Safety-Kernel Empirical Analysis Report

Acumen FHIR Query Translation Engine — Pre-Execution Guard Rails

Adversarial, Classified Absorption Measurement with k6 Evidence

Document Version: 1.0.0 **Classification:** Public

Perceptohedron Technologies Pvt Ltd
Registered Office: Karnataka, Bangalore

July 15, 2026

Abstract

This document presents a controlled, adversarial characterisation of the Acumen FHIR engine's **Safety Kernel** — the set of pre-execution guard rails that reject pathological or malicious query inputs *before* they reach the database. It is the companion to the *Empirical Performance Analysis Report* (v4.0.0), which measured the happy-path engine and explicitly *deferred* classifying its stress-test rejection population (that report, Phenomenon 8, states the ∼30% rejection rate "must be classified… before it is called either 'validation working' or 'defensive absorption' "). This report supplies that classification, not by decomposing the earlier mixed-workload run, but by driving a purpose-built, fully *labelled* adversarial taxonomy against the live engine and measuring, per attack class: (i) absorption correctness, (ii) the compute-cost asymmetry between rejecting a hostile query and serving a valid one, (iii) which server-side guard fired, and (iv) blast-radius containment under a concurrent attack storm. Every request carries its expected outcome, so the rejection population is classified by construction. All quantitative claims are sourced from the merged reports written by `scripts/runKernelBenchmark.js` (client-side classification reconciled against server-side `safetyRejections` counters) and are reproducible with `npm run benchmark:kernel`. Headline results (single-host, development corpus): **100% clean absorption** of every core adversarial class (0 hard failures, 0 false positives across 72 serial and 14,514 concurrent hostile requests); a stable **rejection cost of ∼3 ms median / ∼5 ms p95**, yielding a **4×–15× cost advantage** over serving a valid query depending on cache state; and correct server-side attribution to the chain-depth and value-count guards (previously un-instrumented; fixed for this analysis). Findings are *classified by evidence strength*: Tier 1 (directly measured)—absorption correctness, rejection cost, guard attribution; Tier 2 (preliminary, single-run, hardware-bound)—attack-storm traffic isolation. Documented **guard gaps** (adversarial inputs the kernel does *not* currently reject—notably an `_include` fan-out that executes for ∼14 s) are reported in full and are *not* counted as absorption.

Contents

1 Executive Summary

The Safety Kernel is not a single module but a set of pre-execution guard rails distributed across the query-translation pipeline. This report characterises them adversarially. Measurements were taken on **July 15, 2026** on the QUASAR development laptop against a live server (git `45fba50`) with the instrumentation fixes described in §9 applied. Two runs are cited throughout: a single-VU *characterisation* run (clean per-class cost) and a concurrent *attack-storm* run (absorption at scale + traffic isolation).

- **100% clean absorption, no hard failures.** Every core adversarial class returned a clean `4xx` + FHIR `OperationOutcome`: 72/72 hostile requests absorbed in the serial run and 14,514/14,514 in the storm run, with **0** soft leaks (attack served), **0** hard leaks (`5xx`/timeout), and **0** false positives (valid query wrongly blocked). Source: `coreCorrectness` in the cited reports.
- **Rejection is cheap and bounded.** Median rejection cost is ∼3 ms (chain-depth 3.10 ms, value-count 3.89 ms, malformed 2.91 ms; p95 ≤5.3 ms) versus a 13.1 ms valid-query baseline — a **3.4**×–**4.5**× advantage on a cold server, rising to **12**×–**15**× against a warmer valid baseline observed in prior runs. The *rejection cost itself* (∼3 ms) is the stable, defensible quantity; the multiple varies with what it is compared against.
- **Four distinct guards, now correctly attributed.** The kernel comprises the chain-depth (limit 11), value-count (limit 50), complexity (score threshold), and pipeline-size (∼17 MB) guards, plus a validation layer. The chain-depth and value-count guards were previously uninstrumented; after the fix (§9) they record `safetyRejections.chainDepth` and `.valueCount`, confirmed by non-zero deltas (`chainDepth +32`, `valueCount +9` serial; `+4881`, `+1187` storm).
- **Instrumentation coverage is partial and quantified.** After the fix, 41.8%–56.9% of client-observed rejections are attributed to a named server-side guard; the remainder are cleanly identified as *validation-layer* rejections (malformed tokens/dates/modifiers, operator-injection), which throw without a dedicated counter. This coverage figure is itself a reported metric, not a hidden gap.
- **Traffic isolation is partial under storm** *[Tier 2, preliminary].* Under 25 concurrent attacker VUs on this single-event-loop server, valid-traffic median latency degraded 10.3 → 128.3 ms (12.41×) while the server stayed live (event-loop delay bounded at 30.6 ms, heap 95%). This is a saturation regime on a 6 GB laptop, reported as a limit to understand, not a feature.
- **Documented guard gaps.** Several adversarial inputs are *not* rejected: an `_include`/`_revinclude` fan-out executes for ∼14 s (a genuine denial-of-service vector with no guard), stacked `_has` reverse-chains execute, and `_count=999999999` is clamped (not rejected). These are reported in §8 and excluded from all absorption figures.
- **Two graceful-degradation fixes.** An over-cap request body now returns a graceful `413` (previously an ungraceful `500`); operator-injection strings are neutralised (treated as literals) or rejected. Both are documented in §9.

2 Test Environment and Methodology

2.1 Hardware and Corpus

Component	Specification
Hostname	QUASAR (ASUS TUF FX505DT)
CPU	AMD Ryzen 7 3750H @ 2.30 GHz (8 cores)
RAM	5.9 GB available
OS	Windows 11 (10.0.26200)
Node.js	v22.19.0
Storage engine	MongoDB replica set (rs0), local
Server build	git `45fba50` + instrumentation fixes (§9)
Corpus	Development FHIR dataset (local `fhir` database)

Table 1: Test environment. Single host, single session — results are hardware-bound and not yet cross-validated on the production VM.

2.2 Harness Architecture

Measurement is driven by a standalone harness, `scripts/runKernelBenchmark.js`, structurally analogous to the happy-path `scripts/runBenchmark.js` but purpose-built for the guard rails. It: (1) boots the server with query-metrics collection enabled (or attaches to a running one); (2) *owns* a clean metrics session so the server-side `safetyRejections` counters read as exact per-run deltas; (3) runs the k6 adversarial script `k6/safety-kernel-benchmark.js`; and (4) reconciles the client-observed rejection population against the server-side guard counters, writing a single merged report to `benchmark-results/safety-kernel/`. It is invoked with `npm run benchmark:kernel` (variants: `:storm`, `:quick`, `:attach`).

2.3 Attack Taxonomy

Every request is **labelled** by attack class and expected outcome; the rejection population is therefore classified by construction. Core (pass/fail-gating) classes exercise inputs the guards are *designed* to reject:

Class	Adversarial input	Guard exercised
`chain_depth`	self-referential reference chain, depth 12–25	chain-depth limit (11)
`value_count`	`_id` OR-list of 700 values	value-count limit (50)
`malformed`	operator-injection, empty/double-pipe tokens, bad date prefixes, bad modifiers, broken JSON	validation layer

Table 2: Core adversarial classes and the guard each is designed to trip.

A separate set of *gap probes* (single-shot, non-gating) exercises candidate attacks that characterisation showed the engine does not cleanly reject (§8). A valid control group of fast, indexed queries establishes the "cost of serving legitimate traffic" baseline and detects false positives.

2.4 Outcome Classification

For an *attack* the desired outcome is absorption; for a *valid* query it is a clean pass:

- **Absorbed** — clean `4xx` + `OperationOutcome` (guard worked, cheaply).
- **Leaked (soft)** — `2xx`: the attack was served (false negative).
- **Leaked (hard)** — `5xx`/timeout/socket drop: the guard failed catastrophically (the outcome the kernel exists to prevent).

- **False positive** — a valid query wrongly rejected (`4xx`).

3 The Guard Inventory

The Safety Kernel comprises four numeric guards plus a validation layer, verified against `src/`. The thresholds are empirically derived and OOM-calibrated; the memory constraint for chains and value-lists is documented in code as (Depth × Values) ≤ 550.

Guard	Threshold (source)	Location	Records counter?
Complexity	score (env `QUERY_COMPLEXITY_*`)	`searchHandler.js`	yes (`complexity`)
Chain depth	11 (OOM at 13)	`queryBuilder.js`	**yes (fixed, §9)**
Value count	50 (OOM at 51)	`queryBuilder.js`	**yes (fixed, §9)**
Pipeline size	∼17 MB / depth 100	`searchHandler.js`	yes (`pipelineSize`)
Validation	token/date/modifier/injection	translation layer	no (throws `FHIRSearchError`)

Table 3: Guard inventory. The chain-depth and value-count guards fire first for externally-reachable inputs, so the complexity and pipeline-size guards act as deeper backstops and are exercised only as gap probes here.

A material finding of the characterisation phase is that large `_id` OR-lists trip the **value-count** guard (limit 50), *not* the pipeline-size guard — the query is rejected before any aggregation pipeline is built. The k6 class is named accordingly.

4 Result 1 — Absorption Correctness [Tier 1]

Every core adversarial request was cleanly absorbed, in both the serial and the concurrent regime.

Metric	Serial (6 iter)	Storm (25 VU)
Hostile requests	72	14,514
Absorbed (clean 4xx + OO)	72	14,514
Absorption rate	100%	100%
Leaked — served (false neg.)	0	0
Leaked — hard fail (5xx/timeout)	0	0
False positives (valid blocked)	0	0
Valid control queries served	30	1,195

Table 4: Absorption correctness. Source: `coreCorrectness` in `kernel-benchmark-2026-07-15T15-09-45.json` (serial) and `-15-18-02.json` (storm).

The zero hard-failure count is the load-bearing result: the guards reject with a clean, FHIR-conformant `OperationOutcome` rather than crashing the request path — precisely the failure mode (unbounded query cost → heap exhaustion) they exist to prevent.

5 Result 2 — Cost Asymmetry [Tier 1]

The defensible claim of the Safety Kernel is that rejecting a hostile query costs far less than the compute it prevents. The serial characterisation isolates this (the storm regime does not; see the caveat below).

Class	Median (ms)	p95 (ms)	× vs. valid serving
Valid baseline (control)	13.10	—	1.00
`chain_depth` reject	3.10	4.28	4.22
`value_count` reject	3.89	5.33	3.37
`malformed` reject	2.91	5.04	4.51

Table 5: Cost asymmetry, serial run. Ratio = valid median / rejection median. Source: `costAsymmetry` in `kernel-benchmark-2026-07-15T15-09-45.json`.

Interpretation. The *rejection cost* is stable and cheap: ∼3 ms median, ≤5.3 ms p95, across all three guards. The *multiple* (3.4×–4.5× here) is a function of the valid-query baseline, which varies with cache warmth and corpus: against the warmer 13–40 ms baselines observed in earlier runs the same ∼3 ms rejection yields a 12×–15× advantage. We therefore report the rejection cost as the primary quantity and the multiple as a derived, workload-dependent figure.

Caveat (storm ratios are not cost asymmetry). In the storm run the reported per-class rejection latencies rise to ∼40 ms because every request — hostile and valid alike — queues on the single event loop; the storm "ratio" compares a quiet valid baseline against a saturated rejection latency and is *not* a clean cost-asymmetry measurement. Storm data is used only for absorption-at-scale (§5) and traffic isolation (§7).

6 Result 3 — Server-Side Attribution and Reconciliation [Tier 1]

Because every request is labelled client-side and the harness diffs the server-side guard counters over a clean session, the report reconciles the two: how many client-observed rejections each named guard accounts for, and how many occur at the un-instrumented validation layer.

Attribution	Serial	Storm
Client-observed rejections	72	14,514
→ chain-depth guard	32	4,881
→ value-count guard	9	1,187
→ complexity guard	0	0
→ pipeline-size guard	0	0
Server-attributed total	41	6,068
Validation-layer (unattributed)	31	8,446
Counter coverage	**56.9%**	**41.8%**

Table 6: Reconciliation. Source: `reconciliation` and `serverSide.safetyRejectionsDelta` in the cited reports.

The raw server-side delta from the storm run:

Listing 1: serverSide.safetyRejectionsDelta (kernel-benchmark-2026-07-15T15-18-02.json)

```
"safetyRejectionsDelta": {
  "complexity": 0, "chainDepth": 4881, "pipelineSize": 0, "valueCount": 1187, "total": 6068
}
```

Interpretation. The non-zero `chainDepth` and `valueCount` deltas confirm the instrumentation fix (§9); prior to it these guards fired but recorded nothing, so server-side coverage of this adversarial mix was effectively **0%**. The residual gap (43%–57% of rejections at the validation layer) is an *honestly reported* limitation: malformed-token, bad-date, bad-modifier, and operator-injection rejections throw a `FHIRSearchError` without a dedicated safety counter. Client-side `4xx` classification remains the authoritative rejection total; the coverage figure quantifies how much of it the server-side telemetry independently corroborates. This directly addresses the concern in the

Empirical Performance Analysis Report (Phenomenon 8) that server-side rejection counts were not trustworthy enough to monetise.

7 Result 4 — Traffic Isolation Under Attack Storm [Tier 2, preliminary]

The embedded-WAF value proposition is that hostile traffic does not starve legitimate traffic. The storm run interleaves a valid-traffic scenario with 25 attacker VUs and compares valid latency under fire against the quiet baseline.

Metric	Value
Valid median, quiet	10.34 ms
Valid median, under attack	128.27 ms
Degradation ratio	12.41×
Server event-loop delay (max)	30.65 ms
Server heap (max)	95%

Table 7: Traffic isolation. Source: `trafficIsolation` in `kernel-benchmark-2026-07-15T15-18-02.json`.

Honest reading. Isolation is *partial*. Valid traffic remained fully served (0 false positives, 0 valid errors) and the server stayed live with a bounded event-loop delay, but valid latency degraded ∼12× under a concentrated storm. This is a **saturation regime** on a single-event-loop Node process on a 6 GB laptop with 25 attacker VUs, not a demonstration of cost-free isolation. It is a single run on a single host and is classified Tier 2 (preliminary, hardware-bound); a controlled sweep across attacker concurrency and a run on the production VM are required before any isolation guarantee is asserted. Consistent with the *Empirical Performance Analysis Report*'s treatment of the analogous saturation curve, this is reported as a limit to understand, not a feature to market.

8 Result 5 — Guard Gaps (Un-Absorbed Adversarial Inputs) [Tier 1]

Not every adversarial input is rejected. The gap probes document the boundary of what the kernel absorbs. These are *excluded* from all absorption figures above.

Probe	Outcome	Latency
`_include`/`_revinclude` fan-out	**pass-through (served)**	∼14 s (cold); 168 ms (warm)
stacked `_has` reverse-chains	pass-through (served)	280 ms
`_count=999999999`	pass-through (clamped, ∼1 MB payload)	36 ms
`_count=-500` (negative)	pass-through (ignored)	6 ms
operator-injection {`"$where"...`}	neutralised as literal (served empty)	73 ms
token OR-list (400 values)	rejected `400` (value-count)	4 ms
over-cap request body	rejected `413` (graceful; was `500`)	9 ms

Table 8: Gap probes. Source: per-probe console verdicts, run 2026-07-15. `gaps` summary: 5 pass-through, 2 rejected, 0 server-error.

The material gap is the `_include`/`_revinclude` fan-out: a syntactically legal query with no cost guard that executes for ∼14 s against the corpus — a genuine denial-of-service vector the Safety Kernel does *not* currently cover. Operator-injection is defended (either rejected with `400`, or neutralised by treating the injected operator as a literal string that matches nothing), but this defence is *literalisation*, not rejection, so it appears as a served-but-empty response rather than a `4xx`. These gaps must be closed (or explicitly scoped out) before the absorption effect is represented as a complete defensive perimeter.

9 Instrumentation Changes Made for This Analysis

Three targeted changes were made so the guards could be measured with resolution. They are behaviour-preserving for the guards' *decisions*; they change only telemetry and one error-mapping.

1. **Chain-depth counter.** The chain-depth guard (`queryBuilder.js`, limit 11) threw a `FHIRSearchError` but never called `recordSafetyRejection('chainDepth')`; the counter was permanently zero. The call was added; the guard now records (`+32` serial, `+4881` storm).
2. **Value-count counter.** A new `valueCount` category was added to `QueryMetricsCollector` and recorded at the value-count guard (limit 50). This is the guard large OR-lists actually trip (`+9` serial, `+1187` storm).
3. **Graceful body-cap rejection.** An over-cap request body reached the body-parser as a `PayloadTooLargeError` with no handler and fell through to a `500`. A case was added to `UnifiedErrorHandler` mapping it to a graceful `413` + `OperationOutcome` (verified: the `oversized_body` probe now returns `413`, 0 server-errors).

No guard threshold was altered; the deeper complexity and pipeline-size guards were left as-is. The validation-layer rejections remain uncounted by design (scattered throw sites); closing that residual coverage gap is deferred as it requires a single reconciliation choke point and its own testing.

10 Relationship to the Financial Appraisal

The *Strategic Appraisal* (v8.0.0) lists "Safety-Kernel hostile-traffic absorption" as an *un-monetised* value driver, retained qualitatively and explicitly excluded from the headline valuation pending classification (that document, §"Excluded Upside"; and the caveat "classify the rejection mix before monetizing"). This report is that classification. It establishes, on a controlled adversarial mix: (i) that the absorption is real and clean (100%, 0 hard failures); (ii) that its unit economics are favourable and stable (∼3 ms per rejection); and (iii) precisely where the perimeter is incomplete (§8). It does *not* by itself justify a dollar figure — the isolation result is preliminary (Tier 2), the corpus is a development dataset, and the perimeter has documented gaps — but it converts the driver from "asserted" to "measured, with known limits," which is the precondition the appraisal set for monetising it (e.g. as WAF-substitution value on the database tier). Any monetisation should cite the rejection-cost figure and the coverage/gap caveats, not the absorption *count* alone.

11 Evidence-Tier Summary

Finding	Evidence	Tier
100% clean absorption, 0 hard failures	72 + 14,514 labelled requests	**1 (Verified)**
Rejection cost ∼3 ms / p95 ≤5.3 ms	serial characterisation	**1 (Verified)**
Guard attribution (chain-depth, value-count)	non-zero server counter deltas	**1 (Verified)**
Counter coverage 41.8%–56.9%	client-vs-server reconciliation	**1 (Verified)**
Guard gaps (include fan-out, etc.)	per-probe verdicts	**1 (Verified)**
Cost *multiple* (4×–15×)	baseline-dependent	2 (workload-dependent)
Attack-storm traffic isolation (12.4× deg.)	single run, single host	2 (Preliminary)

Table 9: Findings classified by evidence strength, mirroring the tiering discipline of the Empirical Performance Analysis Report.

12 Conclusion

On a controlled, fully-labelled adversarial workload, the Acumen Safety Kernel absorbs 100% of the inputs its four numeric guards and validation layer are designed to reject, cleanly (`4xx` + `OperationOutcome`, zero hard failures, zero false positives), and cheaply (∼3 ms per rejection). The

instrumentation fixes made for this analysis give the chain-depth and value-count guards their first server-side visibility, and the client-vs-server reconciliation quantifies — rather than hides — the 43%–57% of rejections that still occur at the un-instrumented validation layer. Two honest limits bound the result: traffic isolation under a concentrated storm is only partial and is measured on a single 6 GB host (Tier 2), and several adversarial inputs — most importantly an `_include` fan-out — pass through with no guard. The measurement supplies the classification the *Empirical Performance Analysis Report* deferred and the *Strategic Appraisal* required, and it is fully reproducible via `npm run benchmark:kernel`.

Reddy and Reddy Family Office (i.f.)
Principal's Work Product — Confidential

Appendix A — Reproduction

Listing 2: Reproduction commands and artefacts

```
// Characterisation (serial, clean per-class cost):
npm run benchmark:kernel:attach -- --base-url http://localhost:<port>

// Attack-storm (absorption at scale + traffic isolation):
npm run benchmark:kernel:storm -- --base-url http://localhost:<port>

// Spawn a fresh server automatically (slow first boot: SmartSeeder + indexing):
npm run benchmark:kernel

// Artefacts (merged report; raw k6 stream is opt-in via --raw):
benchmark-results/safety-kernel/kernel-benchmark-<timestamp>.json
benchmark-results/safety-kernel/kernel-benchmark-latest.json
```

Cited runs: `kernel-benchmark-2026-07-15T15-09-45.json` (serial characterisation) and `kernel-benchmark` (attack storm). Attack definitions and outcome classification: `k6/safety-kernel-benchmark.js`. Harness and reconciliation: `scripts/runKernelBenchmark.js`.