ChatGPT

Ephemeral GenAI SIEM: A Serverless, Graph-Driven Approach to Security Event Management

Executive Summary

Security Information and Event Management (SIEM) tools have long been central to enterprise defense, but traditional SIEMs struggle with scale, context, and cost. They often ingest massive volumes of log data into centralized systems, generating thousands of alerts with little context – a high noise-to-signal ratio that overwhelms analysts [1]. Moreover, conventional SIEM pricing models charge by data volume, forcing organizations to retain only a fraction of telemetry (often <10%) to control costs [2]. The result is a brittle status quo: **SIEMs tell you what happened, but not why it matters or what to do next** [3].

This white paper introduces **Ephemeral GenAI SIEM**, a next-generation approach co-developed by Dinis Cruz and ChatGPT Deep Research. It leverages *serverless computing, semantic knowledge graphs, and generative AI (GenAI)* to redefine how security data is collected, analyzed, and acted upon. Key innovations and decisions include:

- **On-Demand Data Collection:** Instead of blindly ingesting all logs, Ephemeral GenAI SIEM dynamically **pulls relevant data directly from sources when an incident or query arises**. Logs and evidence remain at the source (or in cheap cloud storage) until needed, avoiding expensive centralization and retaining complete context. This drastically lowers data ingestion costs and ensures **nearly 100% of useful data can be analyzed** (no more guessing which logs to drop) [2].

- **Serverless & Ephemeral Architecture:** The system runs primarily on serverless functions and temporary analysis engines that spin up as needed and terminate when done. **Cloud object storage (e.g. Amazon S3) serves as the database**, storing raw inputs and intermediate results as versioned files [4]. Using the **LETS (Load, Extract, Transform, Save) pipeline** pattern, every processing step writes its output to persistent storage, enabling full traceability and reproducibility [5]. Compute instances hold data in-memory only for the duration of analysis, yielding a highly scalable, cost-efficient design with zero idle infrastructure.

- **Semantic Knowledge Graph Core:** All security data is converted into a **semantic knowledge graph** – a richly structured representation of entities (devices, users, IPs, malware files, etc.) and their relationships (accesses, communications, ownership, etc.). This graph-centric model, powered by the open-source **MGraph-DB** engine, allows lightning-fast in-memory queries while persisting to JSON files on disk/cloud for durability [6] [7]. Graph-based storage provides flexible, schema-less integration of diverse data sources and naturally captures context across events.

- **Generative AI for Enrichment (with Determinism):** GenAI (LLMs) is woven into the pipeline for tasks that normally require human-like reasoning – parsing unstructured logs, extracting entities, linking related events, or even generating plain-language summaries. However, unlike "black-box" AI, each LLM interaction is *controlled and sandboxed*. We use structured output

schemas to force the LLMs to return data as typed JSON objects [8], which are then merged into the graph. This ensures **explainable and repeatable results** (no unpredictable AI magic): the same inputs always produce the same graph outputs, satisfying security's need for determinism and auditability. Over time, as patterns emerge, many LLM-generated steps can be replaced with code for performance, using AI primarily where it adds unique value (e.g. interpreting novel data formats or natural language).

- **No-Code Development & Self-Evolving Workflows:** The entire stack is designed for agility. Engineers or analysts can define new data transformations or detection logic in plain language or high-level terms, and let the GenAI-assisted system implement them – drastically reducing coding effort. The system **evolves organically from each investigated incident**: when a new data source or correlation is encountered, a new "connector" or parser is quickly spun up (often with AI assistance) and added to the toolkit. Over time, the SIEM grows a library of modules for different log types and analysis routines. This *iterative, Wardley-mapped approach* ensures the solution continuously adapts to emerging threats and enterprise needs, without large upfront deployments.

- **Context-Rich Investigations and Outputs:** Ephemeral GenAI SIEM emphasizes **capturing context and evidence at every step**. Instead of a shallow alert saying "Malware X detected on Host Y", the system builds a knowledge graph trail: how the malware got there (e.g. delivered via email link), what it did, which systems were affected, who owns those systems, and what controls failed. This evidence-driven graph can be visualized to reveal the full attack path and mapped to frameworks like MITRE ATT&CK or MITRE D3FEND for tactic/technique context. Finally, the platform can automatically generate **stakeholder-specific reports** – e.g. a technical report for responders with all indicators and log references, an executive summary for a CISO focusing on business impact and remediation, and even tickets or action lists for IT owners. By aligning facts, hypotheses, and analysis in one graph, the SIEM not only detects events but helps answer *"Why does this matter and what should we do?"* – bridging the gap from data to decision.

In summary, **Ephemeral GenAI SIEM** offers a fundamentally new, cloud-native blueprint for security monitoring: one that scales by not scaling (it only uses resources when needed), that is open and extensible (built on open-source graphs and cloud storage), and that provides deep intelligence by leveraging graphs and AI. This white paper details the motivation, architecture, and workflow of this solution, demonstrating how it can drastically improve detection and response capabilities while controlling costs and complexity.

Introduction: Rethinking SIEM for Scale and Context

Traditional SIEM platforms were originally designed to aggregate logs into a central store, correlate events, and generate alerts for security teams [9]. This model made sense in the early 2000s, but it is straining under the weight of modern IT environments. Organizations today deal with **exploding data volumes, diverse infrastructure (cloud, on-prem, IoT, BYOD), and advanced threats** that mutate and move laterally [10] [11]. In this landscape, conventional SIEMs reveal several critical shortcomings:

- **Volume and Cost Constraints:** SIEMs typically charge based on the volume of data ingested or indexed, making it prohibitively expensive to centralize everything [2]. To manage costs, teams end up filtering out or sampling logs – as a result, as little as *5–10% of security-relevant data* might actually be analyzed [12]. Even with unlimited license options, the overhead of moving and

storing petabytes of data is immense. The outcome is dangerous blind spots: if an organization must *guess* which logs matter, important signals can be missed entirely.

- **Alert Overload, Little Insight:** A classic SIEM may spit out thousands of alerts per day, yet only a handful represent true incidents [13]. The rest are false positives or low-context notifications that *"something happened"* without clarity. Analysts spend hours triaging noisy alerts, manually stitching together clues from disparate tools to understand if an alert is important [14]. Valuable time is lost hunting for context – which user was involved, what system, what payload, was it part of a broader campaign? SIEMs provide scant help here: they **tell you an event occurred but not why it matters** [3] or what next steps to take.

- **Lack of Contextual Intelligence:** Traditional SIEM correlation is largely rule-based and signature-driven. It flags known bad patterns but often misses the forest for the trees. **Threat context is lacking** – e.g., a SIEM may not automatically enrich an alert with threat intelligence (is that IP known malicious?), asset criticality (is the target a crown jewel system?), or user behavior (is this login abnormal for the user?) [15]. This absence of enrichment means responders start investigations from scratch, querying multiple systems to gather context that the SIEM didn't provide.

- **Integration Gaps:** Modern enterprises have sprawling tech stacks, including cloud services, containers, SaaS applications, and remote endpoints. Many SIEMs struggle to ingest or make sense of telemetry from all these sources [11]. Proprietary log formats, API limitations, or sheer data velocity (e.g. cloud audit logs) mean that the SIEM might simply not include those feeds, leaving blind spots. Integration projects to cover these often prove complex and fragile.

- **Scaling and Performance Limits:** As data volume increases, centralized SIEM architectures can choke. Indexing and querying large datasets becomes slow, and correlation rules may fail to keep up in real-time [16] [17]. Some organizations find they must **delay analysis or drop data just to maintain SIEM performance** [18]. In the worst case, security teams start avoiding certain queries because they are too slow or too costly to run on the SIEM – effectively narrowing their visibility to meet tool limitations.

In short, **the classic SIEM approach is showing its age**. As one industry observer put it, *"You're collecting logs, but not catching threats."* [19] The gap between data collection and actionable intelligence is widening. To close this gap, a new approach is needed – one that can economically harness all relevant data, provide rich context and explanations, and flexibly adapt to new data sources and attack techniques.

Design Principles of an Ephemeral GenAI SIEM

Ephemeral GenAI SIEM is designed from the ground up to address the above challenges. It is not a single monolithic product, but rather an *architecture pattern* and workflow enabled by cloud infrastructure and AI. The following core principles guide its design:

- **1. Data Stays at the Source (Until Needed):** Instead of funneling all logs into one expensive datastore, this approach queries data **in-place**. Wherever possible, logs and events remain in their original repositories (e.g. in an application's database, a SaaS platform's audit logs, a cloud storage bucket of raw logs). When an alert or question arises, the system **dynamically loads only the pertinent data** for analysis, rather than pre-ingesting everything. This on-demand model means we incur cost and effort *proportionate to actual investigative needs*, not worst-case

data volumes. The more we can analyze data at the edge or in its native storage, the more we avoid duplicated storage and transfer. Modern cloud environments even allow running code close to data (e.g. AWS Lambda functions near an S3 bucket). By **treating storage as the primary database and minimizing data movement**, we achieve both cost savings and access to complete, high-fidelity data [20] [4].

- **2. Layered ETL with Full Traceability (LETS):** We adopt the **LETS (Load, Extract, Transform, Save) pipeline** architecture [5], a variant of ETL tailored for clarity and reuse. Every stage of data processing is encapsulated in a module that *loads* raw input (from a source or previous stage), *extracts* structured information, *transforms* it (e.g. into graph format or by correlation), and *saves* the result as a new dataset. Crucially, **each stage's output is written to persistent storage (files or objects), becoming an immutable, versioned artifact** [4]. For example, Stage 1 might pull raw firewall logs for a specific hour and save them as a file; Stage 2 parses those logs into a JSON graph and saves that, and so on. This design has several benefits:

- *Determinism & Reproducibility:* Because each step is saved, we can rerun any or all stages at any time, given the same inputs, and get the same result. The entire incident analysis can be reconstructed from original sources, supporting forensic needs and "infrastructure as code" style reproducibility.
- *Explainability:* Analysts can inspect the outputs of each stage to understand how a conclusion was reached. If an alert is generated, one can trace back through the saved intermediate artifacts to see the supporting evidence (e.g. which log line triggered it, which correlation connected it).
- *Fault Tolerance:* If a pipeline fails midway (or if new data arrives), you can resume or re-run from an appropriate stage rather than starting over from scratch.
- *CI/CD Integration:* Pipelines can be version-controlled and tested step-by-step. Each transformation stage can have unit tests using the saved outputs as fixtures. This is a **clean, modular alternative to opaque "black box" SIEM logic** – akin to how modern data engineering values transparency [5].

- *Ephemerality:* Since data at each stage is saved to durable storage, the compute that performed the transformation can be safely destroyed after completion. No process needs to hold state in memory for long. This enables the heavy use of serverless functions and short-lived containers for processing, which scale out on demand and incur no cost when idle.

- **3. Semantic Knowledge Graph Data Model:** Under the hood, all extracted security information is represented as a **knowledge graph**. This means events, entities, and relationships are nodes and edges in a graph structure, annotated with properties (timestamps, IDs, etc.) and linked to one another. For example, an "Alert" node might connect to a "File" node (the malware sample) which connects to a "Host" node (the affected laptop) and a "User" node (the laptop owner), and so on. Graphs are immensely flexible: they can easily incorporate new entity types or relationships as needed (no rigid schema migration), and they naturally model the interconnected nature of security data (attacks are chains of events on various entities). We use an open-source **memory-first graph database (MGraph)** to manage these graphs, allowing us to manipulate and query the data efficiently in-memory, then serialize to JSON for storage [21]. MGraph was specifically designed for **GenAI and serverless** scenarios – it runs in-memory inside a function, and reads/writes its state as JSON to the file system (or S3) as needed [6] [22]. This gives us the speed of an in-memory DB with the persistence of a disk, all without running a dedicated database server. In essence, **we treat JSON files in cloud storage as our graph database** – a key enabler of serverless operations [4]. The graph model also makes it possible to easily visualize the data (e.g., generating a graph diagram of an attack sequence) and to apply

graph analytics (shortest paths, community detection for clusters of alerts, etc.) for deeper insights.

- **4. Minimal Data Transfer (Relevant Subsets Only):** A guiding principle is to **reduce data at each step** to only what is necessary for the current analysis goals. Traditional SIEMs often collect everything "just in case," resulting in huge data lakes where most data is never queried. In our approach, we start from a specific question or event and work outward, pulling in data **iteratively**. For example, if investigating a malware alert on a host, we might first grab the alert details and the host's identifier. Then we pull only that host's log entries around the time of the incident, and perhaps the specific file hash from a threat intel database. We don't ingest logs from unrelated systems or timeframes until they are shown to be needed. This scoped approach keeps each processing stage lightweight. Moreover, once data is converted into the semantic graph, we often compress or summarize it. The graph might store a log event in a normalized form (just key fields and a reference to the raw log for drill-down), rather than storing an entire verbose log line. By aggressively filtering and summarizing as we go, we **ensure that subsequent stages deal with a manageable amount of data** focused on the problem at hand. If a new question arises, we can always go back to sources for more (since we haven't thrown anything away at the source), but we avoid dragging the entire haystack through every stage of the pipeline.

- **5. GenAI in the Loop – with Controls:** Generative AI, especially large language models, are used as *assistants* in the pipeline where appropriate. For instance, an LLM can be tasked to parse an unstructured text log that doesn't have a clearly defined schema – turning a cryptic error message or stack trace into structured data (key=value pairs or a short summary). LLMs can also help identify connections ("this IP address from log A appears in log B" or "log lines suggest these events are part of the same incident"). However, a crucial principle is that **LLM usage should not compromise determinism or trust**. We tackle this by constraining LLM outputs to a specific format. Using techniques like OpenAI's function calling or JSON schema output, we ensure that when an LLM is invoked, it returns a structured object (e.g., a list of entities it extracted, or a suggested link between two nodes) [8]. These results are then validated or reviewed (sometimes a human-in-the-loop might verify a critical step, or code logic might double-check it). Over time, as patterns stabilize, the knowledge gained from LLM analyses can be codified into traditional code – for example, if the LLM repeatedly identifies that a certain log format can be parsed with regex X into fields, we can replace the LLM call with that deterministic parser. In this way, the pipeline **continuously learns and optimizes**, using AI to fill gaps but moving toward efficient algorithms as soon as we're confident. We also isolate LLM calls to specific tasks (never end-to-end incident analysis in one go), which eliminates the opaque "AI magic" problem and makes each AI-assisted step explainable. The end result is a smart assistant that accelerates data interpretation without turning the SIEM into a non-deterministic black box.

- **6. Full Evidence and Provenance Chain:** Ephemeral GenAI SIEM treats security investigations like scientific experiments: every fact or conclusion is backed by evidence, and hypotheses are tracked. The knowledge graph isn't just storing *what* happened, but also *how we know*. Each node in the graph (say, "Malware File XYZ was executed on Host ABC at 12:00") can carry an attribute pointing to the source data or analysis that confirmed it (e.g., a pointer to a specific log file line, or an EDR alert ID). Because our pipeline saves intermediate artifacts, an analyst or auditor can drill all the way down from a high-level incident report to the raw data that substantiates it. This provenance tracking builds trust: stakeholders can verify that conclusions are data-driven and not hallucinated or assumed. Additionally, we integrate **risk and ownership metadata** into the graph from the start. That means as soon as we identify an asset, we try to attach information like its classification (public vs. sensitive data), its business owner, and any known vulnerabilities

or criticality. As soon as we identify a user account, we link it to the person or role and their department. By doing this linkage early, when an incident graph is built, it is already enriched with business context – ready to answer "What is the impact?" and "Who needs to be involved?" *during* the investigation, not as a separate manual enrichment after the fact. This approach ensures that when an alert is raised to decision-makers, it is accompanied by actionable context (e.g., *"Malware X on host Y which is a finance server holding PII data, owned by John Doe, could lead to GDPR impact."*). Traditional SIEMs often leave this critical context as an exercise for the analyst; here we strive to provide it out-of-the-box.

- **7. Ephemeral, Disposable Infrastructure:** A cornerstone of this approach is that **nothing runs 24/7 except storage**. All computation happens in transient bursts. We might use cloud functions, ephemeral containers, or short-lived VMs to execute each pipeline stage or respond to each event. Once a task is done – e.g. logs have been processed into a graph and saved – the compute environment can be torn down completely. In an ideal state, you could *delete the entire SIEM processing stack and rebuild it from scratch* using the data and definitions stored in S3 and code repositories, and you'd get the same results. This makes the system highly resilient (infrastructure as code, easy to redeploy) and **inherently scalable**: if you suddenly have 100 incidents to analyze, you just invoke 100 parallel pipelines without worrying about provisioning and sizing a long-running cluster. It also dramatically reduces attack surface – with no always-on servers or databases (which in traditional SIEMs are juicy targets themselves), there's less to defend. This "here now, gone next minute" compute model follows the principle that *the best defense is not having anything to attack*. Even if an attacker gained access to the SIEM's processing environment, there's nothing persistent to exploit once the function finishes and memory is wiped. This principle was proven in practice by one of our earlier projects that exposed an S3-based API – it had virtually no servers to latch onto, improving security posture by design [23].

- **8. Augment Analysts, Don't Replace Them:** Importantly, Ephemeral GenAI SIEM is **not about removing humans from the loop** or automating away security engineers. On the contrary, it's designed to make security teams far more effective. By relieving analysts from grunt work (like manually gathering logs from 5 systems or cross-correlating timestamps), they can focus on interpretation and decision-making. The system's outputs – especially with GenAI summarization – can provide a starting narrative and highlight the key risk factors, but a human will validate and add judgment. Also, analysts can interact with the knowledge graph through queries or even natural language questions (with an LLM translating the question to a graph query), making the investigation feel like a conversation with your data. The aim is to achieve a **10x productivity boost** for security teams (a goal echoed by others in the industry [24]), enabling small teams to handle the growing scale of threats. Rather than replacing your SOC, this approach turbocharges it – the mundane is automated, the complex is made simpler, and the human expertise is amplified.

- **9. Evolutionary Development (Self-Improving System):** Finally, the architecture embraces continuous evolution. We often start deployment not by boiling the ocean (ingesting everything on day one), but by tackling one incident or use case end-to-end. For example, we might begin with building the pipeline for malware detection on endpoints. As we solve that, we accumulate building blocks – perhaps a parser for EDR alerts, a connector to the Active Directory for user info, a routine to fetch PCAP data from a sensor, etc. Next, we might take on a cloud security use case (e.g., unusual AWS API calls), which will introduce new data sources and require new graphs (like a cloud asset inventory). Because the system is modular, we can plug in these new components and also link them with existing ones (e.g., connecting an AWS alert with an on-prem user account if relevant). Over time, as more scenarios are handled, the knowledge graph grows in breadth, and the library of data connectors and transformation code grows. The SIEM

essentially *builds itself piece by piece* with each real incident – ensuring that development is always driven by real-world needs and delivers immediate value. This approach aligns with **Wardley Mapping and ILC (Innovate-Leverage-Commoditize) cycles**: first innovate with quick prototypes (often using LLMs or scripting to prove it out), then solidify the successful approaches by coding them more robustly (leveraging what we learned), and finally commoditize stable features into standard tools or microservices. The result is a living system that can adapt to the changing threat landscape and incorporate new technologies (like new AI models or data platforms) as they emerge, without a complete redesign.

These principles set the stage for a SIEM that is cloud-native, intelligent, and laser-focused on providing *actionable security insight* rather than a flood of raw data. Next, we dive into the architecture that realizes these principles, and we illustrate the workflow with a concrete example.

Architecture Overview

At a high level, the Ephemeral GenAI SIEM can be visualized as a pipeline that starts from **raw, distributed data sources** and ends with **consolidated security knowledge** (alerts, graphs, and reports), with several transformation phases in between. Key architectural components include:

1. Data Sources and Connectors: These are the various origins of security data, which can be anything from traditional log files and SIEM feeds to API endpoints, message queues, or even screenshots and emails. Examples include: - Endpoint telemetry (EDR logs, OS event logs on PCs and servers) - Network logs (firewall, IDS alerts, DNS logs, cloud VPC flow logs) - Application logs (auth events, error logs, transaction logs) - Cloud service logs (AWS CloudTrail, Azure Activity Logs, SaaS audit logs) - Threat intelligence feeds (IoCs, vulnerability databases) - Asset and identity inventories (CMDB, Active Directory, HR databases for user info) - **Unstructured data** relevant to an incident (an email body that delivered a phish, a malicious file sample, a chat transcript, etc.)

Each source might have a dedicated **connector function** that knows how to retrieve data from it (using APIs, database queries, or by reading files from storage). Connectors are typically small, stateless functions that *load raw data* for a given query or time range (the *Load* in LETS). For efficiency, connectors might pull data in parallel if multiple sources are needed.

2. Persistent Data Lake (Object Storage as Database): All data, once pulled, is stored in a cloud object storage (like S3) in a structured way. We treat this storage as our **source of truth database** [4]. For example, when fetching an hour of Windows Event Logs related to a host, the connector might save a file `raw/hostXYZ/2025-06-18T12-13_events.json` in the bucket. This persistent layer decouples data acquisition from processing – once data is in the lake, subsequent pipeline stages work off those saved files, not the live source. This also means if the same data is needed again (for another analysis or due to a pipeline change), we can reuse it without another external fetch. All data is stored in an **open format** (JSON, CSV, PCAP, etc.) to avoid vendor lock-in and ensure longevity. Organizing the storage by data source and time (and possibly by incident/case) makes it easier to manage and purge when appropriate.

3. LETS Processing Stages: As described, each processing stage takes input from the storage, does some computation, and writes output back. In practice, we implement stages as either serverless functions or containerized jobs orchestrated by an ephemeral framework (could be AWS Step Functions, a Kubernetes job runner, or a simple orchestrator script). The major stages in our SIEM pipeline might include: - **Parsing & Extraction:** Convert raw logs or data into structured form. For each raw file (say, a chunk of firewall logs), a parser stage produces a normalized JSON or graph fragment. This is where

LLMs might assist if the format is tricky. The output is typically a list of event objects or initial graph nodes. - **Entity Graph Assembly:** Take the parsed data and construct the **semantic graph** nodes and edges. For example, if a log line says "User U downloaded file F from IP I," we create nodes for `User:U`, `File:F`, `IP:I` and edges like `User U -> downloaded -> File F` and `IP I -> hosted -> File F`. Each event thus becomes a subgraph linking the entities involved. This stage uses the MGraph library in memory to build the graph, then saves the graph (or the diff) as a JSON file (e.g., in `graph/hostXYZ/2025-06-18T12-13_events.graph.json`). - **Correlation & Merging:** This stage looks at all the graph pieces from various sources and tries to connect the dots. For instance, the EDR alert graph says "malware hash X on host Y", and a firewall log graph shows "host Y communicated with IP Z", and an email log graph shows "user of host Y clicked link from IP Z". The correlator will merge these into a single graph, identifying common entities (host Y, IP Z) and linking sequences of events into an attack storyline. This can be rule-driven (join by common fields) and/or LLM-assisted (ask the LLM if two pieces might be related, based on their data) depending on the complexity. The output is a **unified incident graph** representing the combined knowledge for this incident. - **Enrichment & Contextualization:** Here we fold in external context: attach asset info (what is host Y? production server? who owns it?), attach threat intel (is hash X known malware? is IP Z blacklisted?), attach controls mapping (which security control should have caught this, if any?). This may involve queries to other databases or simply linking nodes to reference graphs (e.g., a vulnerability database graph or MITRE ATT&CK knowledge base graph). After enrichment, the incident graph is not only a technical view of events, but a context-rich representation of the incident in business and risk terms. - **Analytics & Detection Logic:** Although in our scenario we started from a known alert (malware detected), the system can also proactively detect patterns in the graph. This stage can run analytical queries on the graph to find suspicious patterns (e.g., lateral movement paths, multi-stage attacks, privilege escalation indicators) or apply machine learning on the structured data. Any findings (say, "These three seemingly low-level events actually form a MITRE ATT&CK chain of Initial Access -> Execution -> Persistence") can be added as new nodes/alerts in the graph. This is how the system can act as a true detection engine, not just an incident assembler. Because we're now operating on a highly curated dataset (the incident-specific graph), such analytics are efficient and precise, in contrast to generic SIEM rules that run on a firehose of logs. - **Output & Reporting:** The final stage takes the enriched, analyzed incident graph and produces human-friendly outputs. This could include: - An interactive graph visualization (nodes and links) accessible via a web UI. - A timeline view of events. - Natural language summaries generated by an LLM (prompted with the graph data) – e.g. an executive summary of what happened and its impact. - Automatic report documents (in Markdown/PDF) for compliance or post-incident review, with all evidence attached. - Integration back into ticketing or SOAR systems (e.g. create a Jira ticket with key details, or trigger response playbooks). The key is that the **heavy lifting of compiling the story and evidence is done by the system**, so the security team receives a package of actionable intelligence rather than raw logs.

All these stages are orchestrated in a **serverless fashion**. For instance, an initial trigger could be an alert from an EDR system hitting a webhook; that launches a coordinator function which kicks off the necessary connectors and stages for that host and alert. Or an analyst could manually initiate an investigation via a web interface by selecting an alert of interest, which triggers the pipeline. Under the hood, something (could be as simple as a Python script using AWS SDK, or a Step Functions state machine) coordinates the sequence: fetch data, wait for storage, parse, assemble graph, etc. Because each step writes to storage and can be stateless, the pipeline can tolerate variations – e.g., if one data source is slow, it doesn't derail the whole process; the rest just waits for that file to appear.

Use of Open-Source and Standard Tools: Notably, this architecture leans heavily on open technologies. The graph database (MGraph-AI) is open source [25], the data formats are open (JSON, Markdown), and the orchestration can be built with standard cloud services or workflow engines. This avoids lock-in to any particular SIEM vendor. The philosophy is that **the knowledge (graphs, data) is**

the valuable output, and it lives in portable form, while the processing logic can be adapted or reimplemented as needed. Organizations can extend or customize the pipeline by adding their own stages or swapping components (for example, using a different LLM or an internal ML model at the enrichment stage) without breaking the overall framework.

Now, to ground this in reality, let's walk through an example incident to see how all these pieces come together.

Workflow Example: Investigating a Malware Incident

Consider a scenario faced by a SOC in 2025: an alert comes in that **malware was detected on an employee's laptop**. This could originate from an endpoint protection platform (EPP/EDR) like Microsoft Defender or CrowdStrike, which flags a malicious file execution. In a traditional setup, this alert would show up in the SIEM with perhaps a short description and some hashes or file paths. It would be up to an analyst to investigate further by querying various logs. In our Ephemeral GenAI SIEM, however, the response is largely automated and far more comprehensive. Here's how it unfolds step by step:

Step 1: Alert Ingestion and Initial Graph Node Creation
The moment the malware detection alert is generated by the endpoint, it triggers our pipeline. This could be via a webhook from the EDR to our cloud function or a scheduled poll of the EDR's alerts API. The first thing we do is create a **graph node representing the alert event** in our knowledge graph. This node might have attributes like `alert_id`, `malware_name` (if known), `file_hash`, `host_name`, `timestamp`, and `EDR_tool` (source). We save this initial evidence in our storage (e.g., `graph/cases/incident123/alert.json`). This node is essentially a placeholder that says *"We have an alert about malware X on host Y at time T."* It's the starting point of our investigation graph.

Step 2: Identifying Relevant Data Sources
Next, the system determines which data sources might hold relevant information for this alert. Based on the alert details, it knows: - The host involved (say, `HOST123`). - Possibly the file hash or name (`malware.exe` with hash `abc123`). - The timestamp of detection. - The user logged into that host (if provided by EDR).

Using this, the pipeline compiles a list of sources to query, for example: - **EDR Detailed Logs:** Pull the detailed telemetry from the endpoint agent around that time (e.g., what process executed, its parent process, any registry or network activity captured). - **Host OS Logs:** Query the Windows Event Logs or sysmon on that host for events in a time window (perhaps 15 minutes before and after the alert) – this might show how the malware got there (e.g., a PowerShell execution or a user login event). - **Network Logs:** Check the enterprise firewall or proxy logs for any connections from that host, especially around the time of infection (did it download something from an external URL? Connect to a command-and-control server?). - **Email Gateway:** If there's a suspicion the malware arrived via email (perhaps the file name or user behavior suggests it), search the email system for any messages to the user with attachments or links around that time. - **Threat Intelligence:** Look up the file hash `abc123` in threat intel databases (VirusTotal, internal TI platform) for any known info (is it known ransomware? what domains or IPs are associated with it?). - **Vulnerability Scans:** Retrieve any recent vulnerability scan report for `HOST123` to see if it had missing patches that could be relevant (maybe the malware exploited a known vuln). - **Asset Info:** Fetch from CMDB what `HOST123` is (laptop vs server, OS version) and who it is assigned to (owner's name, department). - **Identity Logs:** Pull recent authentication logs for the user of `HOST123` – did that user's account do anything suspicious (VPN logins, failed logins, etc.) around that time?

Not all of these will always be needed, but the system makes an informed guess of what could be useful. Importantly, it doesn't yet **collect everything everywhere**; it focuses on this host, this user, and this time slice.

Step 3: Data Retrieval (Load Phase)
For each identified source, a connector function is invoked. These run in parallel where possible. They use the source's native interface to get data: - The EDR connector calls the EDR API with the alert ID to get the full event details and any related telemetry (perhaps the EDR provides a JSON report of the incident on that host). - The Windows logs connector queries the logging infrastructure (or pulls from a log archive) for events from `HOST123` in the given timeframe, then filters to those relevant (Security log, Sysmon, etc.). - The proxy log connector searches proxy records for `HOST123` or the user's machine IP on that date. - The email connector looks up the user's mailbox or the email gateway for potential phishing emails. - The threat intel connector queries the hash and returns any matches. - Etc.

As each connector gathers data, it writes the raw results to our S3 storage under a unique path for this incident. For example:

```
case123/raw/EDR_HOST123.json
case123/raw/WinLogs_HOST123_0600-0630.log
case123/raw/Proxy_Host123_0600-0630.csv
case123/raw/VirusTotal_abc123.json
...
```

Some data might be large (e.g., logs), but we're only pulling a time-bounded slice. If a source is unavailable or returns nothing, that is noted (and can be flagged as a gap for later).

Step 4: Extraction and Graph Construction (Extract & Transform Phases)
Now parsing and interpreting begins. For each raw data file: - The EDR JSON is parsed to extract key entities: the malware file (with its hash and name), the process that executed it, the user account on the machine, any registry keys or persistence mechanisms noted, etc. Each of these becomes a node or edge in the graph. For instance, we add a `File` node for the malware (tagged with hash, name, size), a `Process` node (with process ID, name), and edges like `Process -> created File` or `User -> executed File`, depending on what the EDR data says. - The Windows event log file is parsed (likely with a predetermined schema, possibly using a library or even an LLM if needed for unusual event formats). We extract events like logon attempts, file write events, or script execution events. These become nodes (e.g., an event node or directly mapped to higher-level nodes: an event "User login from X" might link the `User` and `Host` nodes with a "logged in from IP" edge). - The proxy logs are parsed for any HTTP requests by that host. Suppose we find that at 05:55 UTC, `HOST123` (or its IP) downloaded a file from `evilmalware.com/dropper.exe`. We create a `Domain` node (`evilmalware.com`), a `NetworkRequest` node or edge from Host to Domain, and perhaps a `File` node for `dropper.exe` if we have details. Notably, this directly might explain how the malware got in (downloaded from that domain). - The email search might find an email to the user at 05:50 with a link to that same domain. If so, we create an `Email` node (with sender, subject) and link it: `Email -> delivered URL -> Domain` and `User -> received -> Email`. - The threat intel for the hash might tell us the malware is "AgentX" malware family and that it contacts domain `evilmalware.com`. We enrich the `File` node with this info (malware family = AgentX) and link the `Domain` node to an `ThreatIntel` node or mark it as malicious. - The vulnerability scan might show Host was missing a certain patch; if the malware exploited that, it could be relevant. We could add a `Vulnerability` node and link `Host -> has vulnerability`.

At this point, we have constructed a **knowledge graph that sprawls across multiple data sources** but is centered on our incident. It might look like:

```
User[Bob] --owns--> Host[HOST123] --hasProcess--> Process[malware.exe] --
fileHash--> File[abc123]
User[Bob] --received--> Email[ID456] --link--> Domain[evilmalware.com]
Host[HOST123] --made HTTP request--> Domain[evilmalware.com]
Domain[evilmalware.com] --hosted file--> File[abc123]
File[abc123] --is type--> MalwareFamily[AgentX]
File[abc123] --detectedBy--> Alert[EDR-123] (the original alert node)
Host[HOST123] --hasVulnerability--> Vulnerability[CVE-2024-XXXX]
...
```

Every node/edge also carries metadata like timestamp (for events), source reference (which raw log or record supports it [26]), etc. This graph is saved as `case123/graph/merged_graph.json` for further use.

Step 5: Correlation and Hypothesis Linking
With all pieces in the graph, the system now correlates the sequence of events. Often this simply emerges from the graph connections: we can see an email led to a download which led to execution which led to an alert. However, there might be missing links or multiple possibilities. For example, if we didn't find the email, we might have the download but not how it was initiated. In such cases, we could query other sources (maybe the user visited that URL via web browser rather than email – we might then check browser history logs or DNS logs). The pipeline can use an LLM here to identify any apparent gaps or suggest additional pivots: e.g., *"We saw a malicious file download, but no email – maybe the user navigated there manually or via a website. Check web proxy logs further back or DNS logs."* This might prompt an additional data fetch (going back to Step 3 for another round). The iterative nature means the investigation can deepen as needed, akin to a human analyst following intuition, but guided by the AI to not overlook paths.

Assuming we have a fairly complete graph, the system can then piece together a narrative. It identifies *chains of events*. In our example, one chain might be: `Email -> User click -> Domain -> File download -> Malware execution -> EDR detection`. This can be mapped to the MITRE ATT&CK framework stages: Recon (email phishing) -> Initial Access (user clicks link) -> Execution (malware runs) -> C2 (if the malware connected out, not given here but if it did) -> Detection (response). The system can tag each part of the graph with MITRE technique IDs (if known) for standardized terminology.

Step 6: Enrichment of Risk and Impact
Using the graph, we now enrich with business context (some of this we started in Step 4 with asset info, but here we focus on impact analysis): - What data could have been at risk? If `HOST123` is a laptop, perhaps it has access to certain databases or contains certain sensitive files. If our system has a link to data classification (say we know this user handles finance data), we note potential data compromised. - Who needs to know? We see the host owner is Bob in Finance. So likely the Finance IT partner or CISO needs to be informed if this is serious. We tag stakeholders. - We also calculate a severity or risk score for the incident: e.g., because malware executed and connected out, and the host had vulnerabilities, this is high risk (potential breach). Or if EDR stopped it and nothing further happened, it's moderate. - If the graph indicates any security control failure (say the email was phishing that bypassed filters, or the user had local admin rights allowing execution), we note those as *lessons learned* links in the graph (e.g., a node "SecurityControl=Email Filter" with an edge "failed to detect -> EmailID456").

Step 7: Reporting and Action
Finally, the system generates outputs: - **Graphical View:** An interactive graph or a visual diagram is produced showing the devices, users, and events with arrows connecting them (timeline annotated). This helps the SOC or incident responders to validate the sequence and see if anything is missing. - **Executive Summary:** A natural-language summary is crafted (using an LLM prompt filled with key graph details) describing what happened: e.g., *"On June 18, 2025, a targeted phishing email sent to Bob (Finance) led to the download of malware AgentX on his laptop HOST123. The malware executed but was caught by endpoint defenses. Analysis shows the malware attempted to connect to evilmalware.com, which is a known bad domain. No data exfiltration was observed. Bob's machine will be re-imaged and credentials reset as a precaution."* This summary is tailored for a leadership audience, focusing on impact and resolution. - **Technical Report:** A more detailed report is compiled for the technical team. This could be a Markdown document that enumerates the indicators (file hashes, domains), lists all related events (with timestamps and source logs references), and maps the incident to MITRE ATT&CK tactics. Because our pipeline retained all evidence, we can attach or inline the actual log snippets that correspond to each event for transparency. - **Tickets/Alerts:** If certain actions are needed (e.g., block the domain on firewall, reset user password, patch the vulnerability on that host), the system can automatically create tickets in the ITSM system or send alerts to the responsible teams, complete with context of why that action is needed. - **Learning Feedback:** The incident's data could be fed into detection rules for the future (maybe writing a new rule that if an email links to that domain, alert immediately, or updating SIEM correlation with this pattern). In our architecture, we might store this pattern as a reusable **story graph** that can be searched for in future data (like a template of how AgentX attacks look).

All the results are stored and indexed in the research hub (which could be the knowledge graph repository for all incidents). If a similar incident happens next month, the CISO can even query: *"Have we seen evilmalware.com before?"* and the system will find that domain node from this incident graph and show the past context. This historical memory, built on graphs, becomes incredibly valuable for threat hunting and retrospectives.

Step 8: Disposal and Reset
Once the analysis is done and outputs are delivered, the ephemeral compute infrastructure used can be completely torn down. The data (graphs, raw logs) remains in the storage for future reference, but the analysis environment (containers, memory structures) is gone. If we need to revisit or redo analysis, we simply re-run the pipeline on the stored data or fetch new data. The system is ready to handle the next incident with a fresh set of functions, ensuring no cross-contamination of data in memory and optimal use of resources.

This example demonstrates how Ephemeral GenAI SIEM handles an incident in a way that contrasts sharply with a traditional SIEM workflow. Instead of an analyst manually pulling logs and piecing together clues for hours, the system assembled a comprehensive narrative in minutes, with full evidence attached. The analyst's role shifts to validation and response decision-making, supported by the system's findings. Crucially, this was done **without pre-ingesting all corporate logs into a single system**, and without maintaining a standing army of servers crunching data 24/7. The pipeline only spun up in response to the event, used targeted data to enrich context, and stood down after delivering insight.

Technology Stack and Implementation

To implement the Ephemeral GenAI SIEM, we leverage a combination of cloud services, open-source libraries, and custom code – much of which is developed in the open as part of Dinis Cruz's research initiatives. Below are key components of the tech stack and how they contribute to the solution:

- **Cloud Object Storage (Data Lake):** A service like Amazon S3, Azure Blob Storage, or Google Cloud Storage serves as the backbone for storing all inputs and outputs. This is the system's long-term memory. We organize the bucket into logical folders (by incident or date) and use file naming conventions for versioning. Because this storage is practically infinite and cheap, we don't worry about running out of space – we can keep raw evidence for compliance needs without expensive hot storage costs. And since the storage is separate from compute, it naturally enables the ephemeral compute model (any stateless function can access the data when needed, without local state).

- **Serverless Compute & Orchestration:** Wherever possible, we use serverless functions (AWS Lambda, Azure Functions, GCP Cloud Functions) for connectors and lightweight processing tasks. For heavier or longer tasks (like parsing a large log or running an ML model), we use containerized jobs on a service like AWS Fargate or Azure Container Instances, or Kubernetes pods in an on-demand cluster. The orchestration can be handled by a workflow engine (e.g., AWS Step Functions or Azure Logic Apps) which can sequence tasks and handle retries, or by a custom lightweight orchestrator (even a Python script or a directed acyclic graph defined in something like Apache Airflow, though managed solutions are preferred to keep with the serverless ethos). The use of managed cloud functions means we **automatically scale** – if ten incidents come in at once, ten concurrent pipelines run without us pre-provisioning servers. And if none occur for days, we pay nothing during that idle time.

- **MGraph-DB (MGraph-AI):** This is the in-memory graph database library we use to handle graph operations. It's written in Python and designed to integrate well with serverless environments [6] [27]. We embed this library in our functions or containers that need to build or query graphs. MGraph provides a Pythonic API to create nodes and edges, run filters, and then serialize the graph to JSON. It's highly optimized for our use case – type-safe, minimal dependencies, and with JSON as the primary persistence format [28] [29]. This means we can easily take the output of one LLM call, stick it into MGraph structures, merge with another, and save – all within, say, a single AWS Lambda execution. By using MGraph, we avoid the need for an external graph database server (like Neo4j or JanusGraph) which would break our serverless model and add admin overhead.

- **Generative AI Models:** We incorporate LLMs at specific junctures. These could be via APIs (OpenAI GPT-4, Azure OpenAI service, etc.) or open-source models we host ourselves if data sensitivity is a concern. Key usage patterns include:

- **Parsing Unstructured Data:** For example, feeding an raw log line or an error message to an LLM prompt like "Extract the following fields if present: timestamp, source IP, destination URL, user agent, error code..." and have it output JSON. Because we supply a schema and examples, the LLM's response can be made very consistent.
- **Semantic Mapping:** If we have two sets of entities (like processes on a host and threat intel about malware tactics), we might ask the LLM to find relationships – e.g., "Given these process names and this list of known malware traits, which processes seem related to known malware

behavior?" – returning any matches or suspicions which we then turn into graph links (with a confidence score).
- **Summary Generation:** Using an LLM to turn the graph data into English (for reports or explanations).
- **Question Answering:** Enabling an analyst to ask follow-up questions in natural language. For instance, "Was any sensitive data accessed by this malware?" The system can translate that into a graph query (looking for any access to certain databases or files in the graph) and respond with an answer or further analysis.

We ensure these calls are **stateless and idempotent** parts of the pipeline. Each LLM invocation is treated as a function: input data goes in, output data comes out. We do not rely on hidden model state or memory between calls. And if needed, we can log the prompts and responses for audit (especially important if using external APIs). If an LLM step is non-deterministic (slight variance in phrasing, for example), it doesn't affect the integrity of the graph data since we enforce structure. In essence, we treat LLMs as powerful parsers and correlators under tight control.

- **No-Code / Low-Code Interfaces:** To truly empower rapid development, we incorporate no-code principles. This can manifest in a few ways:
- **Prompt-based Pipeline Configuration:** We can describe a new data source or transformation in English (or a simple DSL), and use an LLM (in a development context) to generate the code needed. For example, an engineer might write a prompt: "Connect to the HR system API and retrieve the employee record for a given username, then output name, department, manager" – the LLM can produce a connector code snippet in Python which can then be reviewed and deployed. This significantly accelerates integration of new sources.
- **Graphical Workflow Orchestration:** Utilizing tools where possible that let us visually define the pipeline (some cloud providers offer drag-drop workflow designers). Even if under the hood it's code, it provides a clear picture for architects and allows adjustments without digging into code.
- **Interactive Notebooks and Testing:** We maintain a library of Jupyter notebooks for developing and debugging pipeline steps. Analysts can use these notebooks to do ad-hoc analysis on the data lake by loading the graphs and querying them using familiar tools. This is a form of low-code interaction with the system, making it feel more like using an analysis tool than programming.

The bottom line is that adding a new correlation rule or data source doesn't require writing thousands of lines of Java and deploying a heavy app (as might be with older SIEM customizations). It can be as simple as writing a prompt or a few lines of Python, which our GenAI helpers and frameworks then integrate into the larger system. This encourages experimentation and quick iteration – essential for keeping up with evolving threats.

- **Visualization and UI:** While a full SIEM product would have a polished UI, our architecture allows for flexible front-ends. We can use open-source tools like Grafana or Kibana for dashboards, since our data is in JSON and can be indexed. For graph visualizations, we have used Graphviz to generate diagrams or libraries like D3.js for interactive web views. Even just rendering the connections as a network chart in a web app can give analysts a powerful exploratory interface. The UI layer is kept thin and separate – it talks to the storage or an API to retrieve incident data and displays it, without containing heavy logic. This means the solution can be integrated into existing portals or wiki pages (e.g., an incident wiki page that embeds the graph image and links to evidence files).

- **Security and Access Control:** Given the sensitive data involved, strong access control is implemented at the storage and function levels. Data in the lake is encrypted and segmented by case. Only the functions with need-to-know can read certain paths (for example, a connector that

writes raw data might have rights to write in `raw/` but not to read everything). The output reports can be pushed to a secure SOAR or case management system that already has role-based access for the SOC. Additionally, because we minimize long-lived components, the attack surface is small – but we still secure the pipeline (e.g., ensuring the LLM calls do not leak data to external logs, using self-hosted models for highly sensitive orgs, and auditing all access to the storage).

In practice, implementing this stack has been greatly accelerated by open-source contributions. For instance, the MGraph-AI library was released as open source [30], meaning anyone can use or contribute to it for their own GenAI+graph projects. The LETS pipeline concept was documented and shared [5], providing a template that others have begun to follow for similar deterministic AI workflows. This white paper itself, co-authored by Dinis Cruz and ChatGPT, demonstrates the no-code philosophy: using AI to articulate and refine the architecture.

Benefits and Future Outlook

Adopting the Ephemeral GenAI SIEM architecture yields numerous benefits for organizations striving to improve their security operations:

- **Significant Cost Reduction:** By eliminating the need to ingest and index all security data into a single system, organizations can avoid the steep licensing and storage costs of traditional SIEMs. Expensive “hot” storage in SIEM is replaced by cheap object storage. Compute costs are incurred only when needed, and even then, they are granular (you pay for a few Lambda invocations and container runs, not for an entire 24/7 server cluster). Edge Delta’s analysis suggests that decentralized processing can lower observability costs by up to 95% [31], and our approach follows a similar ethos of processing data at the source and storing intelligently. Teams no longer have to make painful choices about which logs to keep – they can retain everything in raw form without breaking the bank, knowing that the system will intelligently access what’s needed.

- **Improved Detection and Faster Response:** By having **full context on demand**, analysts can get to root cause and impact much faster. The system automates the grunt work of correlation, so the mean time to understand an incident plummets. Instead of chasing down information across systems, responders get a coherent story and can focus on response actions. The inclusion of AI-driven analysis means subtle connections (that might be missed by rigid rules) can be uncovered – for example, the system might flag that two seemingly unrelated alerts are connected by a common domain or technique, thereby detecting multi-stage attacks that evade single-point detections. Overall, security monitoring moves from a reactive, after-the-fact posture to a more proactive and informed stance, as the system can continuously integrate new intelligence and even simulate "what-if" on past data (since all data remains accessible).

- **Less Noise, More Signal:** The combination of focused data gathering and AI-driven context means alerts generated by this system can be of much higher fidelity. We’re effectively pre-triaging alerts by enriching them immediately. An alert that reaches a human now comes packaged with its context – reducing the chance it’s a false positive or irrelevant. By incorporating risk and business context, the system can also prioritize incidents better. For example, a malware on a test machine vs. on a finance server can be automatically flagged with different severity. This addresses the notorious **alert fatigue** problem [13] – analysts spend time on what matters, as trivial or spurious events are filtered out or contextualized as low priority.

- **Transparency and Trust:** Thanks to the LETS architecture, everything the system does is traceable. This is vital in security where decisions (like ignoring an alert or taking a costly response action) need justification. With our approach, any alert or non-alert decision can be backed by the saved evidence and processing steps. The security team and management can gain trust in the AI components because they can always inspect why something was flagged or not. This is in stark contrast to some AI-based security products that act as mysterious black boxes. Furthermore, having all intermediate data and final outputs in accessible files means during audits or incident post-mortems, the team can share detailed records with regulators or internal audit, demonstrating diligence and thoroughness.

- **Flexibility and Customization:** Every organization's environment is unique, and threats evolve. The open, modular nature of Ephemeral GenAI SIEM means it can be tailored to specific needs. New data sources (say, a custom application log) can be added by writing a small connector and parser – without waiting for a vendor's next release. Detection logic can be adjusted on the fly; if a certain false positive keeps occurring, the team can tweak a stage to filter it out or add a condition, and because the pipeline is code and files, version control and testing of that change are straightforward. You are not locked into the capabilities of a vendor's engine; you essentially have the framework to build exactly what you need, on top of open infrastructure. This also future-proofs the organization – as new technologies like better LLMs or new data analytics frameworks appear, they can be integrated into the pipeline at the appropriate stage.

- **Scalability:** The serverless, on-demand nature of this architecture inherently supports scaling to very large environments or spike events. If there's a widespread incident (e.g., a new worm affecting hundreds of hosts), the system can spawn parallel analysis for each affected host without running out of capacity (subject to cloud limits, which are generally high). Traditional SIEMs often fall over in such scenarios due to query load or simply generate an unmanageable number of alerts. In our design, **more data or more incidents simply mean more parallel functions**, and the cloud handles that scaling. This elastic capability means the SIEM can handle everything from routine daily alerts to crisis situations with equal efficacy.

- **Alignment with Modern IT and SecOps:** As organizations adopt DevSecOps, cloud-native deployments, and agile methodologies, they need security tools that fit into that paradigm. Ephemeral GenAI SIEM is inherently aligned with these modern practices – it's cloud-native, defined as code, and can be integrated into CI/CD (for example, one could automatically run certain security queries or graph analyses whenever a new application version is deployed, as a form of continuous assurance). It also pairs well with the "data mesh" concept, where data remains distributed and query comes to the data. In essence, it transforms the SIEM from a monolithic silo into a *mesh of security data and logic* woven throughout the environment, which is more suited to the microservice and multi-cloud world.

Future Outlook: This architecture opens the door to numerous future advancements. For instance: - **Real-Time Stream Processing:** While our described approach is trigger-based and batch on demand, the same principles can be applied to streaming data. One could set up continuous log watchers that perform lightweight edge processing (filtering and summarizing streams locally, only sending high-value signals to a central graph). This would combine the best of both worlds: low-latency detection with central graph correlation. - **Automated Response Integration:** With such rich context, automated playbooks can be more precise. For example, if an incident graph clearly shows a compromised user account, an automation could disable that account within minutes of the detection, because the system has high confidence and evidence of malicious activity. Essentially, the graph can feed a SOAR platform with machine-readable context to trigger targeted response actions. - **Community and Open Sharing:** If many organizations adopt a graph-based SIEM approach, they could share anonymized graph

patterns of attacks. Imagine a shared repository of attack graphs (similar to how malware signatures are shared) – when one org experiences a novel attack and graphs it, others could use that to scan their own environment for similar patterns. This collaborative detection model could greatly enhance industry-wide defenses. The use of common ontologies and open schemas in our approach makes such sharing feasible (no proprietary log formats, just graphs of TTPs that everyone can understand). - **AI-augmented Threat Hunting:** Analysts could leverage the system to ask high-level questions and have the AI do the heavy lifting. For instance, *"Show me any abnormal admin logins in the past week correlated with large data downloads"* – the system could comb through the knowledge graphs of the week and present findings. This flips the traditional hunting (where humans write queries and parse outputs) to a more intuitive AI-assisted exploration. - **Beyond Security – IT Operations and Risk Management:** The same platform could be extended to IT ops and compliance use cases. Since it's essentially a general graph of IT events and entities, one could detect performance anomalies, or verify compliance controls, by adding appropriate analysis stages. This could blur the line between SIEM, AIOps, and GRC tools, resulting in a unified knowledge-driven oversight of the digital environment.

Conclusion

The Ephemeral GenAI SIEM represents a bold re-imagining of security monitoring. By harnessing the power of cloud scalability, the expressiveness of knowledge graphs, and the intelligence of generative AI, it addresses the long-standing pain points of traditional SIEMs: high cost, overwhelming noise, and lack of context. Our design choices – from using S3 as a database, to treating every step as a reproducible data transformation, to keeping the entire system disposable – were driven by practical experiences in building AI-assisted systems that needed transparency and reliability [5] [6].

In implementing this system, we've adhered to a philosophy of openness (open formats, open source tools) and collaboration between human experts and AI. The result is a SIEM that **scales by doing less upfront** (no more ingest-all-the-things), and **accomplishes more by thinking smarter** – connecting dots that were previously siloed and presenting solutions, not just alerts.

For technical CISOs and security leaders, this approach offers a path to finally escape the trade-off between comprehensive security visibility and operational feasibility. You no longer have to accept that "we only use 10% of our data" or that "the SOC is drowning in false alerts." With Ephemeral GenAI SIEM, you can truly use *all* your data in a targeted way, cut through the noise with AI-driven context, and do so on infrastructure that flexibly scales with your needs. It transforms the SIEM from a static log aggregator into a living, breathing analytical assistant that grows in knowledge every day.

In publishing this white paper, our hope is to share a blueprint that others can adapt and build upon. All the core components discussed are available through open research and code (for example, the MGraph-DB library and example pipelines [6] [32]). We encourage the community to experiment with these ideas, contribute improvements, and collectively push the state of the art in security operations. The challenges we face in cybersecurity are immense and ever-changing, but with approaches like this – combining the best of human strategy and AI capability – we have a fighting chance to stay ahead.

Dinis Cruz and ChatGPT Deep Research, June 2025

Sources: The concepts and implementations described here are based on Dinis Cruz's open research on deterministic GenAI pipelines and semantic graphs [5] [6], industry analyses of SIEM limitations [1] [2], and the practical lessons learned from building GenAI-powered security and news analysis platforms [33] [34].

---

1 3 9 10 11 13 14 15 19 Why Traditional SIEM Isn’t Enough—Peris.ai Brings Real Intelligence
https://peris.ai/post/why-traditional-siem-isnt-enough--peris-ai-brings-real-intelligence

2 12 16 17 18 20 26 31 Overcoming the Limitations of Centralized Monitoring | Edge Delta
https://edgedelta.com/company/blog/overcoming-limitations-centralized-monitoring

4 5 32 34 Introducing LETS: A Deterministic Data Pipeline Architecture | Dinis Cruz posted on the topic | LinkedIn
https://www.linkedin.com/posts/diniscruz_lets-load-extract-transform-save-activity-7333166696649621505-ZjqA

6 7 21 22 25 27 28 29 30 Introducing: MGraph-AI - A Memory-First Graph Database for GenAI and Serverless Apps
https://www.linkedin.com/pulse/introducing-mgraph-ai-memory-first-graph-database-genai-dinis-cruz-wxmde/?ref=mvp.myfeeds.ai

8 33 Building Semantic Knowledge Graphs with LLMs: Inside MyFeeds.ai's Multi-Phase Architecture
https://www.linkedin.com/pulse/building-semantic-knowledge-graphs-llms-inside-myfeedsais-dinis-cruz-jub5e

23 Creating a Secure GenAI News Feed: When the Best Defence is Not ...
https://www.linkedin.com/pulse/creating-secure-genai-news-feed-when-best-defence-having-dinis-cruz-5iqre

24 GenAI bots that make security teams 10x more productive | Dinis Cruz
https://www.youtube.com/watch?v=PoFoh89OBNs