detect

A detection lifecycle with countable gates and a scored backlog

Eight stages, each with an exit you can count. Six numbers that put the backlog in order. Fifteen metrics, each defined once. One data file that a site renders and a repository runs.

Habibullah Tora

Version 0.3.0, September 19, 2026

1. Summary

detect is a framework for running a threat detection engineering program. It has 8 lifecycle stages, each with one exit gate you can count, 6 scoring inputs that rank the backlog by value against cost, and 15 metrics, each defined once. The framework is a single data file, framework.json; the website at detect.htora.dev renders it, and a companion repository runs it, with a CI job named after every gate and a score computed for every rule. Version 0.3.0 ships one complete example detection through the whole pipeline.

2. What the framework assumes

The framework rests on four premises about detection programs. It does not claim they hold everywhere; it is built for the programs where they do.

1. The backlog of detection ideas is always longer than the team's capacity, so the order of work is a decision, and it should be made with numbers that can be argued with.
2. Every live rule has a running cost, paid by analysts in triage minutes, and that cost is rarely written down next to the rule.
3. A count of rules per tactic overstates coverage, because a rule that matches a tool's filename and a rule that matches the behavior itself count the same on a heatmap.
4. Rules leave production far less often than they enter it, so the analysts end up triaging alerts from rules that nobody owns.

3. Design decisions

Six decisions shape everything else.

One schema drives the site and the repository. Stage names, gate wording, metric definitions, and the score model live in framework.json. The website renders that file, and the repository's lint step validates every rule record against it. When the framework changes, both change from the same edit.

Every stage has one exit gate that can be counted. "The rule is validated" is not a gate. "Replay passed in both engines, the converted query matches the approved copy, and a second engineer approved the change" is a gate, because a CI run can report whether it happened.

Prioritization is a scored backlog. Each candidate carries six numbers and a note saying where each number came from; CI computes the score and refuses a score typed in by hand. The site's prioritize page runs the same formula in the browser so a leader can move the weights and watch the order change.

The Sigma status field marks the stage. A rule with status experimental is in build, test means validate, stable means deploy or tune, deprecated means retired. Nothing else has to be kept in sync to know where a rule stands.

Detection zero is the log source silence alert. Every other rule depends on its data arriving, so the first rule a program ships is the one that notices when a source goes quiet. Coverage by that alert is one condition for the highest readiness score.

Coverage is weighted by robustness. Each live rule counts as its Summiting the Pyramid level divided by 5, so seven rules that match tool names can add up to less coverage than four rules that match the behavior.

4. The lifecycle

Eight stages on 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. Two loops close it: a tuning change is a pull request that returns to validate, and a retired rule's replacement enters intake with a link to the retirement note.

Stage	What it is for	Exit gate, counted	Owner	Metrics
1. Intake	Write down a request for a new detection with enough context that it can be scored.	6 inputs, 6 evidence lines, 1 score	Detection engineering lead	intake_to_score_days, backlog_age_days
2. Hypothesis	Write one sentence saying what the rule will catch, which log source it needs, and what it will miss.	1 hypothesis, 1 data source, 1 blind spot	Detection engineer	readiness_at_hypothesis
3. Research	Confirm the thing the rule looks for actually appears in your logs, then measure how often it appears during normal business.	1 reproduction, 1 fixture capture, 7-day baseline	Detection engineer	research_hours_per_rule

Stage	What it is for	Exit gate, counted	Owner	Metrics
4. Build	Write the rule in Sigma and the tests that prove it.	1 lint pass, N conversions, 2 fixtures	Detection engineer	percent_with_tests
5. Validate	Prove the rule fires on the events it should catch, stays silent on the ones it should ignore, and that its converted query still matches the version a reviewer last approved.	N engine passes, N golden diffs, 1 approval	Second detection engineer (reviewer)	first_pass_validation_rate, review_turnaround_hours
6. Deploy	Put the rule into production with alerting switched off, run it over the last 7 days of real data, and only then switch alerting on.	1 seven-day replay, 1 threshold check, 1 enable	Detection engineering lead	deploy_lead_time_days, post_deploy_replay_failures
7. Tune	Keep the share of true alerts above the agreed floor as the environment changes.	1 precision check per 30 days, 0 changes outside the repo	Detection engineer with the SOC lead	precision_30d, triage_minutes_median, minutes_per_true_positive
8. Retire	Remove rules whose alerts cost more analyst time than they are worth.	1 status change, 1 disable, 1 record, 0 or 1 replacement	Detection engineering lead	rules_retired_per_quarter, rule_debt

The full gate wording and the mistake that usually stalls each stage are in Appendix A.

5. Prioritization

5.1 The six 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?

Input	Scale	Question
robustness (B)	1 to 5	How hard is it for an attacker to slip past this rule 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 rule 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 rule consume? Expected alerts per day multiplied by the typical minutes to triage one.

Robustness uses the five Summiting the Pyramid levels: the higher the level, the more an attacker has to change to get past the rule. Run cost is the alerts expected per day multiplied by the typical minutes to triage one. The 1 to 5 wording for every input is in Appendix B.

5.2 The formula

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

A readiness of 0 sets the score to 0 and marks the rule blocked. Scores round to two decimals. With every weight at 1 and k = 1, scores run from 0.06 to 7.50. The shape is Weighted Shortest Job First from SAFe, value over size, with readiness added as a gate because a detection cannot be built on data that is not there.

5.3 Four ways to lean

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 cannot see yet.
asset_led	wI=2	You know which systems matter most and how an attacker would reach them.	Only works with an asset inventory, which most teams lack.
telemetry_led	k=2	A new program that needs early wins from the data already in the SIEM.	Favors what your logs already show, like searching under the streetlight, and can leave the biggest threats uncovered.

A rule required by a regulation or an audit carries a mandate (framework, requirement, due date) instead of a preset. A rule 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 rule with the lower run cost. Mandated rules come with budget and deadlines, and often catch auditors better than attackers.

5.4 Evidence

Every one of the six inputs comes with one line saying where the number came from: an incident id, a report section, a query, a ticket. CI rejects a record missing any of them.

5.5 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 holds in all three.

5.6 The floor, and when a rule has to go

A rule's floor is the typical minutes to triage one of its alerts divided by the most analyst minutes the program will spend per true positive (120 by default). A rule that takes 30 minutes to triage must be right at least 25% of the time; one that takes 6 minutes, 5%.

Retire on any one trigger: its true-positive share stayed below its floor for two 30-day periods in a row; it fired at least once in 180 days and was never right; the log source it needs has been missing for 30 days straight, so its readiness fell to 0; the technique or the system it protects is gone, for example because the platform was decommissioned. It has not fired at all in 365 days. Either the data stopped or the hypothesis was wrong; review it before retiring it.

6. Where to begin

Three questions route a team to a path. Do you have a log source inventory: a list of every source feeding the SIEM, with who owns it, how long it is kept, whether its fields are parsed, and whether an alert fires when it goes quiet? Do you have the last 12 months of incidents, each tagged with the ATT&CK techniques the attacker used? How many analyst hours a week go to building and tuning detections?

A team with no inventory builds one, ships DET-0000 for every source, and then scores. A team with an inventory and no incident history takes threat relevance from public prevalence lists and sector reporting, scores every candidate with readiness 3 or higher, and ships the top 10. A team with both scores its own incidents at 5 and ships the top 10. Work on no more rules at once than your weekly hours divided by the hours one rule takes from build through validate. Use 8 hours per rule until you have measured your own number.

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 rule depends on its data arriving. A source covered by this alert is one step closer to a readiness score of 5.

7. Metrics

15 numbers, each defined once and owned by one stage. The repository produces the build-side ones from score.yml and the file layout; the rest come from what analysts record about each alert. The definitions are in Appendix C.

Coverage honesty is the chart the metrics page opens on. On a 30-rule sample program run through the repository's own metrics code, Execution has the most rules and Credential Access has the most weighted coverage:

Tactic	Live rules	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
Lateral Movement	2	1.4
Initial Access	2	1.2
Discovery	2	1.0

Tactic	Live rules	Robustness-weighted
Exfiltration	2	0.8
Evasion	1	1.0
Impair Process Control	1	1.0

8. The reference pipeline

The repository is a template: fork it, replace the example, point the secrets at a Splunk Cloud stack. Each detection is a directory holding rule.yml (Sigma), ads.md (the ten sections of the Alerting and Detection Strategy format), score.yml (six inputs, six evidence lines, CI-written score), test 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	none	The Goal, Categorization, Strategy Abstract, and Blind Spots and Assumptions sections of ads.md
3. Research	no rule file yet	none	The Technical Context section of ads.md; 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, post-deploy-replay, enable	A deploy record: version, date, SIEM, and the alert count from the 7-day run
7. Tune	stable	precision-check (scheduled)	A tuning pull request that cites the alert ids that prompted it
8. Retire	deprecated	disable	A retirement note in ads.md: reason, date, and the id of the replacement rule

Six tools, one per gate. Lint checks the layout, parses the rule, confirms the id and tag agree, confirms the Sigma status matches the stage, checks the ten ADS sections, confirms every named fixture exists, validates the score record, and rejects any ATT&CK technique id that is not in the pinned map. Convert produces SPL and KQL and fails when either differs from the approved copy. Score computes and sorts. Metrics fills what only the file layout can tell.

Package builds a Splunk app with alert actions off, then on. Replay runs the fixtures through a Splunk container and the Kusto emulator, and runs the 7-day production check against the threshold.

What has been executed on the example: lint, convert with the golden check (a tampered copy failed as intended), a wrong Sigma status failing lint, a bogus technique id failing lint, score, metrics, packaging with alert actions off and on, and dry runs of both replay engines. What has not been executed anywhere yet: the live Splunk and Kusto calls, the two container services in the workflow, and the AppInspect and ACS steps, which are placeholders.

9. What the framework borrows

Source	Where it does work
Palantir Alerting and Detection Strategy Framework	ads.md section names per rule
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; tactic derived by metrics.py from the pinned map Pinned 19.2 (Enterprise) and 19.2 (ICS) on 2026-09-12.
MITRE CTID Top ATT&CK Techniques	threat_relevance anchor 3
SAFe Weighted Shortest Job First	formula shape: value over cost

Open items, verified against primary sources before each release:

- attack: 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, and ICS techniques now include T16xx ids
- summiting the pyramid: verify level wording and current version against the CTID primary source
- ctid top attack techniques: verify current list and calculator inputs against the CTID primary source
- sigma status: 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 cisa priority logs 2025: map the 14 source categories to the telemetry inventory template

10. Limits

The defaults are starting values. The 120-minute cap per true positive, the 90-day mandate window, the 180-day and 365-day retirement windows, and the 8 hours per rule are numbers to replace with a program's own once measured.

Precision, triage minutes, and the retirement triggers need alert outcomes exported from the SIEM. The repository does not yet collect them; the precision check is a scheduled job to be written.

The repository holds one complete detection. The scoring model has been exercised on eight sample candidates and the coverage chart on a 30-rule sample, both generated through the repository's own code, but neither is a production program.

The mapping to the Detection Engineering Maturity Matrix is listed as an open item and is not claimed.

Appendix A. Gates and stalls

Stage	You may leave when	The usual stall
1. Intake	The backlog entry has a number for each of the six scoring inputs, a one-line note saying where each number came from, and a score computed from them.	Requests arrive as nothing more than an alert name, with no note about the threat behind it, so the team builds them in the order they showed up.
2. Hypothesis	The ADS document has a one-sentence hypothesis in the form 'When an attacker does X, log source Y records Z', names the data source, and names one thing the rule cannot see.	The hypothesis describes one tool's fingerprint. When that tool changes, the rule goes quiet.
3. Research	The behavior has been reproduced once in a test environment and the resulting log events saved as test fixtures, and the number of times it shows up during normal business has been counted over 7 days of production data.	The 7-day count is skipped, so the first week in production is spent tuning out noise the count would have shown in advance.
4. Build	The rule file passes the lint checks, converts cleanly into every query language listed in the repository's target registry, and has at least one test event it must match and one it must ignore.	The rule is written straight in the SIEM's own query language. It then runs on one product only, its changes cannot be compared against an approved version, and it cannot move when the SIEM changes.
5. Validate	The test events run through a real instance of every engine that has one, and each returns exactly the expected results; every converted query is identical to the approved copy, or the difference has been explained and accepted; and a second engineer has approved the change.	Whenever the conversion output changes, someone overwrites the approved copy to match it. The comparison then never fails, and never catches anything.
6. Deploy	The rule is live with its alert action off; running it over the last 7 days returned no more alerts than the threshold recorded in score.yml; and the alert action has then been switched on.	Alerting is switched on before the 7-day check, and the SOC pays for the noise in triage time.
7. Tune	Over the last 30 days, the share of the rule's alerts that were true positives is at or above its floor, and every tuning change went through a pull request and the validate gate.	Tuning is done by adding exclusions in the SIEM's own interface, so the rule in the repo no longer matches the rule running in production.
8. Retire	The rule's status is set to deprecated, it is switched off in production, a retirement note records why, and any replacement has been entered at intake with a link to that note.	Nothing is ever retired, so analysts keep triaging alerts from rules that no one owns or understands anymore.

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 walks past.
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).

build_effort (E)

1. Under 4 hours: adapting a public rule 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 a new log source brought into the SIEM.

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.

Appendix C. Metric definitions

Metric	Definition	Unit and aggregation	Stage
intake_to_score_days	Days between a request entering the backlog and receiving its first score.	days, median	intake
backlog_age_days	How long scored requests have been waiting with no one starting the hypothesis.	days, median	intake
readiness_at_hypothesis	Of the requests that reach hypothesis, the share whose data is already in the SIEM in usable shape (readiness 3 or higher).	percent, ratio	hypothesis
research_hours_per_rule	Hours logged in research for one rule.	hours, median	research
percent_with_tests	Share of rules past the build stage that have both a test event to match and one to ignore.	percent, ratio	build
first_pass_validation_rate	Share of pull requests that pass every validation check on the first run.	percent, ratio	validate
review_turnaround_hours	Hours from a pull request being marked ready to a second engineer approving it.	hours, median	validate
deploy_lead_time_days	Days from a rule being merged to its alert action being switched on.	days, median	deploy
post_deploy_replay_failures	Deploys in the quarter where the 7-day production run produced more alerts than the threshold.	count, sum per quarter	deploy
precision_30d	Of a rule's alerts in the last 30 days, the share that were true positives.	ratio, per rule; program median	tune
triage_minutes_median	Typical analyst minutes spent on one alert from the rule.	minutes, median	tune
minutes_per_true_positive	Analyst minutes spent for each true positive the rule produces: triage minutes divided by precision.	minutes, per rule; program median	tune
rule_debt	Share of live rules that fired at least once in 180 days and were never right.	percent, ratio	retire

Metric	Definition	Unit and aggregation	Stage
rules_retired_per_quarter	Rules set to deprecated in the quarter.	count, sum	retire
robustness_weighted_coverage	For each ATT&CK tactic, each live rule counts as its robustness score divided by 5 (a level-5 rule counts as one whole rule, a level-1 rule as one fifth), summed and shown beside the plain rule count. A tactic covered by many easily evaded rules scores low here.	ratio, per tactic	program

Appendix D. Version history

Version	Date	Change
0.1.0	2026-09-12	Framework data, score schema, scoring spec. Eight stages, two loops, six inputs, four presets, 15 metrics.
0.1.1	2026-09-12	Three-line grouping (Backlog, Build, Live) moved from the lifecycle page into the framework data.
0.1.2	2026-09-12	ATT&CK pinned at Enterprise 19.2 and ICS 19.2 through the repo's technique-to-tactic map; lint rejects unknown technique ids.
0.2.0	2026-09-12	Plain-English rewrite of every reader-facing string; robustness-weighted coverage definition corrected to match the code (weighted rule count, not a ratio).