PURETENSOR
INTELLIGENCE DISTILLED

Can Every Alert Rule Fire

Mutation-Testing 496 Production Alert Rules, and the 386 with Findings

Paper: PT-R-2026-058 v1.0

Author: PureTensor Ltd

Date: 21 October 2026

Status: Published

Abstract

An alert rule that cannot fire looks like coverage and provides none. We ran one analyser over every live alert rule on our two Prometheus estates, 270 on the cluster Prometheus and 226 on the monitoring-host Prometheus, 496 in all. For each rule it asked whether the expression can fire on production-shaped data and whether the rule's own unit tests would notice if the expression were broken, using live selector checks, join probes, fixture-realism grading, eight mutation operators, threshold headroom and routing.

At the 21 October baseline 110 rules were clean and 386 (78%) had at least one finding, 785 findings in total. Of the 2,191 mutants we generated, the existing tests killed 1,713 (78.2%). Of 2,896 test series, 457 (15.8%) carried a label that production does not have. Every rule on the cluster estate already had a fire case and a quiet case (20 monitoring-host rules lacked a quiet case), so the main problem was test quality, not missing tests.

A repair stage then assigned a verdict to 669 groups of findings across 118 units. Only 10 carried the rule-defect label, and 3 of those were added tests; the other 7 changed a rule, covering 4 rules. 389 were weak or unrealistic tests, 126 were analyser false positives, 129 were behaviour that is correct by design, and 15 were accepted residuals. After the fixes the mutant kill rate was 92.7% (2,035 of 2,196), test series with a wrong label were 5.1%, high-severity rules fell from 13 to 0, and clean rules rose from 110 to 310. Both after-scores were measured by the instrument the repairs were tuned against, so they show convergence on the grader, not held-out detection. Two of the nine critical findings that remain were wrong, and ten rules came out worse than they went in.

1. A Question Narrower Than "Is Monitoring Healthy"

We run two Prometheus estates: the cluster Prometheus with 270 alert rules in 26 rule files, and the monitoring-host Prometheus with 226. Both already had unit tests written for promtool, Prometheus's own rule checker and test runner. An earlier paper in this series covered a registry of alerting controls, not every rule.

The question here is estate-wide. For each of the 496 live rules, can it fire on production-shaped data, and would its tests fail if the rule were broken? "Can fire" is stronger than "the metric exists". A rule can reference a present metric and never trigger, because a selector pins a label

value the exporter never emits, because the two sides of a join share no label, because a threshold sits beyond anything the hardware produces, or because the test that supposedly proves the rule feeds it series no real exporter would send.

A tool that reports 386 rules with findings invites the reading that 386 rules are defective; most of this paper is about why that is wrong.

2. The Analyser

The analyser is a single Python program, run on the fleet's distributed compute layer because the mutation stage exceeded an operator workstation's memory. It parses each expression into a syntax tree and prints it back; all 496 expressions round-tripped unchanged. Per rule it runs eight checks.

First, the repository rule is compared with the loaded rule on the parsed expression, hold duration and labels. Second, selector liveness: every selector is queried against the live Prometheus over 14 days (cluster estate) or 30 days (monitoring-host estate). A selector that matches nothing is graded by role: critical in the path that makes the rule fire, medium in an exclusion or "or" branch, informational for an absence guard.

Third, join probes. For every binary operation between two vectors (an "and", an "unless", or arithmetic) it counts left series, right series and joined series now; again with the threshold filters stripped from both sides; and again at the last moment both sides were non-empty. It also compares the label names structurally. A structural mismatch on an "and" is graded critical, on other operators high, and a mismatch in values alone is medium.

Fourth, fixture realism: every series in a rule's test cases is checked against live label names and values, and graded by how the rule uses the label. Fifth, mutation testing: each rule is mutated in a reduced private copy of the test workspace and its own tests are rerun with promtool. A mutant that survives shows that the tests do not pin that part of the rule. The eight operators are never (empty the expression), flip (invert the comparison), drop-and, drop-unless, drop-exclusion, loosen and tighten (move a threshold by a factor of ten), and for0 (remove the for-duration, the hold time before firing). A surviving never-mutant is critical, flip high, the three drop operators medium, and the rest low.

Sixth, seven-day headroom: whether each threshold is reachable against recent real values. Seventh, alert history: how many distinct series fired in the history window (14 days plus a 365-day store on the cluster estate, 30 days on the monitoring-host estate), as context rather than a finding. Eighth, routing: each rule's full label set, after external labels and relabelling, is routed through the live Alertmanager configuration and checked for silences and inhibitions.

Before mutation the test suites were green on both estates: 0 failing cases across 90 test files (cluster) and 81 (monitoring-host).

3. The Baseline: 386 Rules With Findings

The audit reached all 496 rules, found no orphans (loaded rules with no repository source) and no drift between repository and live configuration. Table 1 gives the worst severity per rule.

estate	rules	critical	high	medium	low	clean
cluster Prometheus	270	1	7	82	112	68
monitoring-host Prometheus	226	8	6	93	77	42
total	496	9	13	175	189	110

Table 1. Baseline worst severity per rule, 21 October 2026, 04:20 UTC.

There were 356 findings on the cluster estate and 429 on the monitoring-host estate, 785 in all. By check: selector 49, join 54, fixture 202 and mutation 480.

The mutation stage generated 2,191 mutants (1,157 on the cluster estate, 1,034 on the monitoring-host estate). The tests killed 1,713 of them, 78.2%, and 478 survived. Table 2 breaks this down by operator, and combines the two estates.

operator	killed / run, before	killed / run, after
never	493 / 494	496 / 496
flip	605 / 617	617 / 617
tighten	188 / 212	207 / 213
loosen	137 / 212	207 / 213
for0	251 / 468	456 / 470
drop-exclusion	21 / 55	33 / 56
drop-unless	18 / 71	19 / 69
drop-and	0 / 62	0 / 62
all operators	1,713 / 2,191 (78.2%)	2,035 / 2,196 (92.7%)

Table 2. Mutants killed out of mutants run, both estates combined, at baseline and after the repairs.

The largest group of survivors is the hold time: in 132 cluster rules and 85 monitoring-host rules the tests still passed with the for-duration removed.

Tests existed for every cluster rule: each of the 270 had a fire case and a quiet case (539 and 629 cases in total). On the monitoring-host estate 206 of 226 had both; 20 lacked a quiet case (19 had a fire case only; one had no recorded case).

The fixture check explains the looseness. Of 2,896 test series (1,641 and 1,255), 457 (15.8%) carried at least one label problem, across 175 rules: a hostname or job that does not exist, a label value the exporter never emits, a join label the real series lack.

Headroom (323 and 296 thresholds examined) and routing produced no findings. Alert history shows how much rests on tests alone: at baseline 123 cluster rules and 140 monitoring-host rules had never fired in their history window. That is not a defect, but for them a test is the only evidence of fireability.

4. What the 785 Findings Turned Out to Be

We split the findings into 118 units (64 cluster, 54 monitoring-host), each a rule group or a slice of one. For each unit an agent edited a private copy of the files; independent verifier agents then reran the analyser and promtool, one checking the claimed improvement and one checking that the rule's meaning was unchanged, with up to two repair rounds. Agents recorded a verdict for every group and could not change thresholds or retire rules. Rows group findings and are not equal-weight units.

verdict	cluster	monitoring-host	total
test defect fixed (test or fixture wrong or too weak)	206	183	389
analyser false positive	60	66	126
by design	33	96	129
accepted residual	3	12	15

verdict	cluster	monitoring-host	total
labelled rule defect	8	2	10

Table 3. Verdicts on 669 finding groups across 118 units.

By share, 58% of rows were weak or unrealistic tests, 19% analyser false positives, 19% correct behaviour the analyser cannot see, and 1.5% labelled rule defects. In the verifiers' judgement the tests were the weak part and the rules mostly sound.

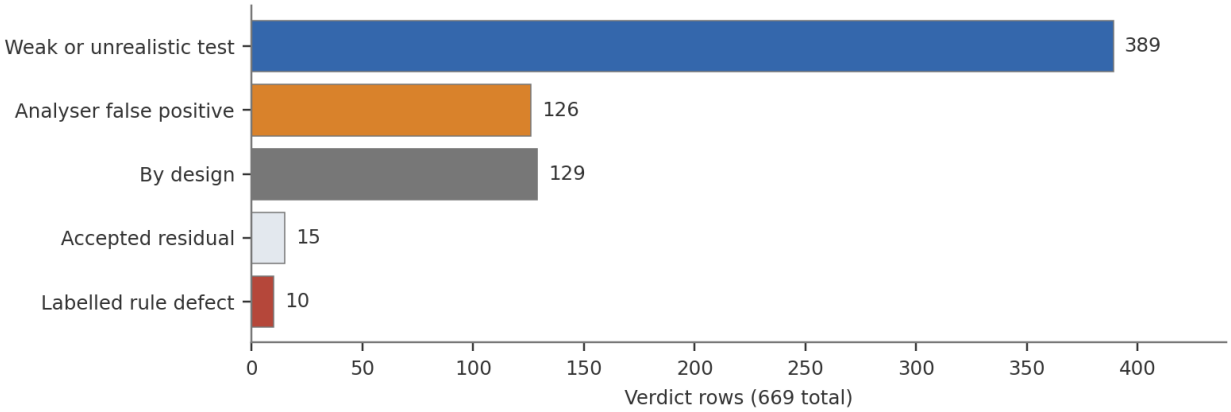


Figure 1. Verdicts on the 669 finding groups. Values are in Table 3.

The ten rows are not ten bugs. Seven rows changed a rule, covering four rules: two expressions and two annotations. The other three (one cluster row, for a hold-duration removal that tests had missed, and the two monitoring-host rows) were added tests that the verifiers labelled rule defects; by our own definition they are test defects. The monitoring-host pull request changed no rule expression.

5. The Defects That Were Real

Both expression changes removed a dead exclusion branch. One rule watches the kubelet proxy path and another the health of the fleet console's own metrics. Each carried a maintenance-hold exclusion that mutes the alert during planned work. In the first rule one of two branches matched on a hostname label that kubelet series never carry, so it could never exclude anything; the instance branch, which for kubelet series is the bare node name, worked. In the second rule it was the reverse: the instance branch compared a scrape address (IP and port) with a bare node name and never matched, while the hostname branch did the work.

Removing a dead branch changed no behaviour; the hold metric was empty, so the branch was a no-op, and no behavioural test could have failed on the old expression. Behavioural tests did not catch it; the join probe and the mutation stage both flagged both rules. We removed the dead branches and added tests that pin the live ones.

The other two defects were annotations. Two rules that check repository truth printed the scrape target in their description instead of the workload name, because the exporter publishes the workload in a renamed label. The expressions were fine and we left them alone.

6. The Analyser's Own Errors

The analyser was wrong often enough to matter: 126 verdict rows were false positives, and two were graded critical.

The two were "and" gates on the monitoring-host estate, in a fan-stopped rule and a spare-capacity rule. The analyser reported that the operands carried different label names and

no explicit matching clause, so the "and" matched nothing: 0 of 2 left series matched 33 right series for the fan rule, and 0 of 6 matched 19 for the spare-capacity rule. The fix workflow tested both against live data and refuted both. The two fan-rule series are voltage-regulator fan sensors reading 0 (seven-day median 0), which the right-hand "greater than zero" filter correctly removes; a control query that relaxed only the left filter matched 27 of 29 (the two exclusions being the idle fans), and the analyser's fully unfiltered check matched 29 of 29. The six spare-capacity series are cloud-VM devices that all read 0 and are excluded by design; unfiltered, the join matched 25 of 25 with identical label sets.

The structural check compared label names of the filtered series sets, and when a filter leaves a few series lacking a label, it concludes the sides are incompatible. We revised the join probe mid-audit to strip filters and find the last moment of co-presence. On the cluster estate one critical and four high join findings fell to a single high (the kubelet rule); the monitoring-host estate rose from 0 to 2 join criticals, these two. Table 1 uses the revised numbers.

A second class concerns the mutation operators. All 62 drop-and mutants (34 and 28) survive before and after, and the verifiers graded all 26 drop-and rows and 25 of 29 drop-unless rows as analyser errors. Root-level drop-unless is also 0 of 49 killed before and after. The mutant was textually the original rule, differing only in parenthesisation or duration printing (seven days printed as one week), and no test can kill a mutant that is the original. This is a blind spot of the operator at the top-level gate, not a score the fixers could raise; they added real gate-removal cases instead.

The third class is the selector window. Nine criticals remain: one cluster, eight monitoring-host. Two are the "and" gates above. The other seven are selectors for series an exporter emits only while a fault exists, such as a state value of "error". The 14- or 30-day window cannot see them, so it reports them dead. The verifiers checked the 365-day store or the exporter's description, and a promtool firing case proves each rule fires on that series shape.

7. After the Repairs

The repairs changed 59 cluster and 53 monitoring-host units; 6 needed no change and 4 needed a repair round. They produced 14,369 patch lines; 108 of the 112 verified patches merged cleanly by script and 4 were not applied automatically (the merge summary does not record why). Merged trees passed promtool: all 130 monitoring-host test files, and on the cluster estate 26 rule files loaded and 270 of 270 alerts were proven both ways.

We deployed the cluster change on 21 October by applying the rule ConfigMaps and signalling Prometheus to reload; the monitoring-host change needed no reload. The drift check records it: the 11:35 UTC re-audit reported two live expressions differing from the repository, and the 13:13 UTC live re-audit none.

estate	rules	critical	high	medium	low	info	clean
cluster Prometheus	270	1	0	64	18	0	187
monitoring-host Prometheus	226	8	0	71	6	18	123
total	496	9	0	135	24	18	310

Table 4. Worst severity per rule after the repairs, live re-audit of 21 October 2026, 13:13 UTC.

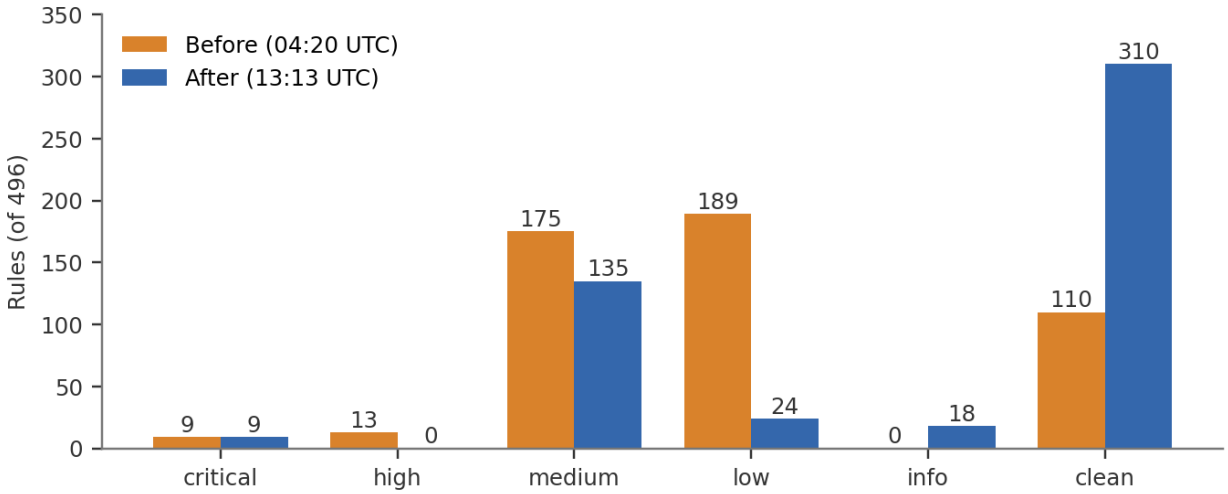


Figure 2. Rules by worst severity, before and after, both estates combined. Values are in Tables 1 and 4.

High-severity rules went from 13 to 0 and clean rules from 110 to 310. Findings fell from 356 to 120 (cluster) and from 429 to 233 (monitoring-host). The kill rate rose from 78.2% to 92.7%, with 161 survivors where there had been 478; the largest movements are in the hold-time operator (251 of 468 to 456 of 470) and the loosened threshold (137 of 212 to 207 of 213).

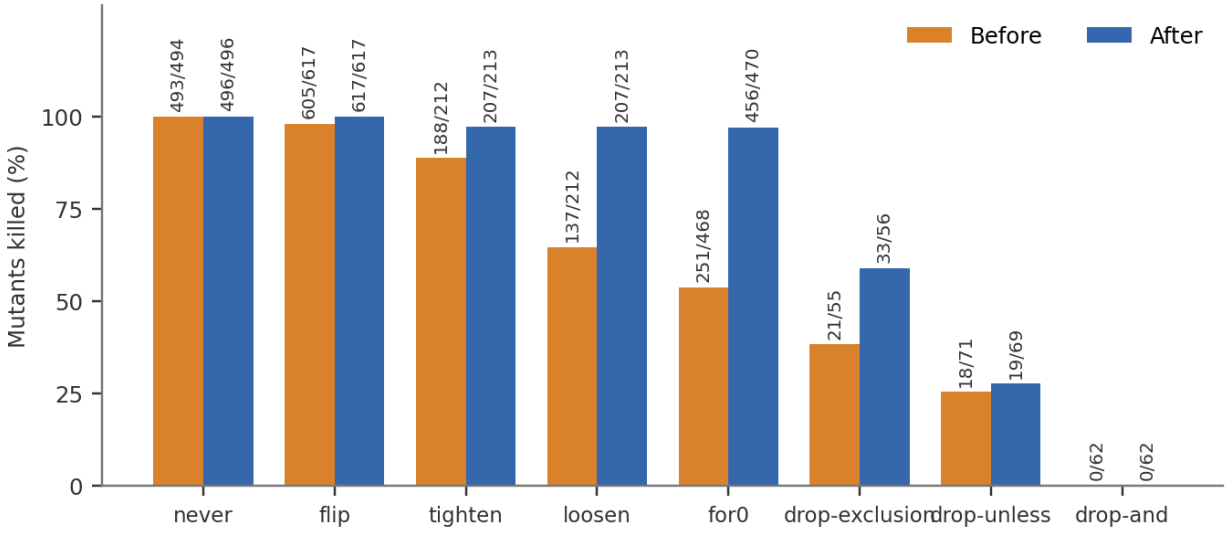


Figure 3. Share of mutants killed by operator, before and after the repairs, both estates combined. Counts are in Table 2.

Fixture realism improved in step: of 3,246 test series, 167 (5.1%) had a label problem, and rules with a fixture problem fell from 175 to 61. Test cases now number 575 fire and 831 quiet (cluster) and 652 fire and 812 quiet (monitoring-host); 225 of 226 monitoring-host rules have both kinds.

"270 of 270 proven to fire and to stay quiet" was already true before the audit; what changed was whether the cases used production label shapes.

Per rule, against the 04:20 baseline, 236 improved, 250 were unchanged and 10 got worse. Eight cluster rules went from clean to low, because a for-removal mutant now survives; they sit in three rule files whose tests the repair rewrote. A separate corpus of mutation-proven firing tests, six files (three covering these eight rules), was on a different checkout (273 rules, 143 test files) used for a residue run, and was not committed. On that branch a rerun at 15:14 UTC (8 rules, 29 baseline cases, no failures) reported all eight clean; on the base without it, eight low findings. None of the six was in the repository on 21 October, so the residue remained open for

all eight. Two monitoring-host rules went from clean to info because a slow case exceeds the 60-second limit and is excluded from mutation, a change in what the tool reports.

8. What These Numbers Will Not Carry

This is one analyser run once on one fleet on one day. Selector and join findings depend on live value filters and 14- or 30-day windows, so another day would flag a somewhat different set. We report counts and proportions without confidence intervals, from one run.

A surviving mutant shows that the tests do not pin part of a rule, not that the rule is wrong. With 126 false positives and 129 by-design rows among the 669, survivors are mostly evidence about test design, and some about the analyser.

The nine remaining criticals are not rule defects according to the verifiers, but the verifiers are the same family of agents that did the repairs. The two refuted "and" gates rest on queries recorded in the verifier log; the figures are transcribed from the verifiers' recorded evidence. The seven fault-only selectors rest on the long-term store and exporter descriptions; we did not trigger the real faults.

Fixture realism measures label shape, not the reality of a fault. We did not prove each rule fires in production; we showed that a test with production-shaped labels makes it fire and fails when the rule is mutated.

The verifiers recorded 36 threshold reviews, all "keep" or "leave as is", with no exporter gaps and no retirement proposals. They flagged two thresholds for operator judgement (one with thin headroom, firing for 7 distinct series in 30 days; one likely too high for current hardware); we changed neither.

What survives is narrower than the headline. The ratio of weak tests to rule-changing rows, 389 to 7, is a measured fact about this fleet on this day. The mechanism behind dead exclusion branches, invisible to behavioural tests and exposed only by mutation, is plausibly general; we saw it twice. The kill-rate gain from 78.2% to 92.7% is real on our estates, but the after-scores were measured by the instrument and mutation operators the repair agents were told to improve against. They show the loop converged on the grader, not that held-out fault detection improved; no seeded-fault or held-out mutant set was run. Of the 161 surviving mutants, 111 are root-level drop-and or drop-unless mutants that cannot be killed by construction, which sets a ceiling above the 62 drop-and alone.

9. What We Changed in Practice

The repaired tests and the two dead-branch removals are live on the cluster estate, and the monitoring-host tests are merged. The next analyser changes: discard textually identical mutants, compare unfiltered operand sets before grading a join critical, and consult the long-term store before calling a fault-only series dead. The five uncommitted corpus files are the first thing to close.

10. Methods and Artefacts

The instrument is a purpose-built PromQL analyser in Python with its own parser and printer, querying the Prometheus HTTP API and routing label sets through the Alertmanager routing logic. Mutation runs use promtool in reduced private workspaces and execute only each rule's own tests. The repair stage used language-model agents with a proof verifier and a semantics verifier per unit. The baseline audit ran at 04:20 UTC on 21 October 2026, the pre-deploy re-audit at 11:35, the post-deploy re-audit at 13:13 and the residue rerun at 15:14. We retain

the machine-readable audits for each state, the verdicts for all 118 units, the per-unit patches, the merge log and the repository history.

PureTensor operates a sovereign AI infrastructure fleet (on-premises GPU compute, storage, and serving) run day-to-day by autonomous agents under human direction. This paper is PT-R-2026-058. The analyser measurements were taken on 21 October 2026; the raw artefacts are retained.