Sly Technologies
Whitepaper

AI Defense Needs a Feature Stream,

Not Another Model

Why state management at machine speed starts with the stream, not the model

4-byte Token Header • **Labeled at Emission** • **SILO-Correlated** • **Bring Your Own Model**

Mark Bednarczyk
CEO, Sly Technologies Inc.
April 2026

Executive Summary

An AI-speed attacker does not give a SOC the time that the previous generation of detection tooling was built to require. Most of the interesting conversations in network defense right now arrive at the same reframe: defense has to stop being a time-management problem and start being a state-management problem.

That reframe is correct. What it does not say is that state management at machine speed requires a data layer the industry has almost entirely skipped.

State management is a continuous, structured, labeled observation problem before it is a model problem. Whatever consumes that observation — a rules engine, a human analyst, an ML model, a large language model producing narrative summaries — is only as useful as the stream feeding it. The reframe describes the destination. This whitepaper describes the road.

Sly Technologies ships a two-tier feature stream for this purpose. The **jNetWorks Token Stream** is produced as a byproduct of the Protocol Stack running inside the jNetWorks capture pipeline: analyzers decoding TCP, TLS, HTTP, DNS, and other protocols emit compact, labeled tokens as they observe state transitions. The **Vantage Token Stream** layers on top: Vantage's SILO correlation engine takes the jNetWorks stream, joins it to external sources (IDS systems like Suricata and Zeek, threat intelligence feeds like VirusTotal and MISP, telemetry from NetFlow and IPFIX), and emits enriched tokens that bind external events to the specific packet flows they correspond to. Both streams coexist in the same `.events` file and the same Query Analytics Tree; consumers see them as a single unified emission.

The whitepaper focuses on the Vantage Token Stream as the operator-facing product, with Section 4 covering the jNetWorks source layer that underlies it. Both are shipping now — the Vantage Platform as a full operator deployment, the jNetWorks SDK v3 as a bundled kit for platform builders that includes the Vantage Token Stream implementation, file storage, and Vantage Query engine.

This document describes the architecture, walks through a complete scenario — the Zeus Bot command-and-control investigation introduced in *When the Breach Happens, Is Your Data Already There?* — and shows how the same stream feeds a SOC analyst, an ML anomaly model, and a behavioral summarizer from a single capture, with honest provenance on every enrichment. It covers the storage model, the three consumption surfaces (virtual `.tokens` file, `/dev/vantage/{capture-name-or-id}/analysis` device, and `TokenChannel<T>` SDK API), and the security model that makes the stream safe to feed into AI pipelines running under different clearance levels than the capture environment itself.

We do not ship an ML component. That is deliberate. We ship the substrate every AI-defense model wants to consume, in a form that makes the model's job as easy as it can be made.

1. The Kill Chain Reframe Is Correct but Incomplete

The cyber kill chain was an interruption model. Its value came from the assumption that an attacker moved through discrete phases — reconnaissance, weaponization, delivery, exploitation, installation, command-and-control, objectives — and that defenders had observable time between phases in which to intervene. Break the chain at any phase, stop the attack.

The chain as an interruption model assumed a latency budget. AI-speed attackers have taken that budget to zero. When reconnaissance to exploitation compresses from days to minutes, and when target selection and payload generation happen inside a single automated workflow, there are no observable windows between phases. The chain does not break. It collapses into a single event from the defender's perspective.

The reframe that follows from this is not about faster detection. Faster detection inside the same interruption model still loses. The reframe is that defense has to become a state-management problem: not *"can we catch them mid-chain"* but *"have we reduced the conditions that make any chain viable."* Continuous posture assessment. Continuous configuration validation. Continuous behavioral baselining at a granularity the attacker operates at.

This reframe is gaining traction for good reason. It survives the AI-speed assumption in a way the interruption model does not.

But the reframe has a gap, and the gap is what this whitepaper is about.

State management at machine speed is not achievable through quarterly audits or weekly configuration reviews. It requires a continuous, structured, labeled observation stream produced at the granularity the attacker operates at. Whatever consumes that stream — a rules engine, a posture analyzer, a human SOC team, an ML anomaly model, a large language model producing narrative summaries — is only as useful as the stream feeding it.

> *The reframe to state management describes the destination. It does not describe the road.*

The conversations I have with security leadership teams increasingly arrive at state management as the answer. The conversations almost never continue on to the question that naturally follows: what does the data layer underneath that state look like? What must be produced, at what rate, in what form, with what labels, with what provenance, to make the rest of the pipeline work?

That question is the subject of everything below.

2. The AI-Native Gap

I attended RSA three weeks before writing this document. By a rough count, seventy-five percent of the show floor had some version of an AI-native pitch: AI-native detection, AI-native triage, AI-native response, AI-native SOC automation. The proposition in nearly every case was end-to-end platform: the vendor's model, running on the vendor's ingestion, trained on the vendor's data, with the vendor's governance story.

Almost nobody was selling the substrate the model requires.

Teams I talk to who have deployed one or more AI-native platforms tend to describe the same patterns. The model performs well in the vendor's demo, which uses the vendor's training distribution. It performs less well in production, where the actual traffic does not match that distribution. There is no way to inspect what the model is seeing, because the pipeline between capture and inference is opaque — usually proprietary, usually a black box, usually protected as a competitive differentiator by the vendor.

The governance story is worse. Running a model across an enterprise's packet traffic requires feeding that traffic — which includes subscriber PII, authentication tokens, internal service identifiers, and sometimes payment or health data — into the vendor's environment. The enterprise inherits the vendor's data-handling posture, the vendor's cloud footprint, the vendor's sub-processor agreements. Data-governance reviews on AI-native defense platforms fail at a higher rate than teams publicly acknowledge. The review usually fails on the same question: *where does our data go, and under whose enforcement?*

The gap is structural, not cosmetic. An AI model is only as good as the stream feeding it, and the stream feeding a typical AI-native platform is whatever the vendor decided to build — which is almost never optimized for the model, usually optimized for the vendor's operational convenience, and rarely instrumented with provenance a downstream consumer can trust.

A different approach starts from the stream instead of starting from the model. Decide what the feature stream needs to look like to support any model a customer might want to run. Build that stream. Let the customer bring the model, run it where their governance permits, and consume the stream through whatever interface fits their pipeline.

That is the approach this document describes.

3. What a Feature Stream Actually Needs

A feature stream for AI-ready network defense has five non-negotiable properties. These are not design preferences; each one, if absent, forces a compromise somewhere else in the pipeline that compounds.

3.1 Compact enough to not be the bottleneck

If the stream saturates the pipeline that produces it, everything downstream slows to accommodate. The header that describes every token is four bytes: the width of a modern CPU register, aligned with cache line boundaries, copyable and comparable at the natural word size of the machine. Emission, enqueue, dequeue, and comparison all happen at register speed.

Control tokens that encode only state transitions and flags stay at four bytes total — the header is the whole token. Tokens carrying analytical payloads — flow keys, sequence numbers, protocol outcomes — typically extend to eight or sixteen bytes. Richer payloads (URLs, tuple keys, identifier fields, protocol-specific metadata) are supported up to 65,535 bytes via an extended format, but are the minority of the stream by count. The design principle: whatever can be represented in four bytes is represented in four bytes. Everything else pays only for the bytes it carries.

The practical consequence is that the stream travels at wire speed. At 10 Gbps of sustained capture, the stream produces a volume one to two orders of magnitude below the packet rate — tens of thousands to a few hundred thousand tokens per second. Consumable bandwidth on any modern hardware, not a saturation risk.

3.2 Labeled at emission, not after the fact

Two bits of every token's header encode a severity classification: informational, normal, warning, or anomaly. The classification is set by the analyzer that emitted the token, using the same logic the analyzer was already applying to decide whether the event was interesting. An IDS signature match emits with anomaly status. A TCP retransmit emits with warning status. A normal HTTP request-response pair emits with informational status.

This is a labeled dataset produced at line rate by the analyzer. It is not a training set extracted after the fact by humans triaging alerts. The labels are present the moment the token lands in the stream, which means training an anomaly detector on the stream requires no separate labeling pipeline, no ticketing-system export, no human-in-the-loop labeling effort. The analyzer did the labeling at emission time, because it already knew.

For AI pipelines that need ground truth, this is the single most valuable property of the stream. Most ML projects in security die in the labeling phase. A stream that arrives pre-labeled by the sensor that produced it removes the most expensive part of the pipeline.

3.3 Self-describing

An eight-bit domain field and sixteen-bit type field identify every token. The domain groups the token by producer namespace; the type is domain-specific and identifies the exact event. The combined identifier tells a consumer what every token is without reference to a schema file.

This matters because it means the stream tolerates change. New analyzers and new SILO integrations can be added and will emit into new domains or new types within existing domains.

Existing consumers ignore domains and types they don't recognize. There is no reindex, no schema migration, no format version to coordinate across the fleet. The stream evolves additively, and deployments on different versions of the analyzer library or different SILO configurations can coexist.

The domain address space is eight bits — 256 primary domains — with an extension mechanism available for deployments that need more space or SILOs with large sub-domain requirements. The current allocation uses a small fraction of the primary space, leaving generous headroom for SILO growth and custom integrations.

3.4 Stratified by audience

Two additional bits of the header distinguish tokens by intended level of detail: advanced, normal, friendly, or learning. An advanced consumer sees full protocol detail. A learning-level consumer sees high-level narrative summaries. A normal-level consumer sees what falls between.

The same stream can feed a fine-grained protocol anomaly detector (advanced-level consumption) and a natural-language incident summarizer (learning-level consumption) without any re-extraction. Each consumer filters the stream to its level at read time; the analyzers emit at all levels because the work is cheap and having the detail available is more valuable than saving a few bytes per token.

For AI pipelines that want to serve both technical and non-technical audiences, this is the property that makes a single stream do the work of what would otherwise be two or three separately extracted streams.

3.5 Semantic events, not per-packet metadata

The stream does not emit a token for every packet. That would be noise. It emits a token when analyzer state transitions: a flow opens, a segment arrives out of order, a TLS handshake completes, a DNS query is answered, an HTTP response closes, a flow closes with the summary the analyzer had been accumulating.

The tokens are semantic events, carrying the analytical state the analyzer already tracks. When a TCP flow closes, the closing token carries the flow's summary metrics — duration, packet count, byte count, retransmit count — because the analyzer was maintaining them. The token is not re-computing anything; it is emitting the state the analyzer was about to discard.

This is what keeps the stream's rate below the packet rate by an order of magnitude or two. Noise is suppressed structurally, not by post-hoc filtering.

4. The jNetWorks Token Stream (Source Layer)

The stream produced directly by the Protocol Stack is the **jNetWorks Token Stream**. It is the lower-tier source from which the Vantage Token Stream is built, and it is also available as a standalone product for platform builders licensing the jNetWorks SDK.

4.1 How the stream is produced

Packets enter the capture pipeline from a port (live capture) or a file (offline capture). The Protocol Stack decodes each packet through its layers: Ethernet, IP, TCP or UDP, and above that the application protocol — HTTP, TLS, DNS, and others. As the decoder traverses each layer, the corresponding analyzer updates its internal state: flow tables, sequence tracking, handshake state machines, request-response pairing.

When an analyzer's state transitions — a new flow is allocated in the table, a sequence gap is detected, a handshake completes, a request-response cycle closes — the analyzer emits a token describing the transition. The token is written to the stream synchronously with the state transition. No post-processing, no batching, no separate pipeline. The state transition and the emission are the same action.

Alongside the strict protocol RFC-compliance analyzers, the Protocol Stack includes a set of lightweight built-in IDS analyzers that detect common attack vectors — network scanning patterns, port sweep signatures, unusual protocol behavior indicative of reconnaissance. These analyzers are optional and deliberately scoped: the jNetWorks capture pipeline commits to sustained throughput SLAs up to 800 Gbps, and the built-in IDS set is calibrated to hold those SLAs. Deployments that need deeper inspection can attach custom analyzers directly into the Protocol Stack, or can pair jNetWorks capture with external IDS systems (Suricata, Zeek) integrated through the Vantage SILO layer described in Section 5.

4.2 Token structure

A token carries a four-byte header containing four fields:

- **Domain (8 bits)** — the namespace that scopes the type field. Primary protocol examples: TCP, TLS, HTTP, DNS, PII, CONTROL. The domain space also accommodates SILO-originated tokens (see Section 5), each external SILO occupying its own domain.
- **Type (16 bits)** — the domain-specific identifier of the event. Examples within the TCP domain: STREAM_SYN, SEGMENT_OUT_OF_ORDER, FAST_RETRANSMIT, DUPLICATE_ACK, WINDOW_RESIZE, STREAM_RST, STREAM_FIN.
- **Status (2 bits)** — the severity classification: INFO, NORMAL, WARNING, ANOMALY.
- **LOD hint (2 bits)** — the intended audience level: ADVANCED, NORMAL, FRIENDLY, LEARNING.

- **Length (4 bits)** — the total token size in 32-bit words, or zero to signal an extended format where a 16-bit length field follows.

Tokens that carry only state transitions and flags are four bytes total. Tokens that carry analytical state (flow keys, sequence numbers, timing deltas, protocol outcomes) extend to eight or sixteen bytes. Tokens that carry variable-length payloads use the extended format.

When a consumer needs to correlate a token back to the raw packet that triggered it, tokens that reference packet content carry a frame number pointing directly at the packet index in the underlying capture. Most consumers do not need this — the analytical state is already in the token — but the bridge back to raw bytes is available when it is.

4.3 Protocol coverage

Analyzers across the Protocol Stack emit into the same stream:

- **TCP tokens** describe flow lifecycle events, ordering and congestion events, and flow closure summaries.
- **TLS tokens** describe handshake events, cipher suite negotiation, certificate details, and downgrade or anomaly conditions.
- **HTTP tokens** describe request and response structure, method, status code, and header anomalies.
- **DNS tokens** describe query and response events, including tunneling signals, NXDOMAIN patterns, and fast-flux indicators.
- **PII tokens** describe identifier-bearing fields detected in captured content: IP addresses, subscriber identifiers, email addresses, session IDs. PII tokens are encrypted at capture time under the deployment's security policy (see Section 8).

One domain per protocol, one stream unifying all of them. A TCP flow carrying an HTTP request over TLS produces tokens from four domains simultaneously: TCP (flow lifecycle), TLS (handshake outcome), HTTP (request structure), and potentially PII (if identifier-bearing fields are detected). The stream is the interleaved emission from all analyzers running against the capture.

4.4 Bitmap gating

Token generation is gated by consumer demand. Each domain carries a 64-bit bitmap — one bit per token type — that tells the analyzer which types to emit. Analyzers check the bitmap before emitting; types with their bit cleared are not produced.

When a consumer attaches to the stream, the attachment registers a subscription. The subscription is expressed as a bitmap per domain — the consumer's declaration of which token types it wants. The analyzer's effective bitmap is the union of all registered subscriptions. Types nobody subscribes to cost zero to skip.

This has two practical consequences. First, the stream self-tunes to what is actually being consumed — there is no wasted emission for types no downstream pipeline cares about. Second, adding or removing a consumer does not require reconfiguring the analyzer; the subscription mechanism handles it automatically.

4.5 In-stack composability

Higher-level jNetWorks analyzers can consume the stream from lower-level analyzers and emit their own derived tokens, which land in the same stream structure and are indistinguishable in format from the primary emissions. A TCP-level summary analyzer, for example, consumes the primary TCP tokens and emits periodic flow-state summaries derived from them.

In-stack composability is an SDK-level capability — analyzers running inside the Protocol Stack, within a single capture pipeline, composing together. The higher-tier composition, where external SILO sources are joined to the packet stream via correlation, is described in the next section. That tier is where the Vantage Token Stream's real analytical power comes from.

4.6 Heritage

The Token Stream shipped in jNetWorks v3.0.0, early 2026. Sly Technologies has been building packet capture and analysis infrastructure since 2005; the jNetWorks SDK itself has been in production since 2013. The Token Stream is a new v3 feature, but the analyzer state it exposes is not new — the Protocol Stack has been maintaining that state internally for more than a decade. v3.0.0 formalized it as a first-class consumable output.

> *We did not invent this in response to the AI-native wave. The AI-native wave revealed a capability the Protocol Stack already had.*

This distinction matters for buyers evaluating the stream against AI-native platforms pitched with comparable claims. A feature stream retrofitted to meet a recent market demand is a different artifact than one that emerges naturally from analytical infrastructure that has been running in production for a decade. The properties described in Section 3 are not design goals we set out to achieve; they are properties of the analyzer architecture, expressed in a form consumers can now use.

5. The Vantage Token Stream (Enriched Layer)

The **Vantage Token Stream** is the jNetWorks Token Stream enriched by Vantage's SILO correlation layer. It is the operator-facing stream — the one a SOC analyst, an ML inference pipeline, or a behavioral summarizer running in a production Vantage deployment actually consumes. Everything from the jNetWorks tier is carried through; additional tokens are emitted by the correlation layer as external sources are joined to the captured packet context.

This is where most of the value lives. The jNetWorks tier answers *what happened in this packet flow.* The Vantage tier answers *what does this flow mean in the context of every other signal we have* — the IDS alerts firing against it, the threat intel the destination IP appears in, the NetFlow records from neighboring infrastructure, the campaign attribution from external feeds.

5.1 The SILO bridge

Vantage organizes external data sources into SILOs (Source Integration Layer Overlays), described in *Why Querying 100 EB Takes Seconds*. Each SILO represents a distinct category of non-packet data:

- **Analysis SILOs** — IDS systems like Suricata and Zeek, producing alerts from their own rule engines.
- **Telemetry SILOs** — flow records from NetFlow, IPFIX, and cloud flow logs.
- **Enrichment SILOs** — threat intelligence and annotation feeds: VirusTotal, MISP, MaxMind, internal watchlists.
- **Capture SILO** — the anchor, holding the raw packet data and the jNetWorks Token Stream.

The Vantage SILO bridge is the machinery that joins external SILO events to the packet stream. Events arriving from an Analysis, Telemetry, or Enrichment SILO are matched against live packet context held in a per-worker correlation cache — keyed by timestamp, frame number, and flow tuple — and when the match succeeds, a new token is emitted into the Vantage Token Stream binding the external event to the specific packet flow it corresponds to.

The correlation cache holds packets that recently flowed through the associated packet channel, load-balanced across workers so that the total traffic stream is covered. Each correlator runs single-threaded per worker against its own cache partition, which is how the mechanism sustains correlation throughput up to 800 Gbps of captured traffic. The per-worker design is deliberate: lock-free, predictable, and horizontally scalable with the capture pipeline itself.

5.2 Source-named domains

Each external SILO source emits into its own dedicated domain. SURICATA for Suricata IDS alerts. ZEEK for Zeek events. VIRUSTOTAL for VirusTotal enrichment. MISP for MISP threat intelligence. MAXMIND for geographic enrichment. NETFLOW and IPFIX for telemetry records.

This keeps the subscription surface clean. An ML pipeline that wants to train on IDS-grade ground truth subscribes to SURICATA and ZEEK. A threat-intelligence enrichment consumer subscribes to VIRUSTOTAL and MISP. A flow-volume anomaly model subscribes to NETFLOW and IPFIX. Downstream consumers can filter by source with a single domain bit in their subscription bitmap, rather than chasing composite type fields.

The domain address space has ample room for SILO growth. The current allocation uses a small fraction of the available domains; new SILO integrations add new domains without affecting existing consumers. Deployments with unusual sub-domain needs — for example, a custom threat intel integration with many internal categories — can use the domain extension mechanism to carry sub-domain addressing within the token payload.

5.3 Events that don't need correlation

Not every external SILO event corresponds to observed packet traffic. An IDS health check, a threat intel feed update, an administrative event from a SILO operator — these events carry meaning but don't bind to a specific packet flow. They emit as normal tokens in their source domain, bypassing the correlation cache, with no packet-stream reference attached.

Consumers can filter by this distinction if they want to — a token without a frame reference is a free-standing event; a token with one is an event bound to specific captured traffic. Both are valid; which matters depends on the consumer's use case.

5.4 Carrier tokens and the confidence lifecycle

Correlation against a live packet cache is best-effort in one specific sense: if an alert arrives after the packet window it corresponds to has already left the cache, the correlator cannot complete the binding synchronously. This happens routinely — IDS systems sometimes emit alerts with seconds or minutes of latency relative to the originating packet.

When this happens, Vantage emits a **carrier token** — a token in the correct source domain with a placeholder binding, tagged with a confidence state that tells consumers how reliable the correlation is:

- **FULL** — the correlator matched the external event against a live cache hit with complete state. The binding is verified and specific.
- **PARTIAL** — the correlator matched, but some upstream state is incomplete. The binding is probably correct; downstream consumers should treat the token as useful but not authoritative.
- **INFERRED** — the correlator could not make a direct match but emitted a token based on structural similarity to a confirmed event. Useful as a hypothesis; consumers that can't tolerate uncertainty should filter these out.
- **UNKNOWN** — the correlator emitted a placeholder pending resolution. Real correlation will come later, if it comes at all.

The original external event — the raw Suricata alert, the VirusTotal lookup result — can be attached as payload inside the carrier token. When resolution completes, a refined token supersedes the carrier: same packet reference, confidence elevated, binding pinned to the specific frame. Consumers that re-read the stream after resolution see the enriched picture; consumers that read earlier see the carrier with its honest confidence marker.

5.5 Retrospective correlation

Carrier tokens and post-emission refinement work because of the storage model. The .events file that holds both tiers of the stream uses an append-anywhere structure — any beaconed window can receive additional tokens after the fact, linked via offset references. The Query Analytics Tree applies the same pattern: any node can receive additional keys after its initial population.

This means correlation that failed against the live cache can be retried against stored packet data. A reprocessing pass reads historical captures, replays the external event against the stored flow, and emits refined tokens with higher confidence into the same .events windows where the original carriers landed. The QAT picks up the new keys and the queryable surface grows more accurate over time.

Deployments can schedule this reprocessing on their own cadence — continuously in the background, opportunistically after a high-value alert, as part of a retention-tier migration, or not at all. The format supports all of these modes without forcing a choice.

5.6 Provenance is a first-class property

The consequence of the confidence model is that the Vantage Token Stream carries honest provenance on every enrichment. A consumer reading a VIRUSTOTAL domain token can tell at a glance whether the lookup completed against live traffic with full binding (FULL), whether the correlation was made after the packet window had passed (FULL against stored data, or INFERRED if stored data wasn't available), or whether the token is a placeholder waiting for resolution (UNKNOWN).

For AI pipelines, this matters. An anomaly detector can weight inputs by confidence. A training pipeline can restrict to FULL-confidence tokens for its ground truth set. An alerting system can raise confidence-adjusted severity. The stream gives the consumer the information needed to handle uncertainty explicitly, rather than hiding it inside the enrichment layer and forcing the consumer to trust the vendor's claims.

This is the kind of property AI-native platforms rarely expose, because doing so requires admitting when an enrichment is best-effort rather than authoritative. The Vantage Token Stream exposes it by design.

6. Storage and Consumption

The Token Stream — both jNetWorks primary tokens and Vantage-emitted correlation tokens — has one authoritative stored form and three consumption surfaces. All three surfaces expose the unified stream; they differ in how the consumer attaches.

6.1 The .events file

The authoritative stored form is the .events file, written alongside the capture. The file is structured as a sparse beacon tree: raw tokens are written in emission order, interleaved at regular window boundaries with beacon records containing navigation keys and references to the previous and next beacons. The beacons allow a consumer to seek to a specific time range or packet window without scanning the entire file; the full keyset used by the Query Analytics Tree remains in the QAT itself, but the beacon tree in .events provides rapid access to raw token data at window granularity.

The file is append-anywhere. Vantage-tier correlation tokens emitted after the original window — either synchronously from the SILO bridge or retroactively from a reprocessing pass — are linked into their target windows via offset references. A consumer reading the stream sees all tokens that landed in a given window, regardless of when they were emitted, in a consistent order determined by the file structure.

6.2 As a virtual file: .tokens

For consumers that want to treat the Token Stream as a file, Vantage exposes a .tokens projection over the .events file. The .tokens file is virtual: it is not a physical file on disk, but a view computed on the fly that flattens the beacon tree into a pure token stream with the navigation structure stripped.

Reading the .tokens file returns the token sequence in emission order, with zero additional storage overhead. Multiple consumers can hold independent .tokens views against the same .events file simultaneously. The view persists for as long as the consumer needs it; the underlying .events file continues recording or being retained under normal policy.

Consumers read the .tokens file using any tool that reads a byte stream. cat, tail -f, dd, head — standard Unix utilities work without modification. For programmatic consumption, the file is a normal file handle: open it, read it, process the token structure inline. Feed it into Splunk, a Python dataframe, a Kafka topic, an ML feature store, or any other pipeline that accepts a structured byte stream.

6.3 As a device: /dev/vantage/{capture-name-or-id}/analysis

For live streams, Vantage exposes the Token Stream as a device handle at /dev/vantage/{capture-name-or-id}/analysis. The device is opened with a standard open() call and read with read(); the bytes delivered are the same tokens being written to the .events file, streamed in real time as they are emitted.

Streaming inference pipelines attach directly to the device. No intermediary translation layer, no message broker, no format conversion. Tokens arrive at the reader at the same moment they are produced by the analyzers or by the SILO correlation bridge, with minimal buffering overhead.

Reading from the device gives a live view; reading from the .tokens virtual file gives the historical view. Both surfaces expose the same stream — one as it is being produced, the other as it was stored.

6.4 As an SDK API: TokenChannel<T>

For teams building their own capture pipelines at the protocol level, the jNetWorks SDK exposes typed, programmatic token channels. The channel is attached to a capture as part of the pipeline configuration:

```
TokenChannel<TcpToken> tcpTokens = net.tokenChannel("analysis-tokens", TcpToken.class);

Capture capture = net.capture("tcp-reassembled", "en0")
   .filter(PacketFilter.tcp())
   .assignTo(tcpChannels)
   .assignTo(tcpTokens)
   .protocol(stack)
   .apply();

executor.fork(tcpTokens, this::analyzeTcpTokens);
```

The channel is typed to a specific token domain, giving the consumer compile-time type safety. Parallel workers can be forked against the channel; the SDK combines emissions from multiple worker threads into a single unified stream.

The SDK API is the path for OEM integrations and platform builders licensing the jNetWorks SDK directly. A vendor building an AI-native defense product on top of jNetWorks uses the SDK to attach token channels, route them into their pipeline, and build their own analyzers and correlation logic on top of the primary stream.

6.5 Subscription and generation

Each consumption surface registers a subscription. The virtual .tokens file, the device handle, and the SDK channel all translate into bitmap subscriptions at the analyzer and SILO correlation layers. Types nobody subscribes to are not produced.

The practical implication: attaching a narrow consumer — for example, an ML pipeline that only cares about anomaly-status tokens in TCP, TLS, and SURICATA domains — is not just efficient at the reader. It suppresses emission at the source. The stream's cost scales with consumer demand.

7. A Complete Scenario: The Zeus Bot Investigation

To make the architecture concrete, this section follows the Token Stream through a complete incident scenario — the same Zeus Bot command-and-control investigation described in *When*

the Breach Happens, Is Your Data Already There? That scenario followed a SOC analyst from an IDS alert at 14:23 on a Tuesday through a 24-minute incident response workflow ending with a chain-of-custody evidence bundle.

This section follows the same event, but from the Token Stream's perspective: what the jNetWorks tier produced at capture time, what the Vantage tier added as correlation completed, and how three different consumers operated against the same underlying bytes.

7.1 Day 1: Capture-time emission from the jNetWorks tier

The C2 beacon from host 192.168.1.100 to 185.234.72.19 on port 80 hit the capture interface on day 1 of the 14-day rolling window — ten days before the IDS signature match that would eventually surface it. The Protocol Stack processed the flow. Neither the flow itself nor any single packet in it triggered an immediate alert from the built-in lightweight IDS analyzers.

What the Protocol Stack did produce was a sequence of jNetWorks Token Stream emissions. A representative subset:

- TCP STREAM_SYN — new flow allocated, source 192.168.1.100, destination 185.234.72.19:80, status NORMAL.
- HTTP REQUEST_START — POST request to /gate.php, status NORMAL, URI carried as extended token payload.
- HTTP REQUEST_NO_USER_AGENT — the request lacks a User-Agent header, status WARNING. The analyzer flagged an unusual structural property at emission time.
- HTTP RESPONSE_STATUS_200 — 200 OK response received, status NORMAL.
- TCP STREAM_FIN — flow closed after 0.107 seconds, 2 packets, 699 bytes, status INFO. The closing token carries the flow's summary metrics.
- PII EXTERNAL_DESTINATION — destination IP is outside the internal address space, status INFO, destination IP encrypted under the deployment's clearance policy.

None of these tokens individually constitutes an alert. The WARNING on the missing User-Agent is a signal available to downstream consumers that choose to attend to it. No Vantage-tier correlation tokens are emitted at this point, because no external SILO event has arrived binding to this flow.

7.2 Storage

The tokens land in the .events file in emission order. The beacon tree places them within a specific time window, with frame-number references pointing at the raw packets in the underlying PCAP. The corresponding keys are promoted into the QAT tree, which indexes the full 14-day capture.

7.3 Day 11: The IDS alert fires

On day 11, a later C2 beacon from 192.168.1.100 to the same destination triggers the Suricata signature for Zeus Bot command-and-control. The alert arrives at the Vantage SILO bridge carrying the IDS rule ID, the matched packet identifier, and the alert metadata.

The Vantage correlator examines the alert's timestamp and tuple against its live packet cache. The day-11 flow is still present in the cache — the alert's latency is milliseconds, well within the cache window. The correlator finds the match, emits a **FULL-confidence** SURICATA **token** into the Vantage Token Stream, bound to the specific frame number of the alert-triggering packet, with the Suricata rule ID and alert details attached as payload.

This is the token the SOC analyst's tooling surfaces at 14:23. The analyst sees a high-confidence IDS hit bound to a specific flow, with the originating rule metadata attached. The 24-minute incident response workflow described in the earlier whitepaper is triggered.

7.4 Retrospective correlation on the day-1 flow

The analyst's navigation backward through the QAT lands on the day-1 beacon flow — the original reconnaissance event that predates the day-11 alert by ten days. At this point, the day-1 packet window is still retained (the rolling buffer covers 14 days), but the packets are long past the live correlation cache.

Vantage's reprocessing pass — triggered either by the investigation, by a scheduled background task, or by a retention-tier migration — examines the day-1 flow against the newly-confirmed day-11 campaign context. It emits new tokens into the day-1 window:

- A retrospective SURICATA token, bound by frame number to the day-1 beacon, with confidence FULL because the correlation completed against stored packet data that was still available. The payload carries the Suricata rule's structural match against the day-1 flow and a reference to the confirmed day-11 campaign.
- A VIRUSTOTAL token for the destination IP 185.234.72.19, carrying the VT lookup result. Confidence FULL for the destination-IP binding (a direct lookup). The campaign attribution portion of the VT response is carried as INFERRED, because the specific campaign tag depends on VT's own attribution confidence rather than on packet-stream verification.
- A MISP token with the campaign tag. Confidence FULL for the destination-IP membership in the MISP feed, which is a direct lookup. Any cross-campaign linkage is carried separately at its own confidence.

These tokens are appended into the day-1 .events window via the append-anywhere mechanism. The QAT node covering that window picks up the new keys. A consumer querying day 1 after reprocessing sees the enriched picture — a flow that looked routine at original emission time now carries IDS correlation, threat intelligence, and campaign attribution, each with honest confidence markers visible at read time.

7.5 Three consumers, one stream

With the stream in this state — jNetWorks primary tokens from day 1, Vantage correlation tokens from both day 1 (retrospective) and day 11 (live) — three different consumers can operate simultaneously:

Consumer 1: The SOC analyst. Navigates via Vantage Query, querying the QAT for flows binding SURICATA alerts to specific hosts and traversing backward from the day-11 alert to the day-1 beacon. The 24-minute incident response workflow runs against this enriched view.

Consumer 2: An ML anomaly model. An anomaly detection model running in the enterprise's ML pipeline attaches to the Token Stream via the virtual .tokens file. It subscribes to tokens with WARNING or ANOMALY status across all protocol domains, and to all SURICATA and ZEEK tokens at FULL or PARTIAL confidence. The model scores the day-1 flow based on its token sequence — the WARNING on the missing User-Agent, the short flow duration, the external destination combined with the retrospectively-added SURICATA and VIRUSTOTAL correlations — and produces an anomaly score that, in future similar traffic, would flag comparable flows earlier in their lifecycle.

Consumer 3: A behavioral summarizer. A large language model trained to produce natural-language summaries of network incidents attaches to the same stream but subscribes only to tokens with LEARNING LOD across all domains, at FULL confidence only (it cannot tolerate speculative enrichment). It produces a plain-English account: *"On day 1, an internal host initiated a short outbound HTTP POST request to an external server on an unfamiliar URI with no User-Agent identification. The destination was later confirmed to belong to a known command-and-control campaign. The exchange completed in approximately 100 milliseconds."*

Three consumers. One stream. One capture. The bytes were produced once at capture time and once more at correlation completion, as byproducts of the Protocol Stack and the SILO bridge running their normal pipelines. Each consumer attached at the surface appropriate to its operational context and subscribed to the token types, confidence levels, and LOD ranges relevant to its task.

7.6 What this shows

The scenario demonstrates a property that is hard to appreciate without seeing it worked through: the Token Stream is not a specialized output for a specific downstream consumer. It is a substrate against which multiple consumers, at different technical depths, with different governance postures, with different tolerances for uncertainty, can operate simultaneously.

The cost of adding a fourth consumer — say, a machine-readable compliance feed that watches for specific regulatory-relevant events at FULL confidence only — is a bitmap and confidence-filter subscription at one of the three surfaces. No new data product needs to be built. No new extraction pipeline needs to be written. The stream that already exists accepts another subscriber.

8. Security and Governance

Feeding a stream like this into an AI pipeline raises a set of governance questions that need to be addressed head-on. Any stream that carries PII, identifier-bearing fields, or protocol content sensitive to an organization's regulatory posture is only usable in AI pipelines if the governance story is airtight.

8.1 The Vantage security model

The Token Stream is governed by the Vantage security model, which applies a unified 0–20 clearance level scale to all captured data and derived analytical output. The clearance level drives encryption strength, obfuscation policy, redaction policy, and access control in a single policy dimension.

Levels 0–4 are clear or lightly-obfuscated, suitable for internal monitoring and development. Levels 5–8 apply AES-256-GCM encryption with PII tokenization, suitable for enterprise SOC environments. Levels 9–12 apply full encryption with HMAC integrity and full PII obfuscation, suitable for lawful intercept environments. Levels 13 and above apply progressively stricter controls up to fully air-gapped, HSM-backed operation.

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

8.2 Enforcement at the daemon, not the client

The security model's non-negotiable principle is that enforcement happens at the capture daemon, not at the client. A Lynx instance, an ML pipeline, a SIEM integration, or any other consumer of the Token Stream receives only the data its session's clearance permits. The data above the consumer's clearance is not filtered at the client — it is never transmitted to the client in a readable form.

For PII tokens specifically: the token is emitted and stored with its sensitive fields encrypted. When a consumer attaches, the daemon resolves the consumer's clearance against the token's classification and delivers the token with sensitive fields decrypted only if the clearance permits. A consumer at a lower clearance receives the same token structure with those fields redacted or zero-filled. The token sequence is the same; the visible content depends on who is reading.

This has a direct implication for AI pipelines. An ML model running in a lower-clearance execution context (a separate VM, a different network zone, a cloud-hosted inference environment) can consume the Token Stream without the underlying sensitive data ever leaving the daemon in readable form. The model trains on — and infers against — the projection of the stream its clearance permits.

8.3 Split clearance for AI pipelines

A SOC analyst operating at a high clearance level and an ML inference pipeline operating at a lower clearance level can attach to the same Token Stream simultaneously, and each reads a different projection.

The SOC analyst sees full protocol content, decrypted PII fields, and unredacted flow metadata. The ML pipeline sees the same token structure but with PII fields tokenized (consistent pseudonymized values that preserve joinability across tokens without exposing the underlying identifiers) and sensitive content redacted. The ML model trains on an enforcement-controlled projection of the stream, not on the raw underlying data.

This is the difference between sending sensitive data into an AI-native vendor's platform (where the governance story is the vendor's posture) and keeping enforcement at the source while the model consumes what it is entitled to consume. Teams that have been through a failed data-governance review on an AI-native platform understand why this matters.

8.4 Audit and session keying

Every access to the stream is logged to an immutable, cryptographically chained audit ledger. The ledger records the session, the clearance level, the tokens accessed, and the projection applied. For regulated environments — lawful intercept, financial services, healthcare — the audit trail is a compliance requirement, and it is produced automatically as a property of the enforcement pipeline.

Session keys are per-user and time-bounded. A credential compromise affects only the tokens accessible under that session's clearance during the session's validity window. Keys above the session's clearance level cannot be derived from the session's key material.

9. We Ship the Stream, Not the Model

Sly Technologies does not ship an ML component. That is deliberate. It is also, we expect, what the market will eventually ask for from every vendor in this space.

The AI-native vendor pitch — end-to-end platform, proprietary model, trust our detections — asks the buyer to take the model on faith, move data into the vendor's pipeline, and accept the vendor's latency, governance story, and lock-in. The buyer gets a black box with a service agreement.

The alternative is to separate the stream from the model. The stream is infrastructure: it has properties that can be specified, verified, and held to a standard. The model is a consumer of that infrastructure: it can be swapped, retrained, audited, and replaced without disturbing the stream. A buyer who adopts a Token-Stream-based architecture is not locked into any particular model; they are locked into a stream interface that every model can consume.

We ship two streams. The **jNetWorks Token Stream** for platform builders licensing the jNetWorks SDK — the raw, protocol-level stream produced directly by the Protocol Stack, suitable for vendors integrating packet analytics into their own products. The **Vantage Token Stream** for operators deploying the Vantage Platform — the enriched stream with SILO correlation, carrier-token confidence lifecycle, and the full retrospective correlation machinery, suitable for SOC, MDR, and detection engineering use cases. Either way, no proprietary model is required. The stream stands on its own.

The conversion from Token Stream to the input format a specific ML model expects — embeddings, one-hot feature vectors, tokenized text, structured JSON — is a thin, model-specific bridge. Token type maps to feature index. Status bits map to anomaly labels. Confidence maps to training weight. LOD filter maps to sample stratification. PII token redaction maps to feature masking. The bridge is cheap to write and lives entirely on the consumer side; the stream properties that make the bridge easy are the hard part, and the hard part is done once, at the capture and correlation layers.

Training infrastructure can consume historical `.events` files directly, replaying tokens through the bridge in bulk. Online inference can attach to the live device handle and score tokens as they arrive. The same stream supports both modes without modification.

> *We ship the substrate. Bring your own model.*

Over time, we expect to ship our own analytical layers on top of the stream — that is already how Vantage's SILO correlation works. Whether we ever ship a first-party ML component is an open question. What is not open is the commitment to keep the stream interface stable and consumable by any model a customer wants to run. The stream is the product. The model is the customer's.

10. Getting Started

The Vantage Token Stream ships with the Vantage Platform, available now. Every capture — live or offline — generates the unified `.events` record automatically, including both jNetWorks primary tokens and Vantage correlation tokens as they complete. The virtual `.tokens` file and the `/dev/vantage/{capture-name-or-id}/analysis` device are available on every Vantage deployment.

For teams building their own capture infrastructure at the protocol level, the **jNetWorks SDK v3** is available now under internal and OEM license. The SDK bundle includes the Vantage Token Stream implementation, the `.index`/`.events` file storage layer, the Vantage Query engine, and the supporting storage modules — everything an SDK customer needs to produce and consume Vantage-compatible output from their own capture infrastructure. Full Vantage Platform adoption unlocks the enriched Vantage Token Stream with complete SILO correlation, the carrier-token confidence lifecycle, and the analytical tooling built on top.

Request a demonstration at slytechs.com/contact. We will walk through the Token Stream with your traffic, on your infrastructure, against the pipeline you already built.

Related reading

- *When the Breach Happens, Is Your Data Already There?* — the 24-minute forensic readiness workflow referenced throughout this document. Covers the full SOC-side incident response scenario.
- *Stop Rolling Over Your Evidence* — the intelligent retention architecture that keeps the .events record indefinitely at 0.59% metadata overhead.
- *Why Querying 100 EB Takes Seconds* — the Query Analytics Tree architecture that indexes the Token Stream for sub-second queries at any scale, and the SILO namespace model referenced in Section 5.

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