com
MCP Scanner (Cisco AI Defense). Apache 2.0.	Before an agent's tools are trusted	Scans MCP tools, prompts, resources, and server code with YARA rules, a model judge, and heuristics.	Static scanning works offline, and the API engine needs a Cisco subscription.	github.com

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

19.7 Reading an accuracy claim

A triage agent's record has four cells. A true positive is a real attack it escalated, and a false positive is benign activity it escalated. A true negative is benign activity it closed, and a false negative is a real attack it closed. The false-close rate of 19.5 divides the false negatives by every real attack the agent saw.

Common figures hide the false negatives. A share of alerts closed automatically counts true and false negatives together, so every missed attack sits inside it. An agreement rate counts the alerts where agent and analyst made the same call, and both can miss the same attack. A sample of closed alerts measures the share of closes that were wrong, which stays small when benign alerts dominate. With a hundred benign alerts for each attack, an agent that closes half the attacks and nine in ten benign alerts makes about 0.6% of its closes wrong.

Only real attacks show how often an agent misses one. An agent that misses none of 30 real attacks may still miss up to 1 in 10, at 95% confidence. One that misses none of 100 may still miss up to 3 in 100. Eleven attacks with none missed leave room for a miss rate of 24%.

Ask any maker for all four counts for each alert class, from the team's own alerts during the trial, with the number of real attacks stated. The table lists what makers and the researchers studying their products publish, read on 2026-10-18, with the most complete first.

Publisher	What it publishes	What it leaves out	Sources
SIR-Bench (Amazon Web Services researchers, 2026)	Counts that give all four cells for the authors' own agent on 794 replayed cases. It caught 461 of 475 true positives and rejected 234 of 319 false positives.	A commercial product and production alerts. The cases are not yet released.	arxiv.org (1), arxiv.org (2)
Copilot Guided Response (Microsoft researchers, 2024)	A full confusion matrix for one region's test split of GUIDE, real incidents graded true positive, benign positive, or false positive. Macro-F1 is 0.87, and misgrading a true positive as benign or false positive is rare, at 2.4%.	Results from the triage agent Microsoft ships. The model in the paper is a research system.	arxiv.org (1), arxiv.org (2)
ALERT-BENCH (Virginia Tech, Google, and North Carolina State University researchers, 2026)	False-negative rates beside the share of alerts escalated, on 225 attack and 1,022 benign alerts. Every approach studied missed at least 40.4% of the attacks, and the authors' AIDA missed 7 while escalating 18.4% of alerts.	A full matrix for every system, a second attack scenario, and a release.	arxiv.org (1), arxiv.org (2)

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Publisher	What it publishes	What it leaves out	Sources
Charlotte AI reasoning triage (CrowdStrike researchers, 2026)	Precision and recall for each verdict on 42,686 real Windows endpoint detections kept out of training. Benign calls tuned to 98% precision on validation data were 90.8% precise on the test set, so about 1 in 11 was a real attack. Malicious calls were 98.9% precise, and overall accuracy is 82.6%.	Counts and the share of detections that were malicious. The product claims more than 98% agreement with Falcon Complete's analysts, which leaves misses both made uncounted.	arxiv.org, crowdstrike.com (1), crowdstrike.com (2)
Qevlar AI (2025 test)	99.8% accuracy, 0.3% false positives, and 0.1% false negatives on 5,000 email alerts that analysts at four organizations had reviewed in January 2025. The analysts themselves scored 97%, 4%, and 2%.	What each percentage is out of and how many alerts were malicious. The three figures do not add up on any one basis.	qevlar.com
Intezer (2024 review)	Production rates for 2024 across over 5.4 million alerts in 500+ customer environments, from a sample of about 5%. Accuracy was 93.45% on the 3.81% of alerts it escalated and 97.68% on the 68.40% it closed, at 95% confidence with a margin under 2%.	Counts, a definition of accuracy, and how many of the wrong closes were real attacks.	intezer.com (1), intezer.com (2)
Phishing Triage Agent trial (Microsoft, 2025)	The agent's precision and recall, both 100%, on 93 user-reported emails, 11 of them malicious.	Enough attacks to bound the miss rate, which could still be 24%.	arxiv.org (1), arxiv.org (2)
Exaforce (2025 review)	98% accuracy across 1.29 million alerts in 2025, defined as AI recommendations that matched outcomes confirmed by human review.	How many alerts were reviewed. The same post says 98% of alerts were benign or false positives after review, so closing every alert would score about the same.	exaforce.com
Prophet Security	99.8% agreement with the analysts of one Fortune 500 manufacturer across 12,000 investigations, and a 98.5% accuracy rate on its product page.	How agreement and accuracy are measured, and any count of misses.	prophetsecurity.ai (1), prophetsecurity.ai (2)
Conifers.ai	Accuracy above 99% across roughly 500,000 investigations, checked against the ground truth of each customer's environment.	A definition of accuracy, a period, and any split by verdict.	conifers.ai

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Publisher	What it publishes	What it leaves out	Sources
ReliaQuest GreyMatter	99.4% accuracy on routine tier 1 and tier 2 alerts.	A definition, a sample, a method, a date, and any word on misses.	reliaquest.com
Torq	Auto Triage accuracy that rises from roughly 94% to 99% within weeks.	A definition, a sample, or a method.	torq.io
Elastic (agentic SOC benchmark)	Scores from 1 to 10 for each model on eight intrusions and roughly 95 alerts, graded against an answer key Elastic built, with misses marked down.	Precision, recall, and any result from customer alerts.	elastic.co
Splunk (Cisco, .conf26)	The triage agent's suggested dispositions at the .conf26 event: 44% true positive, 42% benign positive, 6% false positive, and 7% with no suggestion.	Whether the suggestions were right. Cisco says to read the figures as observations from the event environment.	blogs.cisco.com
Command Zero	A 95 to 98% auto-triage rate.	Any accuracy figure. A share handled automatically counts the misses with the correct closes.	commandzero.ai
Google Security Operations	Over 5 million alerts investigated by its triage agent, each in about 60 seconds.	Any accuracy or miss figure.	cloud.google.com
Sumo Logic	That senior analysts regularly review a sample of the agent's verdicts.	Any result from that review.	sumologic.com
Dropzone AI (quality control post)	Definitions of precision and recall, tracked as rolling averages and checked with ISO 2859-1 sampling.	Any figure from that tracking. Simbian's ranking reports that Dropzone's CEO cited a false-negative rate below 1% in a press interview.	dropzone.ai, simbian.ai

A precision tuned on one set of alerts can slip on the next. In CrowdStrike's paper, benign calls tuned to 98% on validation data reached 90.8% on the test set, which is why 19.4 keeps sampling closes after promotion.

Palo Alto Networks, SentinelOne, and Panther publish volume and speed figures for their triage agents, and this reading found no accuracy figure from them. It found none for 7AI, Radiant Security, Arctic Wolf, or Expel either.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Simbian's 2026 ranking of AI SOC vendors says almost no vendor publishes a measured false-negative rate, and that Simbian does not publish one either.

A benchmark's pass rate needs the same reading. Ask how people other than each task's author score, and how wide the interval is. The best score on DefendBench, 31 of 100 tasks, fits a true pass rate anywhere from about 23% to 41% at 95% confidence. A tuning score needs the four counts as well. Ask for the attack events a tuned rule still catches and those it lost, and the benign events it still matches and those it dropped.

The tables in 19.6 list what each maker documents on the date given. Products change monthly, so check a claim against its linked page before relying on it.

The author has not run any product in those tables. The questions in 19.3 and the steps in 19.4 are how a team finds out for itself.

The thresholds in 19.4 are defaults the author chose, and the outside checks page lists them as untested.

20. Borrowed practices

Source	Where it does work
Palantir Alerting and Detection Strategy Framework	ads.md section names per detection, plus a Retirement record section this framework adds, so a record has 11 sections
Sigma rule status field	stage marker on every rule file; the Sigma rule itself is the single source each query language is generated from
MITRE CTID Summiting the Pyramid	robustness input, 5 levels (anchor wording paraphrased)
MITRE ATT&CK	technique field in score.yml, validated by lint, which also accepts ATLAS ids; tactic derived by metrics.py from the pinned map. The coverage app on the site tracks Enterprise, ICS, and Mobile separately and follows MITRE's releases. Pinned 20.0 (Enterprise) and 20.0 (ICS) on 2026-10-16.
MITRE ATLAS	technique ids for attacks on AI systems, accepted in score.yml beside ATT&CK ids; the ATLAS and D3FEND page lists the techniques ATLAS ties to a mitigation that detects. Release 2026.09, read through tools/atlas_extract.py into attack/atlas-2026.09.json.
MITRE D3FEND	the defensive techniques D3FEND links to each Enterprise and ICS technique, listed in the coverage map's technique detail by the artifact each one analyzes. Release 1.6.0, read through tools/d3fend_extract.py into attack/d3fend-1.6.0.json.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Source	Where it does work
MITRE CTID Top ATT&CK Techniques	threat_relevance anchor 3
SAFe Weighted Shortest Job First	formula shape: value over cost
Elastic Security Labs, Cloud Threat Emulation methodology	The emulation section: start modes, scope layers, plan-before-execute, foothold realism, telemetry verification, coverage outcomes, teardown. Part 1, published 21 September 2026, Terrance DeJesus and Bryan Porras Blanch.
MITRE Center for Threat-Informed Defense, Adversary Emulation Library	The plan-first shape of an emulation, and the micro and full scope layers. Public repository, read September 2026.

Sigma's status field does the work of a stage label from build on, with deploy and tune sharing stable. Section 17 lists the status and the CI jobs for each stage.

20.1 Open items

These get checked against their primary sources before each release:

- MITRE ATT&CK: re-run tools/attack_pin.py on each ATT&CK release and review the map diff; v19.2 renamed Defense Evasion to Stealth and added Defense Impairment; v20.0 added nine techniques, among them direct and indirect prompt injection, and moved none, and ICS techniques now include T16xx ids
- MITRE ATLAS: re-run tools/atlas_extract.py and the kit's tools/attack_pin.py --atlas-only on each ATLAS release and review the diff. The five mitigations the extract counts as detecting are named in the script and need checking against each release
- MITRE D3FEND: re-run tools/d3fend_extract.py on each D3FEND release. Release 1.6.0 maps to ATT&CK 19.0, so check which revoked ids moved to their replacements and whether a later release maps the techniques ATT&CK 20.0 added
- MITRE CTID Summiting the Pyramid: verify level wording and current version against the CTID primary source
- MITRE CTID Top ATT&CK Techniques: verify current list and calculator inputs against the CTID primary source
- Sigma rule status field: verify the status value list against the current Sigma specification
- Detection Engineering Maturity Matrix: map stages to dimensions from the primary source; unmapped until verified
- ACSC and CISA priority logs for SIEM ingestion (2025): map the 14 source categories to the telemetry inventory template

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Appendix A. Definitions of done and stalls

Stage	You may leave when	The usual stall	If it can't be met
1. Intake	The request is written down. It names the behavior, the asset, and who asked. Six scores are recorded with one line of evidence each.	Requests arrive as nothing more than an alert name, with no note about the threat behind it, so you build them in the order they showed up.	A candidate whose scores can't be evidenced stays at intake, and the missing evidence becomes a question for the next intake meeting. A candidate with telemetry readiness 0 is blocked, and its log source request goes to the platform owner.
2. Hypothesis	The record carries a one-sentence hypothesis: when an attacker does X, log source Y records Z. It names the log source. It names one thing the rule cannot see.	The hypothesis describes one tool's fingerprint. When that tool changes, the detection goes quiet.	If no log source you collect records the behavior, the candidate goes back to intake with telemetry readiness 0 and a log source request. If the only hypothesis anyone can write names one tool's fingerprint, rescore its robustness at intake before going on.
3. Research	The behavior has been emulated from a believable starting identity. The events it produced are saved as fixtures. Every field the rule will use was read from a real event. Its normal rate has been counted over 7 days of production data. The lab is torn down and the teardown verified by run tag.	Field names come from documentation or memory, so the detection is almost right and never fires. The other common failure is skipping the 7-day count, which spends the first week in production on noise the count would have shown.	If the events never reach the SIEM, or a field the rule needs arrives unparsed, the candidate goes back to intake with its telemetry readiness lowered and a log source request. If the 7-day count shows the behavior is routine here, rewrite the hypothesis before building anything.
4. Build	The rule file passes lint. It converts cleanly into every target in the registry. It has one event it must match and one it must ignore.	You write the detection straight in the SIEM's own query language. Then it runs on one product only, you can't compare its changes against an approved version, and it can't move when the SIEM changes.	If a target can't express the rule, mark that target convert-only in targets.yml and say so in the record. If Sigma can't express it at all, as with a correlation across events, write it as a query file beside the same record and fixtures, and review that file directly.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Stage	You may leave when	The usual stall	If it can't be met
5. Validate	The fixtures run against a real instance of every engine that has one, and each returns what it should. Every converted query matches the approved copy, or the difference is explained and accepted. A second engineer has approved the change. On a one-person team the approval comes from a colleague outside the team, a peer elsewhere, or the author after a 24-hour wait.	Whenever the conversion output changes, someone overwrites the approved copy to match it. After that the comparison can't fail, so it catches no changes.	If a fixture fails or the reviewer asks for changes, the pull request goes back to build and stays open. If the reviewer rejects the hypothesis itself, the detection goes back to hypothesis.
6. Deploy	The detection is live with alerting off. Over its first 7 days live it returned no more alerts than the threshold on its record, and at least one if research counted the behavior in its baseline. A pull request then moves it to tune, and the move switches alerting on.	Alerting goes on before the 7-day check, and the SOC pays for the noise in triage time.	If the first 7 days run over the threshold, alerting stays off and the detection stays at deploy. The fix is a pull request that goes back through validate, and the 7 days start again from its merge. If they return nothing though research counted the behavior, check the fields and the index against a real event.
7. Tune	Over the last 30 days its precision is at or above its floor. Every tuning change went through a pull request and back through validate, and the query running in production matches the approved copy.	Someone tunes the detection by adding exclusions in the SIEM's own interface, so the rule in the repo no longer matches the rule running in production.	If precision stays under the floor for two 30-day periods in a row, propose retirement. If an incident or an exercise shows the detection missed what it claims to catch, fix this detection. The missed events become a positive fixture, and the change goes back through validate and its 7-day count.
8. Retire	Status is deprecated and the detection is off in production. A retirement note records why, and any replacement is at intake with a link to that note.	Nothing is ever retired, so analysts keep triaging alerts from detections that no one owns or understands anymore.	A retirement is not undone. If the technique still matters, the replacement is a new detection at intake with a link to the retirement note.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Appendix B. Score anchors

threat_relevance (T)

1. No public reporting in the last 24 months.
2. Public reporting in the last 24 months, but nothing tying it to your sector.
3. On MITRE CTID's Top ATT&CK Techniques list, or near the top of a vendor's annual threat report.
4. Named in reporting about your sector in the last 12 months (an ISAC, a regulator, or a peer).
5. Seen in your own incidents or red team findings in the last 12 months.

impact (I)

1. Reconnaissance, or a low-consequence action on a single host.
2. One host compromised, and existing controls contain it.
3. Data access or code execution on a business-critical system.
4. A step toward the crown jewels: lateral movement or privilege escalation heading for tier-0 identity (domain admin and the systems that control it), OT, or payment systems.
5. Direct action on tier-0 identity, an OT safety system, or a payment system.

robustness (B)

1. Matches a throwaway indicator: a file hash, IP address, domain, or filename the attacker can change at will.
2. Matches one tool the attacker brings along: its strings, its arguments, the files it drops. A different tool evades it.
3. Matches how a built-in system program is abused: the arguments or behavior of something that was already on the machine.
4. Matches something most ways of doing the technique share, such as an API call or a sequence of events.
5. Matches something every way of doing the technique must produce.

telemetry_readiness (R)

1. The log source is not collected at all.
2. Collected, but the event type or field the rule needs is missing.
3. Collected, but sampled, truncated, or kept for less than 30 days.
4. Collected in full and kept 30 days or more, but the fields the rule needs are not parsed out.
5. Collected and parsed, but not enriched with who the user is or what the asset is.
6. Collected, parsed, enriched, and covered by the silence alert (DET-0000).

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

build_effort (E)

1. Under 4 hours: adapting a public detection that already comes with test events.
2. 4 to 8 hours.
3. 8 to 16 hours.
4. 16 to 40 hours: needs a simulated attack or new field parsing.
5. Over 40 hours: needs new parsing plus a multi-step emulation.

run_cost (C)

1. Under 5 analyst minutes a day.
2. 5 to 15 minutes a day.
3. 15 minutes to 1 hour a day.
4. 1 to 3 hours a day.
5. Over 3 hours a day.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Appendix C. Metric definitions

Metric	Definition	Unit and aggregation	Stage	Kind
intake_to_score_days	Days between a request entering the backlog and receiving its first score.	days, median	intake	KPI
backlog_age_days	How long scored requests have been waiting with no one starting the hypothesis.	days, median	intake	KPI
readiness_at_hypothesis	Of the requests that reach hypothesis, the share whose data is already collected in full and kept 30 days or more (readiness 3 or higher).	percent, ratio	hypothesis	KPI
research_hours_per_rule	Hours logged in research for one detection.	hours, median	research	KPI
percent_with_tests	Share of detections past the build stage that have both a test event to match and one to ignore.	percent, ratio	build	KPI
first_pass_validation_rate	Share of pull requests that pass every validation check on the first run.	percent, ratio	validate	KPI
review_turnaround_hours	Hours from a pull request being marked ready to a second engineer approving it.	hours, median	validate	KPI
deploy_lead_time_days	Days from the merge that sets a detection's stage to deploy to the merge that moves it to tune and switches its alerting on.	days, median	deploy	KPI
post_deploy_replay_failures	Promotion checks in the quarter that failed, because the first 7 days live returned more alerts than the threshold or none at all when research counted some.	count, sum per quarter	deploy	KPI

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Definition	Unit and aggregation	Stage	Kind
detection_latency_median	Minutes from the timestamp on the event to the alert arriving in the queue. It is the floor under every response time you report, because nobody responds before the alert arrives. Scheduled searches inherit their interval, so a detection running hourly cannot beat 30 minutes on average.	minutes, median per detection, 30 days	deploy	KRI, red over 60 minutes
precision_30d	Of a detection's alerts in the last 30 days that an analyst closed with an outcome, the share that were true positives. A benign true positive, behavior that happened and was allowed, counts against precision. An alert closed as undetermined counts on neither side.	ratio, per detection; program median	tune	KPI
triage_minutes_median	Typical analyst minutes spent on one alert from the detection.	minutes, median	tune	KPI
minutes_per_true_positive	Analyst minutes spent for each true positive the detection produces: triage minutes divided by precision.	minutes, per detection; program median	tune	KPI
upkeep_minutes_per_detection	Engineering minutes spent in a month keeping live detections working (precision checks, tuning pull requests, query fixes after a field or schema change), divided by the number of live detections. It replaces the 20-minute default in the maintenance ceiling once a quarter has been measured.	minutes, monthly, across the live set	tune	KPI

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Definition	Unit and aggregation	Stage	Kind
rule_debt	Share of live detections that fired at least 10 times in 180 days and were never right. It counts the same detections the second retirement trigger names, so it is the share of the live set that should already be retiring.	percent, ratio	retire	KRI, red over 20%
rules_retired_per_quarter	Detections set to deprecated in the quarter.	count, sum	retire	KPI
robustness_weighted_coverage	Per ATT&CK tactic, each live detection counts as its robustness divided by 5. A level-5 detection counts as one, a level-1 detection as one fifth. Shown beside the plain detection count. A tactic covered only by easily evaded detections needs five of them to count as much as one level-5 detection. It is a sum, so it grows with every detection added. The coverage app's weighted share is the per-technique version: each technique scores only its strongest live detection, so it tops out at 100%.	weighted count, per tactic	program	KPI
incident_detection_rate	Share of confirmed incidents in the period that a live detection alerted on first. The rest were found by a user report, a third party, an audit, or luck. Each incident this misses goes to intake as a candidate, with the incident as its evidence.	percent, per quarter	program	KPI
silent_source_hours	Hours in the quarter that a log source in the inventory sent nothing for longer than its silence threshold. Report the sources behind live detections first.	hours, per quarter, per source	program	KRI, red over 24 hours

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Definition	Unit and aggregation	Stage	Kind
priority_technique_gaps	Share of the program's priority techniques with no live detection. The quarterly review keeps the priority list: the techniques used by the threat groups the team tracks, or the top of the threat-led backlog.	percent, per quarter	program	KRI, red over 40%
misses_by_cause	Confirmed incidents and exercises in the quarter that no live detection caught first, counted by the miss code the detection review recorded.	count, per quarter, by miss code	program	KRI, red the same code in two quarters running
open_detection_fixes_age	Days the oldest fix from a detection review has waited without passing validate, counted from the day the incident was confirmed.	days, oldest open, at quarter end	program	KRI, red over 60 days
false_close_rate	Share of alerts later confirmed malicious that triage had closed as benign, by a person or an agent. Section 19's ai_false_close_rate is the same count for an agent alone.	percent, per quarter	program	KRI, red a wrong close on a detection rated impact 4 or 5
time_to_scope_hours	Hours from triage escalating an alert to the case recording the incident's scope: which hosts, accounts, and data.	hours, median per quarter	program	KPI
time_to_contain_minutes	Minutes from an incident being confirmed to the first containment action completing.	minutes, median per quarter	program	KPI
notifications_on_time	Share of required notifications, to leadership, customers, or a regulator, sent within their deadline.	percent, per quarter	program	KRI, red any regulator or customer deadline missed
playbook_coverage	Share of alerts in the quarter whose detection has a live playbook at tier 1 or higher.	percent, per quarter	program	KPI

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0

Metric	Definition	Unit and aggregation	Stage	Kind
playbook_minutes_saved_per_alert	The manual baseline minutes on the playbook's record, minus the analyst minutes an alert takes with the playbook running, per alert class.	minutes, median per quarter	program	KPI
playbook_actions_unattended	Actions a playbook completed with no person in the loop, counted by ladder tier. A tier 4 count above zero names the expendable asset the record allows.	count, per month, by tier	program	KPI
playbook_actions_reverted	Playbook actions a person refused in propose mode or undid after they ran. Each one is a tuning pull request.	count, per quarter	program	KRI, red a reverted tier 4 action, or a refused close on an impact 4 or 5 alert
incidents_without_followup	Confirmed incidents that produced no intake candidate, no tuning pull request, and no log source request within 30 days of confirmation.	count, per quarter	program	KRI, red the same responder twice, or any incident in two quarters running

DETECT by Habibullah Tora, https://detect.htora.dev, licensed CC BY-NC 4.0. Version 0.136.0.

DETECT by Habibullah Tora, detect.htora.dev, licensed CC BY-NC 4.0 · version 0.136.0