CarbonMantis

TECHNICAL WHITEPAPER

Proof, Not Noise

Safe-by-default, evidence-driven penetration testing for web applications and APIs

v1.0 · August 2026

carbonmantis.com

Contents

01 Executive summary

02 The problem: security testing teams actually run

03 Confidence-rated findings

- Confidence is earned by evidence
- Every finding is reproducible — and minimized
- Corroboration raises confidence; it does not multiply noise
- High-signal CI gates

04 Platform security

- Multi-tenant isolation at every layer
- Credential protection: envelope encryption
- Isolated execution plane
- Control-plane SSRF defense
- Evidence minimization and redaction
- Authentication, authorization, and audit

05 Verifiable, tamper-evident evidence

- What every scan seals
- How verification works — independently, offline
- Why it matters
- Cryptographic details

06 Architecture and scale

- Contract-first, queue-driven, observable

07 Detection freshness and reproducibility

08 Developer workflow and standards

09 Scope and limitations

10 Conclusion

01 Executive summary

CarbonMantis is an automated penetration testing platform for web applications and APIs. It runs industry-standard offensive tooling against targets you own, then delivers **confidence-rated findings** — each with reproducible evidence, a confidence rating, and clear remediation — instead of a pile of low-signal alerts.

This whitepaper is written for the people who decide whether a security tool earns a place in their stack: security-owning engineers and DevSecOps practitioners. That audience asks two hard questions, and CarbonMantis is engineered to answer both:

1. **Can I trust the output?** The number-one reason teams abandon automated security tools is false positives. CarbonMantis treats confidence as a first-class, evidence-derived property: a finding is only *Confirmed* when exploitation was safely demonstrated with verifiable proof, and every finding is reproducible by a developer in seconds.
2. **Can I trust the vendor?** A penetration-testing platform is a high-value target and is trusted with sensitive data. CarbonMantis is built to the standard it tests others against: strict multi-tenant isolation, per-tenant envelope encryption, an isolated and network-locked execution plane, a control plane hardened against server-side request forgery, and evidence that is minimized and redacted before it is ever stored.

At a glance

- **Signal, not noise** — a three-tier confidence model (Confirmed / Probable / Possible) where the level is earned by evidence, and every finding ships a reproducible, redacted proof.
- **Safe against production** — mandatory ownership verification, non-destructive default payloads, per-target rate limiting, automatic abort on target degradation, and opt-in gating for aggressive modules.
- **Secure by construction** — per-tenant envelope encryption, row-level tenant isolation, an isolated execution plane, and an SSRF-hardened control plane.
- **Privacy by design** — raw request/response bodies are never persisted; evidence is synthesized and redacted server-side before storage.
- **Built to scale** — contract-first (OpenAPI) services, queue-driven idempotent workflows, ephemeral autoscaling workers, and full-pipeline distributed tracing.
- **Developer-native** — a first-class CLI, version-controlled configuration, SARIF output, and deterministic CI gates on severity and confidence.
- **Honest** — CarbonMantis does not claim to replace expert-led testing for business-logic abuse, chained exploits, novel techniques, or social engineering.

02 The problem: security testing teams actually run

Web application and API security testing has historically been **expensive and episodic**. A manual penetration test is thorough but arrives a few times a year — far slower than modern release cadence, and out of reach as a per-commit gate for most teams. So the gap between "we shipped a change" and "someone checked it for exploitable bugs" stays wide.

Automated dynamic scanners promised to close that gap, but most created a new problem: **noise**. A scanner that reports 200 issues where 150 are false positives doesn't save time; it creates a triage tax, erodes trust, and gets muted. The typical scanner makes this worse by emitting a binary verdict ("vulnerable / not vulnerable") with no signal about how *sure* it is.

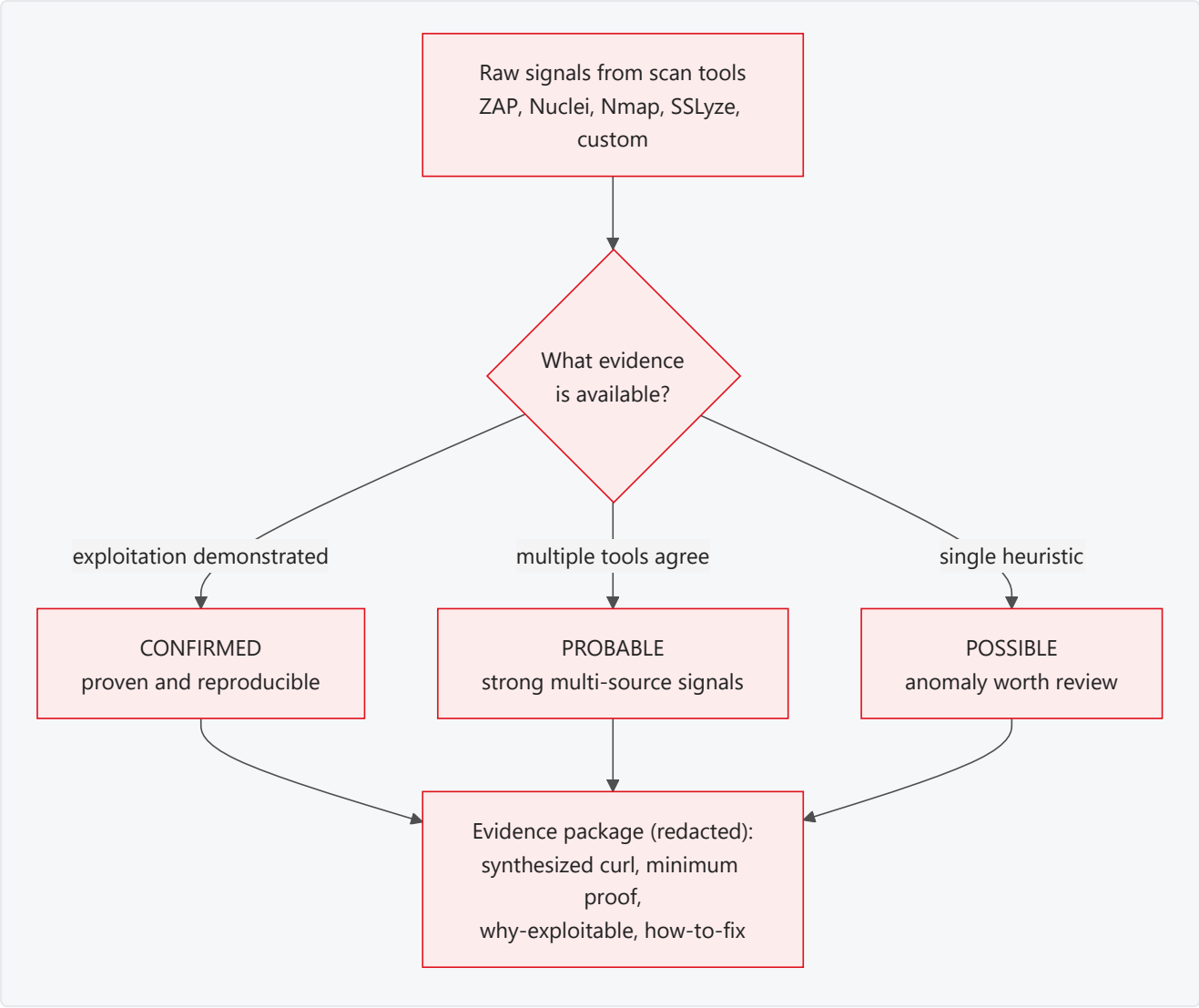
CarbonMantis exists to deliver the practical value of continuous security testing **without** the noise tax — high-signal, evidence-backed coverage of the vulnerability classes automation can handle reliably, integrated directly into developer workflows, and honest about where its boundary lies.

03 Confidence-rated findings

Confidence is earned by evidence

Every CarbonMantis finding is assigned one of three confidence levels. The level is determined by **the evidence available**, not by the reporting tool's default severity.

Level	Criteria	Example
Confirmed	Exploitation safely demonstrated with verifiable proof.	Reflected XSS marker appears in the response; blind SQLi time delay matches the injected delay against a control baseline; an SSRF payload triggers a request to the CarbonMantis callback server.
Probable	Strong indicators from multiple independent sources, but safe exploitation was not fully demonstrated.	Two or more tools independently flag the same endpoint and parameter; error-based signals strongly suggest injection without timing or callback confirmation.
Possible	A single tool heuristic or a behavioral anomaly worth investigation.	One tool reports a potential issue from response patterns; anomalous behavior warranting a human look.



A finding is elevated to **Confirmed** only when at least one form of verifiable proof is obtained: payload reflection (the injected marker appears in the response), timing verification (the response delay matches an injected delay against a control baseline), an out-of-band callback (the target reaches the CarbonMantis callback server carrying the finding's correlation token), behavioral proof (the response demonstrates unauthorized access, such as an IDOR returning another user's record), or a deterministic configuration observation (a missing header, TLS 1.0 enabled, directory listing active).

Because proof is obtained **safely**, some genuinely real issues land as *Probable* rather than *Confirmed* when demonstrating exploitation would risk the target. CarbonMantis states this plainly rather than inflating confidence.

Every finding is reproducible — and minimized

Confidence ratings are only useful if a developer can verify them, and evidence is exactly where secrets and PII most easily leak. CarbonMantis resolves both concerns at once: every finding carries an evidence package built for immediate reproduction, and that package is **synthesized and redacted server-side before anything is persisted**.

1. **A synthesized reproduction** `curl` — rebuilt from structured fields (method, URL, redacted parameters), never passed through from a tool, with sensitive values redacted.

2. **The minimum proof for the finding's proof type** — the reflected marker and the parameter that carried it, the measured baseline-vs-probe timing deltas, or the specific misconfigured header. **Raw request/response bodies are never persisted.**
3. **"Why we believe this is exploitable"** — the proof chain in plain language.
4. **"How to validate the fix."**

> **What we deliberately don't store.** Raw request/response bodies are the largest uncontrolled place secrets and PII hide, so CarbonMantis structurally never keeps them. Evidence is redacted in layers — always-on credential scrubbing, built-in PII defaults, and your own org-specific filters — applied to every persisted target-derived field. A minimum proof artifact per proof type survives redaction, so a *Confirmed* finding stays Confirmed. (See §4, Platform security.)

Example: a Confirmed blind SQL injection

> **Finding:** Blind SQL injection in the `search` parameter of `GET /api/users` **Severity:** Critical · **Confidence:** Confirmed (timing) · **CWE-89** · **OWASP A03:2021 — Injection**

```
# Control request (no payload) — baseline latency
curl -s -G 'https://api.example.com/api/users' \
  --data-urlencode 'search=alice' \
  -o /dev/null -w 'control:  %{time_total}s\n'
# control:  0.08s

# Test request (5-second injected delay)
curl -s -G 'https://api.example.com/api/users' \
  --data-urlencode "search=alice' AND SLEEP(5)-- -" \
  -o /dev/null -w 'payload:  %{time_total}s\n'
# payload:  5.09s
```

The response time tracks the injected `SLEEP(5)` delay (5.09s versus a 0.08s baseline) and repeats deterministically across control/test pairs. Notice the proof needs **no response body** — the timing deltas alone justify the confidence tier, which is exactly why dropping raw bodies costs nothing here.

Corroboration raises confidence; it does not multiply noise

CarbonMantis runs multiple tools whose coverage overlaps. Rather than emitting the same issue several times, findings are deduplicated by a stable fingerprint — `SHA-256(finding_type + normalized_endpoint + parameter + payload_class)`, where `normalized_endpoint` collapses path parameters (`/api/users/123` becomes `/api/users/{id}`). When multiple independent tools hit the same fingerprint, that agreement **raises confidence** instead of producing duplicates: one finding per real issue, with corroboration expressed as a higher tier.

High-signal CI gates

Because confidence is structured data, it can drive deterministic CI policy. Combine a severity threshold with a confidence floor so pipelines fail only on issues you would actually block a release for:

```
# Fail the build only on Confirmed criticals — near-zero false-positive gate
carbon scan gate --fail-on critical --min-confidence confirmed
```

`Possible` findings can be excluded from gates by default and surfaced only in review, so the noisy tail never blocks a deploy while remaining available when someone goes looking. Each finding is mapped to recognized standards — OWASP Top 10 (2021), OWASP API Top 10 (2023), OWASP WSTG v4.2, a CWE identifier, and a CVSS v3.1 score and vector — which makes output portable and audit-defensible.

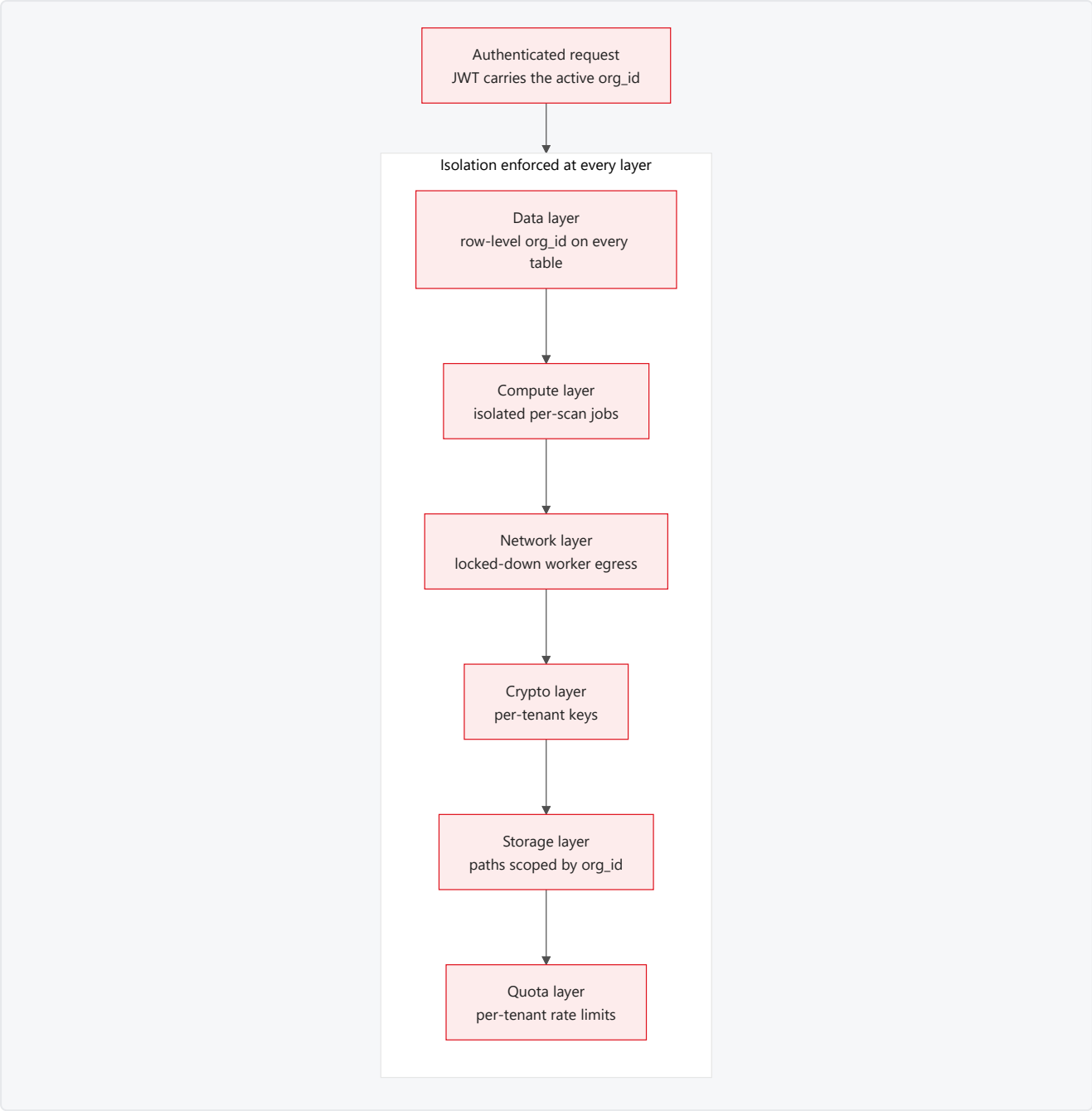
04 Platform security

Adopting a security scanner inverts the usual trust question. You are not only asking “does it work?” — you are asking: *if I can’t see that this vendor builds securely, why would I trust it to test my systems, hand it my credentials for authenticated scans, and let it fire attack traffic at my production environment?* CarbonMantis answers by construction.

Multi-tenant isolation at every layer

Isolation is enforced independently at several layers, so no single mistake collapses the boundary between tenants.

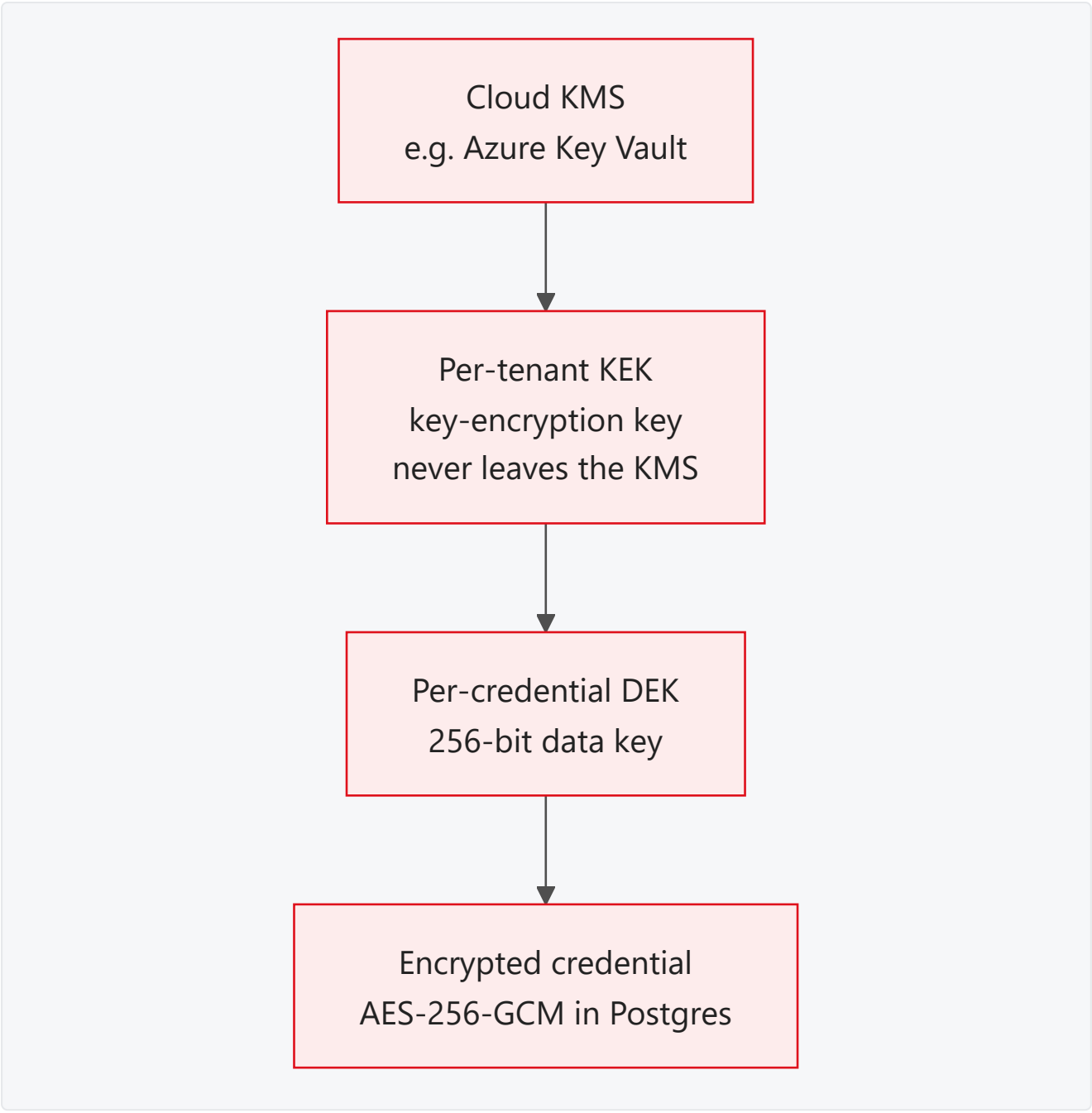
Layer	Mechanism	Guarantee
Data	Row-level `org_id` on every table, enforced in the repository layer; there is no global-query method.	Cross-tenant reads are architecturally impossible.
Compute	Ephemeral, isolated per-scan jobs with no shared state.	One tenant’s scan cannot observe or disrupt another’s.
Network	Restricted worker egress plus NAT; workers cannot reach internal systems.	A compromised tool cannot pivot into the platform.
Cryptography	Per-tenant key-encryption keys.	One tenant’s key never decrypts another’s data.
Storage	Object paths scoped by `org_id` prefix.	Artifacts and reports are segregated per tenant.



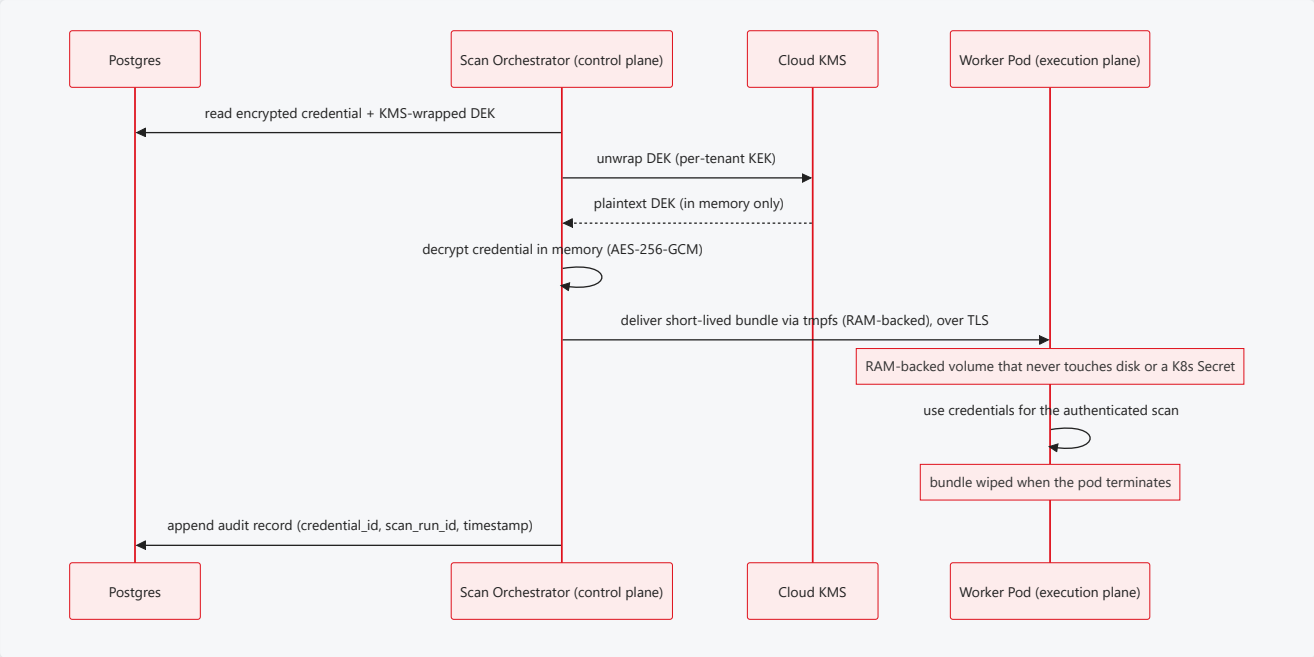
The active organization is derived exclusively from the JWT `org_id` claim — never from a URL, query string, or request body — so a client cannot request another tenant's data by editing a path. Cross-tenant isolation is exercised by integration tests that create data in one organization and assert that queries from another return zero results.

Credential protection: envelope encryption

For authenticated scanning, CarbonMantis stores customer-provided credentials — the most sensitive data in the system — protected with **envelope encryption keyed per tenant**.



Each credential is encrypted with its own 256-bit data-encryption key (DEK) using AES-256-GCM; the DEK is wrapped by a per-tenant key-encryption key (KEK) held in a cloud KMS and is never persisted in plaintext. Compromising one tenant's KEK exposes only that tenant — the single shared application key that most systems use is explicitly rejected. Credentials are decrypted only in the control plane, only in memory, then delivered to workers over a RAM-backed volume that is destroyed with the pod.



Workers have no network path to the KMS, credential bundles are never stored as Kubernetes Secrets, credentials are never returned in the API or UI (only masked indicators), every decryption is audited, and deletion is a cryptographic wipe.

Isolated execution plane

CarbonMantis workers actively send attack traffic to customer targets, which makes them the most dangerous component to run — so they are separated from everything that holds data or secrets. The control plane (API, identity, orchestration, aggregation, reporting, and all database and secret access) runs apart from the **execution plane**, where scan tools run as ephemeral, isolated jobs on a dedicated node pool. Worker network policy denies all egress by default, then permits only outbound traffic to public target web ports, artifact upload, and the callback endpoint; egress to RFC 1918 private ranges and cloud metadata endpoints is blocked. Tool containers run non-root, with read-only root filesystems where possible, no privileged mode, and enforced resource and time limits. Worker traffic exits through a NAT gateway with a static, publishable IP range so you can allowlist CarbonMantis in your WAF or firewall.

Control-plane SSRF defense

CarbonMantis makes outbound requests to user-supplied URLs during ownership verification. Left naive, that is a classic server-side request forgery (SSRF) hole. Because we are a pentest product, we close it by design: a custom dialer validates the **actual resolved IP at connect time** — on the initial request and every redirect hop — which defeats DNS rebinding. Only publicly-routable, global-unicast destinations are permitted; every special-use range is denied (loopback, link-local including cloud metadata, RFC 1918, IPv6 unique-local, carrier-grade NAT, multicast, and documentation ranges). The policy **fails closed**: an unset policy denies, an unknown policy name is a startup error, the permissive development policy is rejected in production, and the transport ignores environment proxies so they cannot route around the IP check. The same class of bug we report to customers, we engineer out of our own control plane.

Evidence minimization and redaction

CarbonMantis minimizes what it ever stores and redacts the rest server-side, before anything is persisted.

- **Raw request/response bodies are never persisted**, and the reproduction `curl` is synthesized from structured fields rather than a tool string passed through — so there is no tool-supplied body to leak.
- **Layered, additive redaction:** always-on credential scrubbing of known sensitive keys (which customers cannot disable); built-in PII defaults for common patternable data (Social Security numbers, Luhn-validated credit-card numbers, email addresses); and customer-configurable per-org filters (additional key-names and value regexes for app-specific PII). Custom filters are additive only, org-scoped, and evaluated with Go's linear-time RE2 engine, so a pattern cannot cause a ReDoS.
- **Coverage and enforcement:** redaction applies to every persisted target-derived field — evidence, endpoint, parameter, and proof description — and runs in the aggregator before persistence, never client-side. Filter changes are audited and apply to scans started after the change; existing data can be cleaned with an explicit retro-redact or rescan job.

Authentication, authorization, and audit

Passwords are hashed with Argon2id. Tokens are RS256 JWTs with short-lived access tokens (15 minutes) and rotating refresh tokens (7 days); TOTP MFA is supported. CI authenticates with org-scoped, rotatable API keys shown only once at creation. Role-based access control (Owner / Admin / Scanner / Viewer) is enforced throughout — notably, a Scanner can *use* stored credentials to run an authenticated scan but can never *view* them, and that use is audited. An append-only audit log records significant actions with actor, resource, IP, and timestamp.

05 Verifiable, tamper-evident evidence

A pentest report is only as valuable as it is trustworthy. If the record of what was scanned could be quietly altered — by an attacker who breached the platform, *or by the vendor* — it cannot serve as evidence. CarbonMantis is built so that you never have to take our word for it: every scan is sealed into a cryptographically signed, append-only record you can verify yourself, offline, against a key we cannot secretly swap. **Don't trust us — verify.**

What every scan seals

Each scan emits two independent, signed, append-only streams: **execution-fact attestations** — a monotonic record of what actually executed (the tools that ran, the targets, the scan units, and timestamps), the "proof the work you paid for happened" layer — and a **security audit trail**, a tamper-evident record of security-sensitive operations (credential handling, session brokering, lease redemption). Both are anchored to **WORM (write-once, read-many) immutable storage**. We anchor compact cryptographic checkpoints — signed digests, not raw scan data — so the evidence is small and privacy-preserving.

How verification works — independently, offline

You do not have to trust CarbonMantis's servers, database, or staff to trust the evidence:

1. **Every checkpoint is digitally signed (ECDSA P-256).** The signature covers a canonical, domain-separated encoding binding the organization, a sequence number, the log root hash, the previous checkpoint's digest, the signing-key identity, the environment, and the issue time — a single changed bit breaks verification.
2. **The verification key is pinned out-of-band.** The public key is distributed through a channel independent of the API that serves your data, and the verifier trusts a static pinned keyset with **no trust-on-first-use** — a key it was not told to trust is rejected. A compromised API therefore cannot hand you a forged record together with a matching forged key.
3. **An open verifier checks the record.** A standalone tool (shipping with conformance fixtures) confirms the signature (authentic and unaltered), the key's validity window and revocation status (retired or compromised keys are rejected), and the hash chain plus monotonic sequence (nothing was inserted, removed, reordered, or rolled back).
4. **The independent WORM anchor proves non-equivocation.** Because the anchor sits on immutable storage with its own independent key and hash chain, the vendor cannot have shown a different history to a different party, nor truncated or rolled the log back server-side.

Net: signature (integrity + authenticity) + chain and monotonicity (completeness + ordering) + key validity (freshness) + WORM anchor (non-equivocation) — all verifiable independently, offline, against a key we cannot swap. The record of exactly what ran is sealed the moment it is created and remains verifiable forever after; **even we cannot rewrite or backdate it.**

Why it matters

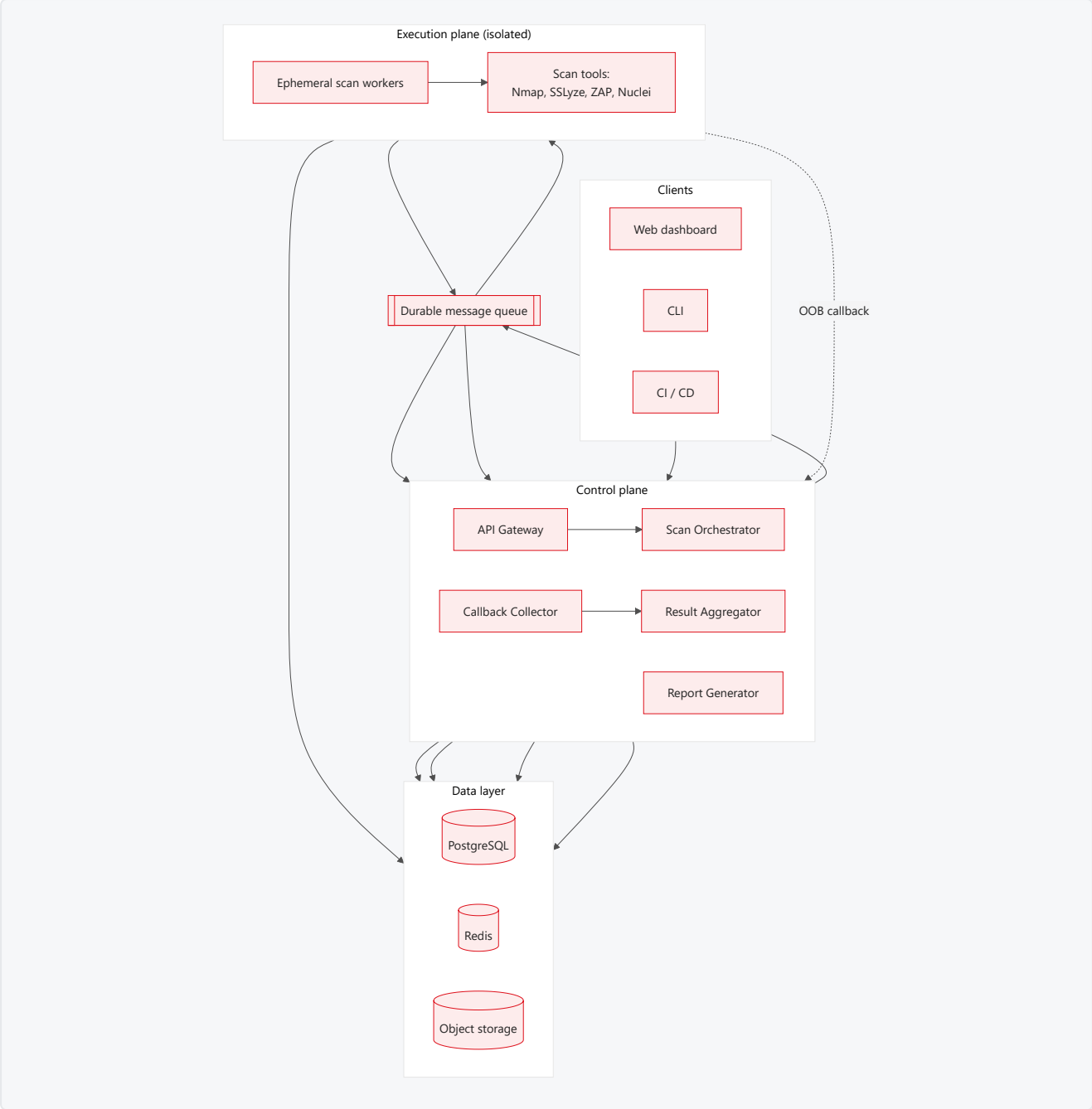
Most vendors ask you to trust their word and their dashboard. CarbonMantis gives you evidence you can verify yourself — a report you can hand to an auditor who verifies it offline, security questionnaires answered with proof rather than prose, and a breach-resilient audit trail an attacker cannot silently rewrite. Immutable, timestamped, signed retention is exactly the shape of evidence that compliance frameworks asking for tamper-evident record-keeping expect.

Cryptographic details

- **Signature:** ECDSA over NIST P-256; the signed message is SHA-256 of a canonical, domain-separated field encoding (org, sequence, root, previous-checkpoint digest, key id, environment, RFC3339Nano issue time).
- **Trust root:** P-256 public keys distributed out-of-band as SPKI PEM; a static pinned keyset (no trust-on-first-use), with validity windows and an externally-anchored revocation instant — a signature is accepted only if its key was valid at the checkpoint's issue time.
- **Integrity and ordering:** an append-only log with a monotonic sequence and a previous-digest hash chain, so truncation, rollback, reordering, and insertion are all detectable.
- **WORM anchor:** an independent, monotonic, keyed-MAC'd, hash-chained checkpoint stream on Azure immutable-blob storage (locked time-based retention / legal hold), with stream partitioning so the execution-fact and audit streams cannot suppress one another, and an independent key so a party holding the database or audit key still cannot forge anchor records.
- **Verifier:** a standalone binary with conformance fixtures that a customer or auditor runs against signed evidence and a pinned trust root, with no live dependency on CarbonMantis infrastructure.

06 Architecture and scale

CarbonMantis is split into a control plane, an isolated execution plane, and a data layer, connected by a durable message queue.

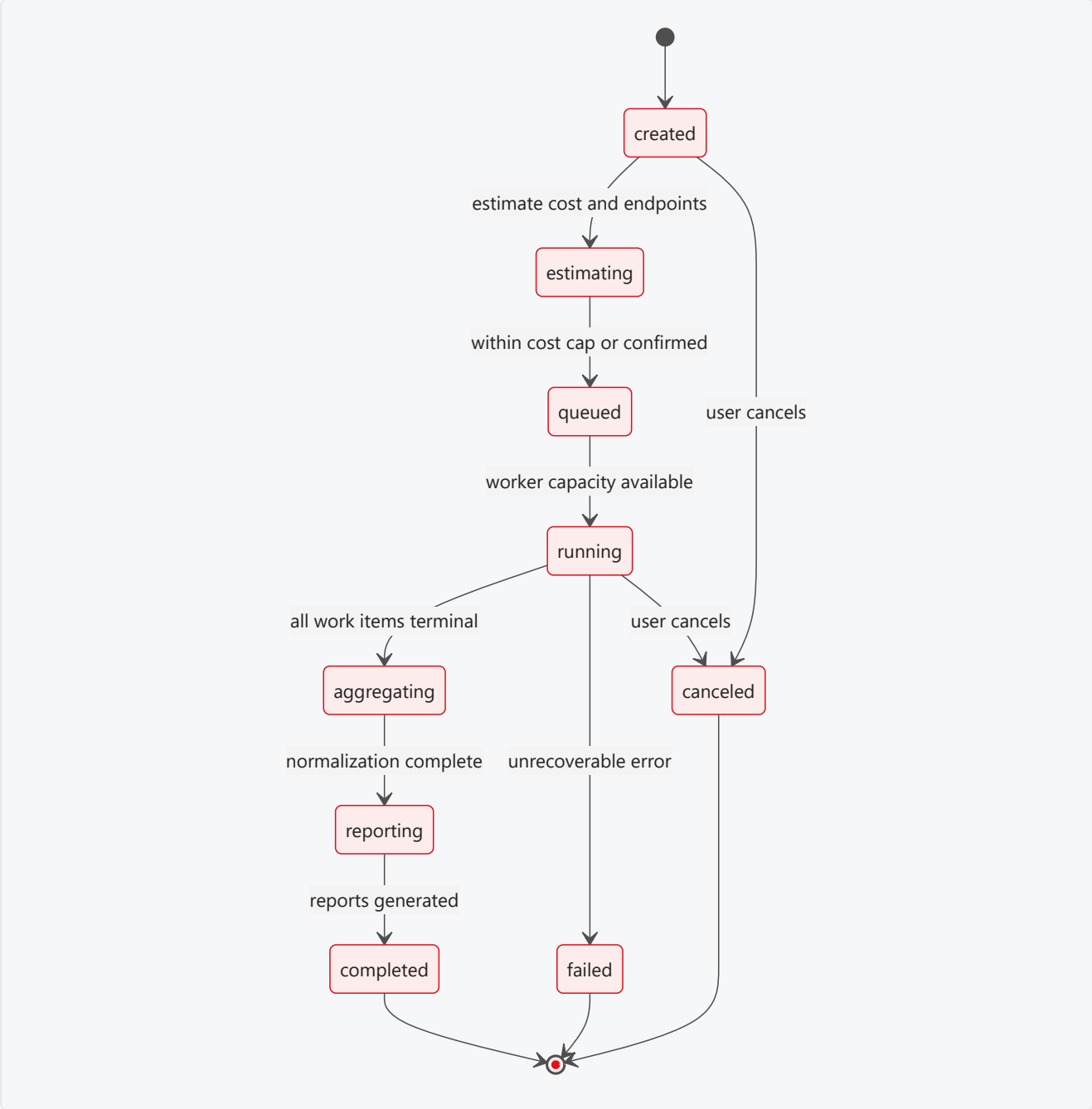


Contract-first, queue-driven, observable

The API is defined once in an **OpenAPI specification**; Go server interfaces and TypeScript clients are generated from it, and CI fails if generated code drifts — the "frontend and backend disagree about the response shape" bug is structurally impossible. All errors use RFC 9457 Problem Details, so every client handles failures consistently.

Scan execution, aggregation, and reporting are decoupled through a durable message broker with **at-least-once delivery plus idempotency keys** (effectively-once processing). Failed work items retry on escalating backoff (15s, 60s, 300s, 900s) and then move to a dead-letter queue rather than vanishing; non-retryable failures short-circuit to `failed`; priority and per-queue concurrency prevent any tenant from monopolizing capacity.

Every scan follows an explicit state machine with database-guarded transitions, which is what lets the orchestrator run as multiple stateless replicas without racing.

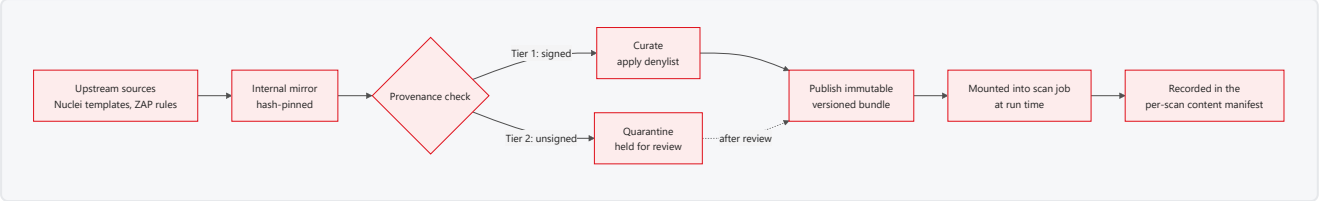


Control-plane services are **stateless and scale horizontally**; scan workers are **ephemeral and autoscale on queue depth**, with no standing fleet to over-provision. Per-tenant concurrency limits, per-target single-scan locks, and per-plan budgets enforce fair multi-tenancy. PostgreSQL is the system of record (ACID transactions for the state machine, rich querying for finding diffs and trends, JSONB for flexible evidence), with object storage for large artifacts and reports and Redis for caching, pub/sub, and locks.

Observability is part of the feature, not a follow-up: **OpenTelemetry** distributed traces follow a single scan across the API, orchestrator, queue hops, workers, aggregator, and reporter, correlated with metrics and trace-linked structured logs (which never contain secrets or PII). The application speaks only OTLP and is backend-agnostic. CarbonMantis runs on Kubernetes with the control and execution planes on separate node pools and managed cloud data services.

07 Detection freshness and reproducibility

A scanner is only as good as its **detection content**. New-CVE rules appear upstream within hours, so CarbonMantis treats detection content as a first-class, versioned artifact with a refresh pipeline — not something baked once and left to rot. Crucially, **freshness never costs isolation**: scan workers never fetch content from the internet. Content is prepared in the control plane and delivered to scans as an immutable, pinned bundle.

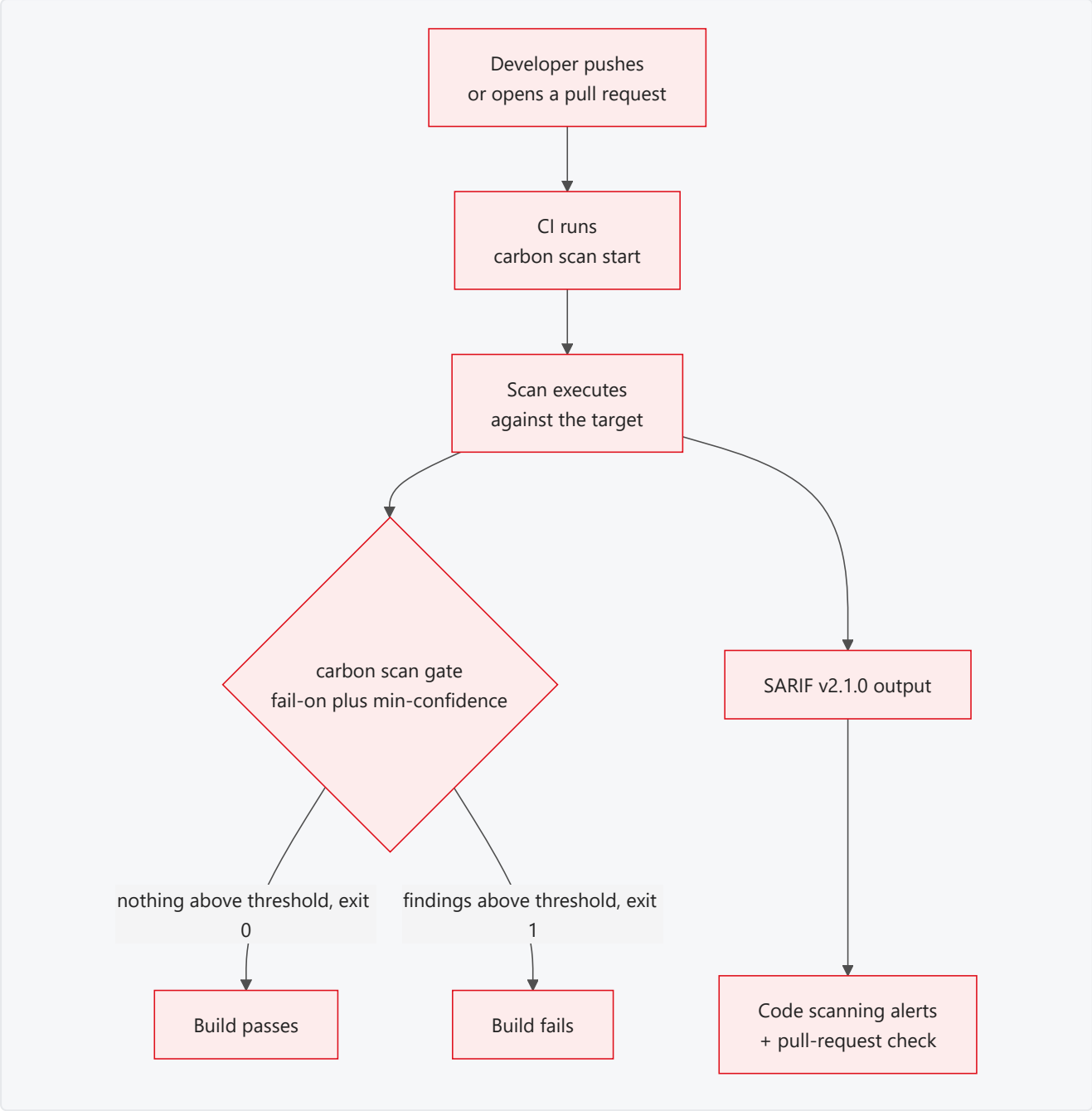


Delivery is tuned per tool: Nuclei templates are published as an immutable, versioned bundle mounted read-only into hermetic scan jobs (supporting hours-level freshness with zero worker egress), while ZAP rules are baked into the tool image and refreshed by scheduled rebuilds with internet access disabled. Supply-chain trust is gated: signed Tier-1 content auto-promotes minus a maintained denylist of noisy or intrusive rules, unsigned Tier-2 content is quarantined for review, and unverifiable content is rejected (fail-closed). Freshness is a monitored SLO; on breach the pipeline holds last-known-good content and alerts rather than serving stale or unverified rules.

Because a finding depends on both the tool binary and the detection content, every scan records an immutable **content manifest** — per-tool content version and integrity hash, the curation-policy and pipeline versions, and the tool binary image digest. You can reproduce and explain any finding months later, and rollback is simply re-pinning a previous known-good version, independently per tool.

08 Developer workflow and standards

CarbonMantis is built for pipelines, not just dashboards. A first-class Go CLI covers auth, the scan lifecycle, findings, and reports, with machine-readable output for automation. Scan configuration lives in your repository as a version-controlled `carbonmantis.yaml`, so scans are reviewed like any other code. The `gate` command returns deterministic exit codes (0 pass, 1 findings above threshold, 2 scan error, 3 configuration error), and gates combine a severity threshold with a confidence floor.



Scans emit **SARIF v2.1.0**, so findings can be uploaded to code-scanning surfaces such as GitHub Code Scanning, where they appear as tracked security alerts (located by endpoint, since CarbonMantis tests a running app) with a check on every pull request; JSON and JUnit XML output are also available. Progress streams over Server-Sent Events, which reconnect automatically so long scans stay observable in CI logs.

Findings map to OWASP Top 10 (2021), OWASP API Top 10 (2023), OWASP WSTG v4.2, CWE, and CVSS v3.1, following PTES and NIST SP 800-115 methodology.

09 Scope and limitations

The fastest way to lose a security engineer's trust is to overclaim, so CarbonMantis publishes its boundary. It automates the vulnerability classes that are reliably machine-detectable with proof:

Area	Examples	Typical proof
Injection	SQLi, XSS, command injection, SSRF	Reflection, timing correlation, out-of-band callback
Transport / TLS	Weak protocols, cipher and certificate issues	Deterministic configuration observation
Security headers	Missing or misconfigured CSP, HSTS	Deterministic configuration observation
Information disclosure	Sensitive files, directory listing, verbose errors	Direct retrieval / observed response
Vulnerable components & exposure	Known-vulnerable services, exposed interfaces	Version/behavioral signals, template matches
Access control (authenticated)	IDOR, forced browsing, privilege escalation	Behavioral proof (access to another user's record, redacted)
API security (authenticated)	BOLA/BFLA, mass assignment, JWT issues	Behavioral proof against defined roles

CarbonMantis intentionally does **not** attempt business-logic abuse, multi-step chained exploits requiring domain interpretation, novel or zero-day techniques without safe deterministic detection, or social engineering and physical attacks. For those, it is a **complement** to expert-led manual penetration testing, not a substitute: automation handles breadth and cadence, and humans handle depth and novelty. Where safe demonstration is not possible, CarbonMantis reports a real issue as *Probable* or *Possible* and explains why, rather than inflating confidence.

10 Conclusion

CarbonMantis is designed around a simple thesis: automated security testing is only worth adopting if its output is trustworthy and its platform is exemplary. It earns trust in the findings through a confidence model backed by reproducible, redacted evidence, and it earns trust in the vendor by building to the standard it sells — tenant isolation, per-tenant envelope encryption, an isolated execution plane, an SSRF-hardened control plane, and evidence that is minimized before it is stored. It scales through a contract-first, queue-driven, observable architecture; it stays current through a controlled detection-refresh pipeline; and it fits real workflows through a CLI, SARIF, and deterministic CI gates. Most importantly, it is honest about where automation ends and expert humans begin.

> **Learn more at carbonmantis.com.**