Sly Technologies
Whitepaper

Enforcement at the Source

The Vantage security architecture — why the client cannot bypass what the daemon never sends

Zero-Trust Enforcement • **0–20 Clearance Scale**
Per-User Session Projections • **Merkle-Chained Audit**

Mark Bednarczyk
CEO, Sly Technologies Inc.
April 2026

Executive Summary

The Vantage security model is built on one principle: enforcement happens inside Vantage's daemon processes, never in the client. A client that is compromised, misconfigured, or operated by an unauthorized user cannot access data above its clearance level — not because the software refuses to display it, but because the data was never transmitted in a readable form. Bypassing enforcement is cryptographically impossible.

Three cooperating daemons implement this model. **Sentinel** captures traffic from the wire, runs the Protocol Stack, writes the capture file with QAT and events embedded, and performs encryption at capture time through the vantage-sdk and the deployment's key vault. **Lynx** is the analysis daemon: when a user (human analyst, SOC operator, integration client) connects, Lynx runs the nine-stage enforcement pipeline against captures on disk and delivers clearance-scoped output through its React, terminal, console, or pipe frontends. **Quarry** is the virtual filesystem and control-plane daemon: it owns the `.tokens` projection, the `/dev/vantage/{capture-name-or-id}/analysis` device, and the Vantage Ledger — the cryptographically chained audit record shared across all three daemons. All three coordinate through a service-and-session discovery protocol.

A unified 0–20 clearance scale drives encryption strength, PII obfuscation, redaction policy, and access control in a single dimension. Each user session is a distinct security context, and the same underlying capture produces different output for different sessions — a level-15 SOC analyst attached through Lynx and a level-7 ML pipeline attached through Quarry read different projections of the same data, produced independently by different daemons, without either daemon's output being visible to the other session.

This whitepaper covers the architecture that makes this work: the three-daemon model, the enforcement pipeline, the per-user session projections, encryption and key management, the policy model that defines PII stances and retention rules through `.manifest` references, the Vantage Ledger, and the split-clearance pattern that lets AI and ML pipelines consume network analytics without sensitive data leaving any daemon in readable form.

Level 0 — the default for single-user standalone deployments, covering roughly ninety percent of users — applies no enforcement overhead beyond standard file permissions. The security model scales in complexity only as the deployment's clearance requirements scale.

1. The Principle: Enforcement at the Data Source

Most security models in network analytics enforce access at the client. The daemon sends the full dataset, and the client decides what the user is allowed to see. Role-based access control, UI-level field masking, query-result filtering — all of these are variations on the same pattern. The client is trusted to redact.

This model has one fatal property: the data the user is not allowed to see is nevertheless transmitted to them, in cleartext, over the wire. If the client is compromised, misconfigured, patched incorrectly, or simply running a debug build with the filters disabled, the enforcement collapses. The user sees everything, because the daemon sent everything.

For casual deployments this is acceptable. For deployments carrying PII, subscriber identifiers, authentication tokens, or regulated content — which describes most enterprise and all carrier deployments — it is not.

Vantage inverts the model. Enforcement happens inside Vantage's daemon processes, before a single byte is transmitted to any client. The daemon serving the client resolves its session clearance against the sensitivity of every field in every token, every packet, and every indexed structure, and delivers only what the clearance permits. Content above the clearance is not redacted on the way out; it is never decrypted in the first place. The encryption keys required to read it are not derivable from the session's key material.

The Lynx frontend — and every other consumer, including SIEM integrations, ML pipelines, and custom SDK consumers — is responsible only for rendering or processing what arrives. A compromised client shows its user exactly the same data as an uncompromised client: whatever the daemon decided the session's clearance permitted. There is no client-side code path that could, if subverted, reveal more.

> *This is the architectural claim every other decision in this document is constructed to support.*

2. Three Daemons, One Clearance Model

Vantage is not a monolith. The platform is three specialized daemons that cooperate through a discovery protocol, each owning a distinct part of the data lifecycle and each enforcing the same clearance model in its own domain. Understanding which daemon does what is essential for understanding where security boundaries sit.

2.1 Sentinel — Capture and Protection

Sentinel owns the wire. It receives packets from capture interfaces, runs the Protocol Stack, decodes traffic, and writes the capture file with the Query Analytics Tree and the events stream embedded in PCAPNG custom blocks. Sentinel also performs encryption at capture time, using the vantage-sdk to derive Data Encryption Keys from the deployment's vault according to the configured clearance level.

Sentinel's external session output is deliberately narrow. It publishes capture statistics, session metadata, and service-discovery beacons advertising the captures it is producing. It does not serve packet content or analysis output to users directly. Sentinel's job is to produce and protect the capture file; consumption happens elsewhere.

This separation matters for the security story. Sentinel holds the most sensitive key material in the system — the master DEKs and vault credentials — and its attack surface is minimized by keeping user sessions out of its process space. A Lynx compromise or a Quarry compromise does not expose Sentinel's key hierarchy.

2.2 Lynx — Analysis Sessions

Lynx is the analysis daemon. It is where user sessions live. When an analyst connects — via Lynx's React web frontend, terminal UI, console, or pipe interface — the connection attaches to a Lynx daemon running somewhere on the deployment. Lynx then uses the discovery protocol to locate the Sentinel that owns the relevant capture and attaches to it for data.

The nine-stage enforcement pipeline described in Section 4 runs inside Lynx, not inside Sentinel. When an analyst queries for packets, Lynx validates the session, resolves the clearance, derives the DEKs permitted by that clearance, decrypts only the fields the session can access, applies obfuscation and redaction, transforms the output for the session's level-of-detail target, renders it in the session's locale, and emits audit records to the Vantage Ledger.

Multiple Lynx users can share a session on a live capture by connecting to a common Lynx daemon — typically the inviter's. That Lynx uses discovery to locate the Sentinel producing the capture, opens it, and serves the shared view to every attached user. Each user's clearance rides with their session. Lynx enforces against the session's clearance, not the daemon's or the inviter's.

2.3 Quarry — Virtual Filesystem and Control Plane

Quarry owns the filesystem surfaces and the control plane. The virtual `.tokens` projection over a capture's events stream is a Quarry surface. The `/dev/vantage/{capture-name-or-id}/analysis` device handle is a Quarry surface. The typed SDK channels that OEM integrations attach to route through Quarry.

When a filesystem consumer — an ML pipeline tailing .tokens, a Python script reading the device handle, an SDK integration pulling tokens programmatically — attaches, Quarry runs its own enforcement pipeline. The same nine stages, the same clearance scale, the same DEK derivation, applied to the filesystem surface instead of to an interactive analyst session. The projection the filesystem consumer reads is not the stored stream; it is the stored stream passed through Quarry's enforcement context for that consumer's clearance.

Quarry also hosts the **Vantage Ledger** — the cryptographically chained audit record that all three daemons write to through channels provided by the vantage-sdk. The Ledger is Quarry's control-plane responsibility; Sentinel and Lynx emit audit entries to it rather than maintaining independent ledgers. Section 8 covers the Ledger in detail.

2.4 The Discovery Protocol

The three daemons locate each other through a service-and-session discovery protocol. Sentinel advertises the captures it is producing. Lynx advertises its active user sessions and the captures those sessions are viewing. Quarry advertises its filesystem surfaces and its Ledger endpoint. Every daemon sees every other daemon's published state.

This enables several patterns that would otherwise require manual configuration. A Lynx user can browse a console or dashboard of live captures across the deployment and attach to any of them, without knowing in advance which Sentinel is producing which capture. QAT level-of-detail synchronization across daemons happens through direct daemon-to-daemon coordination, not through an intermediate file. Shared sessions find each other through the protocol rather than through static URLs.

Clearance enforcement is invariant across this coordination. When a Lynx session moves between captures owned by different Sentinels, the user's clearance is unchanged and is enforced equally by whichever Lynx serves the session. When a Quarry filesystem consumer attaches to events from a capture on another host, the clearance scoping is the same. The daemons coordinate freely; the clearance model stays fixed to the user's session.

3. The 0–20 Clearance Scale

All access control, encryption strength, obfuscation policy, and redaction decisions in Vantage are driven by a unified 0–20 security level scale. Higher levels mean stricter controls, stronger cryptography, and narrower access. The scale applies identically across Sentinel, Lynx, and Quarry — a session's clearance determines what Lynx or Quarry serves it, and the key material Sentinel generates at capture time determines the clearance range in which a capture can ever be accessed.

The tiers:

- **Level 0 — Public / Standalone.** No enforcement overhead beyond file permissions. No vault, no encryption, no key management. Default for single-user deployments. Maximum performance. Approximately ninety percent of users operate here.
- **Levels 1–4 — Internal.** Session authentication and TLS transport required. IP address hashing for basic PII protection. Suitable for internal NOC monitoring where the data sensitivity is low but operational hygiene matters.
- **Levels 5–8 — Confidential.** AES-256-GCM encryption applied to captured content by Sentinel. PII tokenization replaces identifier-bearing fields with consistent pseudonymized values. Multi-factor authentication, role-based access control, and audit logging required. This is the default operating range for enterprise SOC deployments.
- **Levels 9–12 — Restricted.** Full AES-256-GCM encryption with HMAC integrity protection. Full PII obfuscation with field redaction for data above the consumer's

clearance. Hardware security module (HSM) integration for key custody. Vantage Ledger cryptographically chained and typically streamed to external retention. Suitable for lawful intercept environments, financial services deployments subject to regulatory data-handling requirements, and healthcare environments handling protected health information.

- **Levels 13–17 — Secret.** Full encryption with further restrictions on key access and transport. On-premises vault required; no external network connectivity for key operations. Suitable for 5G core probes, government-grade deployments, and defense contractor environments.
- **Levels 18–20 — Top Secret / SCI.** Air-gapped operation, one-time-pad augmentation of symmetric encryption, zero-fill of sensitive fields on eviction from memory. HSM required. Suitable for SIGINT and critical national infrastructure environments.

Damage classification mapping to NSA EO 13526 (e.g., SECRET//NOFORN) is a mandatory metadata field at levels 9 and above.

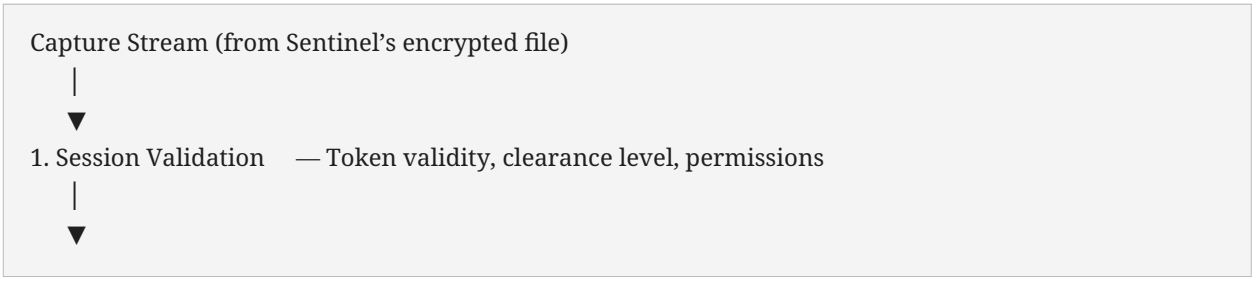
The scale is deliberately broad because the alternative — a separate security product for each tier — creates operational discontinuity that regulated environments cannot absorb. A deployment that begins at level 5 and evolves into level 12 does so through configuration, not through a migration project. The same daemons, the same wire format, the same client surfaces; different enforcement intensity.

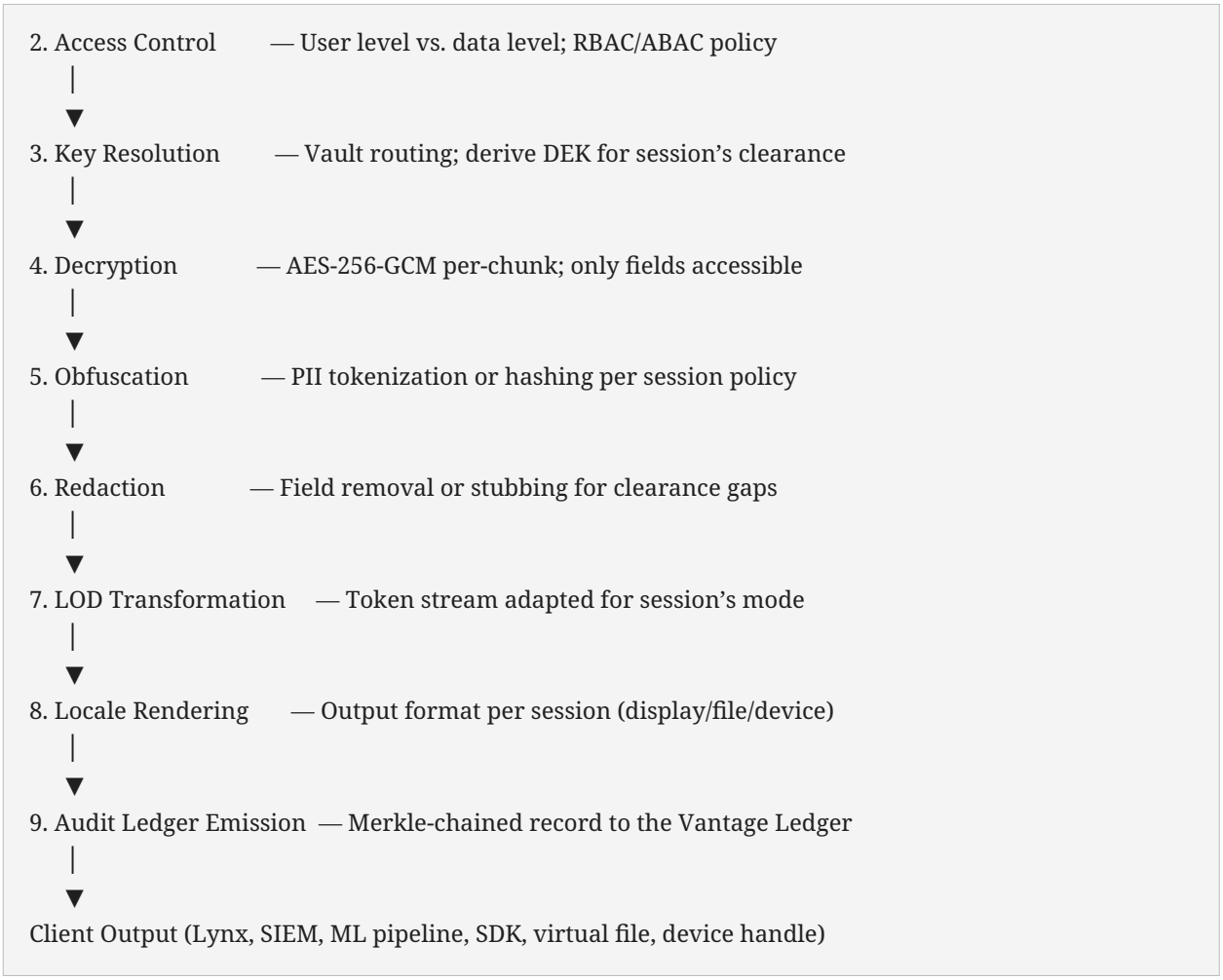
Split override. Encryption strength and obfuscation strength can be independently overridden. A deployment can operate at level 12 for access control while applying level-14-strength encryption (for instance, to meet a specific regulatory requirement that does not match the access-control tier). Overrides can only strengthen, never weaken, any given dimension.

4. The Enforcement Pipeline

All data access in Vantage flows through a nine-stage enforcement pipeline. The pipeline runs inside Lynx for interactive user sessions, and inside Quarry for filesystem-surface consumers. Both daemons implement the same pipeline; the difference is which surface the output emerges on — a rendered display for Lynx, a file or device byte stream for Quarry.

The pipeline is per-user-session: each session has its own context, and the same underlying data produces different output depending on the session’s clearance and obfuscation policy.





Several properties of this pipeline are worth calling out explicitly.

Every stage runs inside the serving daemon. The pipeline does not span processes. Lynx-served sessions run stages 1–9 in the Lynx process; Quarry-served sessions run stages 1–9 in the Quarry process. The client — a human in front of a React frontend, or a Python script reading a virtual file — receives only the output of stage 8, with the audit confirmation from stage 9 written to the Ledger in parallel. Stages 3 through 7 run in the daemon's process space, with DEKs resident only in that process and only for the duration they are actively used.

Session validation is first. Before the pipeline evaluates data sensitivity, it validates the session itself — token validity, clearance level, permissions, expiration. A stale or compromised session fails at stage 1 and no data is resolved for it. Expired sessions are not refreshed automatically; they require re-authentication through the vault.

Key resolution is bounded by clearance. At stage 3, the daemon derives the DEK for the session's clearance from the key hierarchy Sentinel established at capture time. Keys for clearances above the session's are not derivable from the session's key material. A level-7 session that somehow gains access to level-15 encrypted chunks cannot decrypt them — the DEK required is not in reach.

The client receives rendered output, not protected content. After stage 8, the bytes leaving the daemon for the client are already at the correct LOD, already obfuscated, already redacted, already rendered in the session's locale. The client does not need to know what level-15 content would have looked like, because the client never touches the ciphertext that would reveal it.

The Ledger emission is non-blocking at the session level but binding on the audit trail. Stage 9 emits the access record to the Vantage Ledger through a vantage-sdk channel; the session continues serving the client in parallel. The Merkle chain guarantees that every access is captured in the Ledger before the record of the next access appears, regardless of delivery timing.

5. Per-User Sessions and the Projection Model

The enforcement pipeline is per-user-session, not per-capture or per-deployment. Every user or consumer attaches through their own session, with their own clearance, their own obfuscation policy, their own LOD target, their own locale. Each serving daemon maintains its session contexts independently.

The same underlying capture produces different output for different sessions, simultaneously, without either session's content being visible to the other.

Consider a realistic deployment. Four consumers attach to the same live capture produced by a single Sentinel:

- A **SOC analyst at level 15**, connected through **Lynx**'s React frontend from a SOC workstation.
- An **MDR partner at level 12**, connected through **Lynx** (a different Lynx instance, on a different host) from a partner SOC.
- An **ML pipeline at level 7**, attached through **Quarry**'s virtual `.tokens` file from a lower-trust analytics zone.
- A **compliance feed at level 10**, attached through **Quarry**'s SDK channel for regulatory reporting.

Four sessions, four clearances, one underlying capture file produced by Sentinel. Each session reads a different projection, produced by whichever daemon serves it:

- The SOC analyst at level 15 sees full protocol content, decrypted PII, unredacted flow metadata, full TLS handshake detail.
- The MDR partner at level 12 sees the same token stream but with subscriber-identifying fields tokenized to consistent pseudonyms that preserve joinability without exposing the underlying identifiers.
- The ML pipeline at level 7 sees the token structure with PII fields fully hashed, sensitive HTTP URI paths redacted, and several high-sensitivity token types removed entirely.

- The compliance feed at level 10 sees what its regulatory framework requires — typically flow metadata and summary statistics, with packet-level content redacted.

None of these sessions can see the projections of the other sessions. Each serving daemon maintains the context separation in its own process space, using clearance-scoped key material that prevents cross-session decryption even in cases of partial memory exposure. The ML pipeline's Quarry daemon cannot produce the SOC analyst's projection even if asked; the DEKs required are not derivable from the ML session's key material.

Security-keyed cache. Where Lynx and Quarry cache intermediate results for performance, the cache keys incorporate the full security context:

```
Cache Key = hash(dataBlockId + securityLevel + obfuscationPolicy + redactionPolicy)
```

This ensures that a cached result computed for a level-15 session is never served to a level-7 session. Different sessions see different views of the same underlying data, and the cache respects that boundary. The cost is a slight reduction in cache hit rates across sessions with different clearances; the benefit is that the cache cannot become a side channel that bypasses the enforcement pipeline.

Cross-daemon session continuity. A Lynx user navigating between captures produced by different Sentinels keeps the same session, and therefore the same clearance and the same projection policy, across the transition. Discovery locates the new Sentinel, Lynx attaches to the new capture, and the user's session state is preserved — including the Ledger trail of what they have accessed across the move.

6. Encryption and Key Management

Encryption in Vantage is Sentinel's responsibility at capture time, with key derivation flowing through the vantage-sdk against the deployment's configured vault. Lynx and Quarry derive the DEKs they need for their sessions from the same hierarchy, scoped to each session's clearance.

AES-256-GCM with hardware acceleration. The encryption algorithm is AES-256-GCM, with VAES-512 acceleration on modern Intel and AMD silicon. Sentinel uses the SDK to set up the chunk-level encryption as captures are written; Lynx and Quarry use the SDK to decrypt chunks they are authorized to read.

Chunked encryption for random access. The encryption unit is a 64 KB chunk. Each chunk carries its own initialization vector (derived from a global per-capture seed XOR'd with the chunk offset) and its own 128-bit GCM authentication tag. This enables random access into the capture — a Lynx query that targets a specific packet window decrypts only the chunks covering that window; a Quarry filesystem consumer reading `.tokens` decrypts only the token chunks it is currently serving.

DEK per clearance level, derived once, reached many times. Sentinel's vault establishes the master key hierarchy at capture initialization. A distinct DEK exists for each clearance level, derived via Argon2id for the master key, then HKDF with level-specific salts:

```
DEK[level] = HKDF(Argon2id(masterKey, levelSalt, params),
                  info="vantage-dek-L{level}")
```

Lynx and Quarry, when serving a session at clearance N, derive DEKs for levels 0 through N using their SDK channel to the vault. Keys for clearances above N cannot be derived from the session's material; the HKDF structure is one-way and clearance-scoped. This is what makes the "keys for higher clearances are not in reach" claim from Section 4 concrete rather than rhetorical.

Hybrid encryption modes. At lower clearance levels, Sentinel supports hybrid encryption where packet headers remain in cleartext while payloads are encrypted. This preserves compatibility with deep packet inspection workflows — Suricata, Zeek, and similar analyzers running against Sentinel-produced captures can read protocol headers for rule matching without the key material required for payload access. Useful in lawful intercept deployments that need real-time DPI on live traffic but must protect payload content at higher clearance.

```
Level   Mode     Detail
─────   ──────   ──────────────────────────────────────────────
 0–4    Clear    No encryption
 5–10   Partial  L4+ payload encrypted; headers cleartext
11–20   Full     Entire chunk encrypted except first 128 header bytes
```

Performance. On modern Intel and AMD hardware with VAES-512, AES-256-GCM encryption adds negligible overhead through levels 5–8 — effectively full line rate on Sentinel's capture path, effectively transparent on Lynx and Quarry's decrypt path. FPGA offload (available on Napatech and similar capture cards) pushes this further. Software-only throughput on commodity hardware without acceleration is roughly 2–3 GB/s at level 5–8, scaling down at higher clearance levels as HMAC and HSM operations are added. Level 0 has zero cryptographic overhead, matching the capture pipeline's maximum throughput.

Session keys and credential compromise. Session keys are time-bounded and clearance-scoped. A credential compromise gives an attacker access to exactly what the compromised session's clearance permitted, for exactly the remaining duration of the session's validity window, on exactly the daemon serving the session. Keys above that clearance are not reachable from the compromised session's material. Sentinel's master key hierarchy is not exposed to compromised Lynx or Quarry sessions.

7. Policy and the .manifest File

The rules that determine what gets obfuscated, what gets redacted, what gets retained, and how PII is handled are expressed as **policies** — named, versioned artifacts that exist independently of any specific capture. When Sentinel writes a capture, it records the `policy-name-id` of the active policy in a `.manifest` file alongside the capture. Every session that subsequently reads that capture applies that policy.

This is an important architectural distinction. The policy is not a runtime configuration of the daemons; it is a first-class artifact of the data. A capture carries its policy reference with it. A policy applied to one capture does not bleed into another capture under a different policy. Two captures sitting side by side on disk can enforce completely different PII and retention models because they carry different policy references in their manifests.

7.1 What a policy defines

A single policy bundles the decisions a deployment needs to make about a class of captured data:

- **PII stance** — which field types are classified as PII, what obfuscation strategy applies to each (consistent tokenization, one-way hashing, redaction, zero-fill), and at what clearance level each protection activates.
- **Field vocabularies** — organization-specific identifier formats, custom protocol fields, or regulated-domain field types (healthcare identifiers, financial account numbers, telecommunications subscriber IDs) that the policy declares and maps to obfuscation strategies.
- **Retention rules** — how long the capture is retained, when the QAT and events streams transition to long-term metadata-only retention, and when the raw packet data is released. The *Stop Rolling Over Your Evidence* whitepaper covers the retention architecture in depth; policy is where the parameters are set.
- **Clearance defaults** — the minimum clearance required to access the capture, and the default session clearance for consumers who attach without specifying one.
- **Audit verbosity** — how detailed the Vantage Ledger entries are for accesses to this capture, and which event classes trigger immediate external retention streaming.

A policy is a complete specification of how the deployment wants to treat a kind of capture. The daemons (Sentinel at capture time, Lynx and Quarry at access time) read the policy and apply it; they do not encode policy decisions themselves.

The obfuscation strategies. The four strategies a policy can specify for a field type map directly to different architectural guarantees:

- `tokenize` — the field is replaced with a pseudonymized value that stays consistent across tokens within a session, preserving joinability (an IP appearing in five different tokens

appears with the same pseudonym in all five) without exposing the underlying identifier. The tokenization key is scoped to the session, so the same IP produces different pseudonyms for different sessions.

- `hash` — the field is replaced with a one-way hash. Useful when joinability is not required and the field's presence itself is the relevant signal.
- `redact` — the field is removed or stubbed with a marker. The token structure is preserved but the field's content is not delivered to the client at all.
- `zero_fill` — the field is zeroed in daemon memory when it leaves the working set, rather than relying on OS memory reclamation. Used at the highest clearance levels (18+) to protect against timing-based memory disclosure.

7.2 Policies are authored in Vantage Query

Policies are defined, modified, and inspected through Vantage Query — the same command surface analysts use to search captures. This keeps policy authorship inside the operational workflow rather than forcing it into a separate admin tool.

A realistic policy definition for an enterprise SOC deployment might look like:

```
CREATE POLICY enterprise-soc-default AS
 PII {
  src_ip, dst_ip             OBFUSCATE tokenize  WHEN level < 12,
  http.authorization         REDACT              WHEN level < 15,
  http.cookie                REDACT              WHEN level < 15,
  email                      OBFUSCATE hash      WHEN level < 10,
  subscriber_id              OBFUSCATE tokenize  WHEN level < 12,
  payment.card_number        REDACT              ALWAYS,
  payment.cvv                ZERO_FILL           ALWAYS
 }
 VOCABULARIES {
  INCLUDE "pci-dss-v4",
  INCLUDE "internal-subscriber-schema-v2"
 }
 RETENTION {
  raw_packets    30 DAYS,
  qat_events     INDEFINITE,
  audit_ledger   INDEFINITE EXTERNAL siem.example.com:9000
 }
 CLEARANCE {
  minimum        5,
  default        7
 }
 AUDIT {
```

```
  verbosity     standard,
  stream        EXTERNAL FOR level >= 12
}
```

The policy can then be applied to captures at creation time:

```
APPLY POLICY enterprise-soc-default TO CAPTURE live-interface-eth0
```

Or referenced in archival workflows:

```
ARCHIVE CAPTURE nightly-backup WITH POLICY enterprise-soc-default
```

Modification is a versioned operation — a policy update creates a new policy version that applies to captures created after the change. Existing captures keep the policy version recorded in their .manifest, so a policy change cannot retroactively alter how existing data is treated. This is a compliance requirement in regulated deployments and a correctness requirement everywhere else.

7.3 How policies propagate through the pipeline

When a session attaches to a capture, the serving daemon (Lynx or Quarry) reads the capture's .manifest, resolves the policy-name-id to the policy itself, and applies it through stages 5 and 6 of the enforcement pipeline. The session's clearance determines *which* parts of the policy activate — a level-7 session triggers the WHEN level < 12 obfuscation rules but not the WHEN level < 15 redactions, because those have already been applied at a higher clearance level.

The policy is the same for every session reading the capture. What differs is which parts of it apply, determined by clearance. Four sessions at four different clearances reading the same capture all apply the same policy, and the combination of policy + clearance produces each session's distinct projection.

7.4 What this produces

Policies are inspectable. An auditor examining a capture can read its .manifest, retrieve the referenced policy, and see exactly what PII stance, retention rules, and clearance gates applied to that specific data. There is no implicit configuration, no daemon-level override that would change how the capture was handled, no drift between what the policy says and what the daemons did.

> *Intent, execution, and audit are the same artifact.*

For regulated deployments, this is the answer to the "show me your data handling controls" question that data-governance reviews open with. The controls are not a document describing intent; they are the policy referenced by the capture's .manifest, executed by the daemons, and logged in the Vantage Ledger.

8. The Vantage Ledger

The Vantage Ledger is the cryptographically chained audit record of every data access across the platform. It is a Quarry control-plane service. Sentinel, Lynx, and other Quarry instances emit audit entries to it through channels provided by the vantage-sdk; Quarry maintains the ledger's custody, chaining, and retention.

What gets recorded. Each Ledger entry captures:

- The session identifier and the user it was issued to
- The serving daemon and the capture accessed
- The clearance level active for the access
- The specific data accessed (time range, token domains, packet windows, filesystem surface)
- The `policy-name-id` and version applied, from the capture's `.manifest`
- A hash of the bytes delivered to the client (never a copy of the content itself)
- The timestamp and origination of the access

Merkle-chained integrity. Entries are cryptographically chained in a Merkle-style construction. An entry cannot be modified, deleted, or backdated without invalidating every subsequent entry. Tampering is detectable by any party with access to the Ledger's current tip hash. The chaining is maintained across the combined stream from all three daemons — Sentinel's capture-start records, Lynx's session accesses, and Quarry's filesystem-surface accesses all appear in the same chained sequence.

Local and external retention. The Ledger supports both deployment patterns simultaneously. The vantage-sdk provides a local-variant channel suitable for standalone deployments and smaller environments, where the Ledger lives alongside the capture infrastructure. For larger deployments and regulated environments, the Ledger is streamed in real time to an external retention system — a SIEM, a compliance vault, a dedicated audit store — over the same vantage-sdk channel interface. The choice between local, external, or both is a policy configuration; the daemon code path is the same.

Why it lives in Quarry. Quarry's role as the control plane makes it the natural home for an audit service that all daemons need to use. Sentinel's job is to protect the data on the wire and in the file; adding audit-ledger custody to Sentinel's responsibilities would enlarge its attack surface and couple audit availability to capture availability. Lynx's job is to serve user sessions; a per-Lynx ledger would fragment the audit record across hosts. Quarry already mediates between the other daemons for filesystem surfaces and discovery coordination; taking the Ledger on is architecturally consistent with that role.

Policy-driven configuration. For small deployments, the local Ledger variant is sufficient. For enterprise and regulated environments, Ledger policy typically specifies both local custody (for availability during network interruption) and real-time external streaming (for long-term retention

and compliance). Retention periods, external endpoint specifications, and chain-verification policies are all configured through the Ledger's control-plane interface rather than per-daemon.

What this produces. The architectural point is that Vantage Ledger operation is not something an operator enables; it is a property of the enforcement pipelines running. Every Lynx stage-9 emission flows to the Ledger. Every Quarry filesystem-surface access does the same. Every Sentinel capture initialization and key-hierarchy event gets logged. For regulated environments — lawful intercept operations, financial services, healthcare, government deployments — the Ledger is the compliance proof layer, produced automatically as a consequence of the platform operating.

9. AI and ML Pipeline Integration

The question of how AI and ML pipelines consume network analytics without creating a data-governance disaster is addressed extensively in the companion whitepaper *AI Defense Needs a Feature Stream, Not Another Model*. The security model described in this document is what makes that integration safe.

The key architectural pattern is **split clearance**, with Quarry as the enforcement boundary for filesystem-surface ML consumers.

A SOC analyst operating at level 15 through **Lynx** and an ML inference pipeline operating at level 7 through **Quarry** attach to the same Token Stream simultaneously, from the same underlying capture produced by Sentinel. The SOC analyst sees full protocol content and decrypted PII. The ML pipeline sees the same token structure with PII tokenized (consistent pseudonymized values that preserve joinability across the training set without exposing the underlying identifiers) and sensitive content redacted.

The ML model trains on — and infers against — the projection of the stream its clearance permits, produced by Quarry. The underlying sensitive data never leaves any Vantage daemon in readable form. The model can be deployed in a lower-clearance execution context (a separate VM, a different network zone, a cloud-hosted inference environment) without inheriting the capture environment's clearance requirements.

This is the difference between two AI deployment postures:

Conventional AI-native platform. The enterprise ships captured traffic — including PII, authentication tokens, and sensitive content — into a vendor's environment, where the vendor's model processes it. The enterprise inherits the vendor's data-handling posture, cloud footprint, and sub-processor agreements. Data-governance reviews on this posture fail frequently, and when they fail, they fail on the same question: *where does our data go, and under whose enforcement?*

Vantage split-clearance pattern. The model runs in the customer's own infrastructure, at a clearance level appropriate to its operational context. Quarry projects the Token Stream down to

that clearance before transmission. Sensitive data never leaves the Quarry daemon in readable form. The governance question becomes trivial: the data does not go anywhere; the model consumes a filtered view that cannot expose more than the filter allows.

> *For customers whose AI governance reviews have previously failed on vendor platforms, the split-clearance pattern is often the architectural difference that lets the deployment proceed.*

The Token Stream whitepaper covers the feature-stream architecture and the Quarry consumption surfaces — virtual `.tokens` file, `/dev/vantage/{capture-name-or-id}/analysis` device, typed SDK channels — that ML pipelines attach through. Every one of those surfaces runs through the enforcement pipeline described in Section 4, implemented in Quarry, writing to the Vantage Ledger on every access. The clearance model is not a separate layer sitting on top of analytics; it is the layer through which all analytics are delivered.

10. Future Direction: The Bonafide Framework

Sly Technologies is the author of an open specification called **Bonafide** — a framework for user-sovereign encrypted data vaults, hosted at bonafide.id. The specification describes a privacy-by-architecture model in which personal data belongs to the individual rather than the institution holding it, and institutional access operates under cryptographic leases rather than permanent custody.

The Vantage security architecture described in this whitepaper is built on primitives that are forward-compatible with the Bonafide model. The DEK-per-clearance hierarchy, the per-user session pipelines, the Merkle-chained Vantage Ledger, the clearance-scoped key derivation, and the three-daemon separation of concerns all align with architectural patterns Bonafide specifies for future institutional runtimes.

Vantage today operates under the institution-centric model that current enterprise and regulated deployments require: the institution controls the data, and clearance-scoped access is the enforcement mechanism. Bonafide compliance is an architectural direction for the longer term, contingent on ecosystem adoption that Sly Technologies does not unilaterally control.

For readers interested in the privacy-by-architecture approach to institutional data handling, the full specification is published at bonafide.id.

11. Getting Started

The Vantage security model ships with the Vantage Platform. Level 0 is the default; higher clearances activate through vault configuration and platform policy. The enforcement pipeline is always present in Lynx and Quarry; below level 5, the stages are no-ops on the data path.

For deployments evaluating whether the clearance model fits a specific regulatory or operational context, Sly Technologies can walk through the three-daemon architecture, the enforcement pipeline, the key hierarchy, and the Vantage Ledger in detail against the framework the deployment is subject to. The architecture is designed to accommodate the deployment's existing compliance posture, not to impose a separate one.

Request a consultation at slytechs.com/contact.

Related reading

- *AI Defense Needs a Feature Stream, Not Another Model* — the Token Stream architecture and the Quarry-enforced split-clearance pattern for AI/ML pipeline integration.
- *When the Breach Happens, Is Your Data Already There?* — the forensic readiness workflow and the Vantage Ledger's role in incident response.
- *Why Querying 100 EB Takes Seconds* — the Query Analytics Tree architecture that Sentinel produces and Lynx and Quarry query over.
- *Stop Rolling Over Your Evidence* — the intelligent retention architecture and the long-term custody model for Ledger records.

Mark Bednarczyk is the founder and CEO of Sly Technologies Inc., a network packet capture and analysis company based in the Greater Tampa Bay area of Florida. Sly Technologies has been building packet infrastructure since 2005 and ships the Vantage Platform and the jNetWorks SDK. slytechs.com