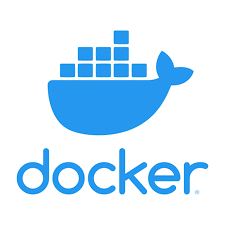
Demo Project – Not Finalized

THIS PROJECT WAS DONE AS PART OF THE EDUVOS GD & IS SHOWCASE



By: Salman Sadiq & Vaidant Makan

The End of SOC Alert Fatigue

AI-ASSISTED

SECURITY OPERATIONS CENTER (SOC) WORKFLOW

**SENTRY**

Table of Contents

[1. Project Overview 2](#_Toc215056371)

[1.1 Problem Statement 2](#_Toc215056372)

[1.2 Solution: The Sentry Workflow 3](#_Toc215056373)

[2. Technical Architecture and Workflow 4](#_Toc215056374)

[2.1 Technology Stack 4](#_Toc215056375)

[2.2 Sentry Data Flow (The 'Loop') 5](#_Toc215056376)

[2.3 LLM Selection and RAG Justification 6](#_Toc215056377)

[3. Implementation Details 7](#_Toc215056378)

[3.1 Virtualized Environment (Hyper-V) 7](#_Toc215056379)

[3.2 WindowsHost (Central Processing and AI) 8](#_Toc215056380)

[4. Demonstrable Vulnerabilities for Testing 9](#_Toc215056381)

**SENTRY: AI-ASSISTED SECURITY OPERATIONS CENTER (SOC) WORKFLOW DOCUMENTATION**

## 1. Project Overview

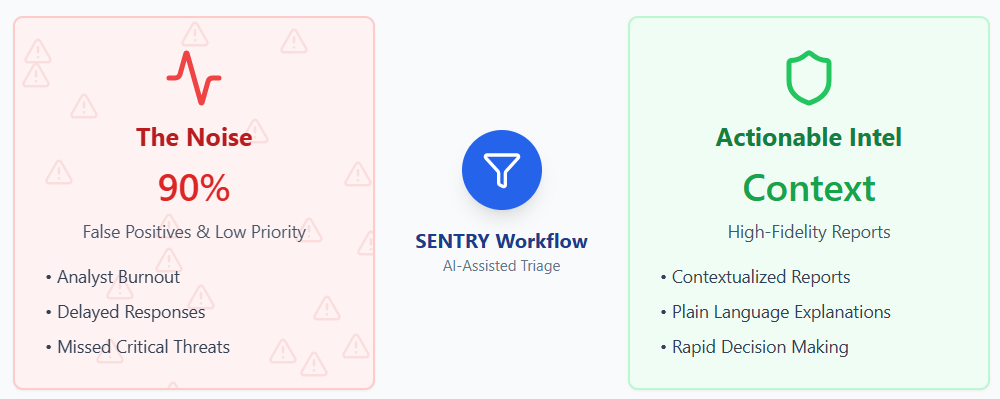
**Project Name:** Sentry — Security Operations Centre Digital Watchtower

**Purpose:** To develop and demonstrate an AI-assisted security workflow designed to combat alert fatigue, significantly improve SOC response times, and deliver highly contextualized, actionable security intelligence.

### 1.1 Problem Statement

Modern SOC analysts are constantly overwhelmed by the sheer volume of security alerts, with an estimated **90% being false positives or low-priority noise**. This operational inefficiency leads directly to:

* **Analyst Burnout and Fatigue:** High mental load and repetitive tasks degrade analyst performance.
* **Delayed Response:** Critical threats are often missed or responded to slowly due to a noisy environment.
* **Ineffective Communication:** Raw security logs lack the context required for clear communication between technical teams and non-technical leadership.
* **Tool Complexity:** Current security tools often fail to provide prioritized, contextualized, or plain-language explanations of complex events.



### 1.2 Solution: The Sentry Workflow

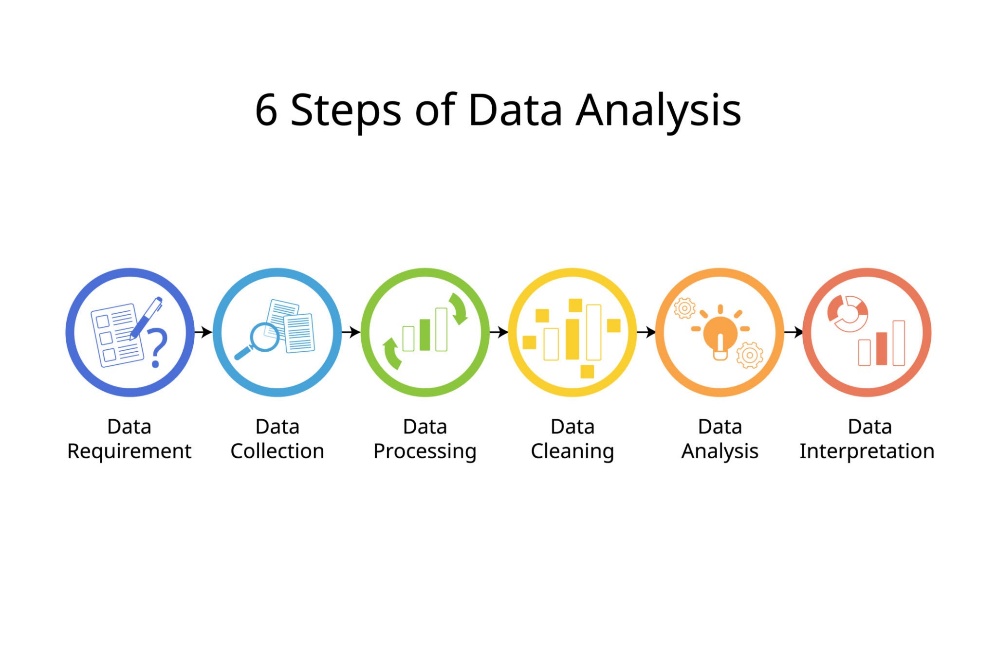
**Sentry** is a local-first, AI-assisted workflow that automates the initial triage and contextualization of intrusion detection system (IDS) alerts.

**Key Features:**

* **Ingestion:** Processes high-volume, enterprise-grade JSON alerts from the **Suricata** IDS.
* **Automation & Enrichment (n8n):** Alerts are normalized, grouped into incidents, and enriched with external data (GeoIP, WHOIS, ASN, etc.).
* **Local LLM Integration:** The enriched data is fed into a lightweight, on-premises Large Language Model (LLM) via **Ollama**.
* **Retrieval-Augmented Generation (RAG):** The LLM uses RAG against an ingested security knowledge base (e.g., NIST NVD data) to ensure accurate, up-to-date analysis.
* **Actionable Reporting:** Generates clear, prioritized incident reports tailored for both technical remediation teams and executive leadership.

**Value Proposition:**

* **Reduces Noise:** Groups hundreds of similar events (e.g., brute force attempts) into a single comprehensive incident report.
* **Accelerates Decision-Making:** Provides clear recommendations and prioritization.
* **Scalability:** Operates locally, minimizing reliance on cloud APIs and allowing for scalable deployment without model retraining.
* **Contextualization:** Generates reports with an accuracy goal of at least $75\%$, providing deep context on the severity and impact of the threat.



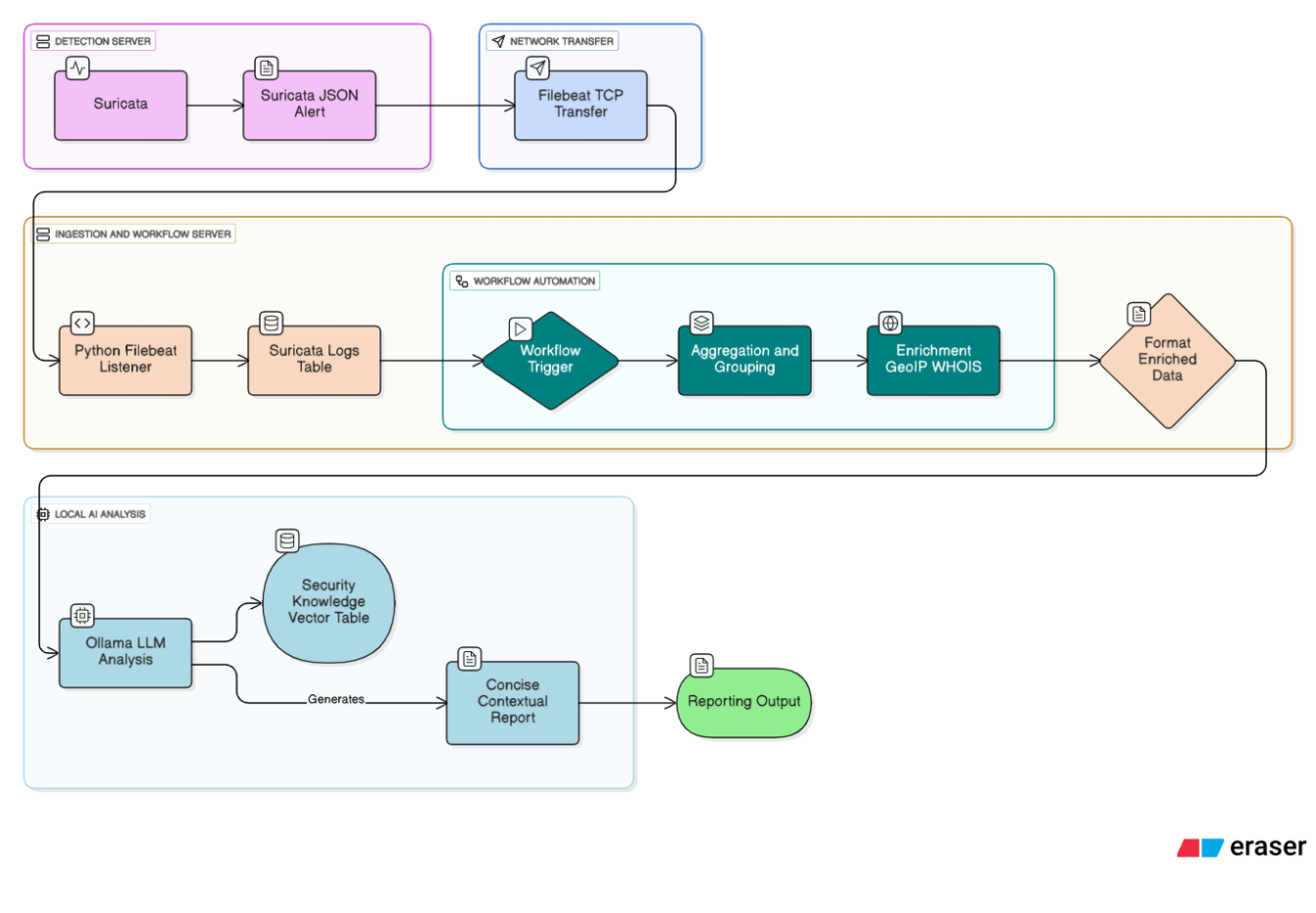
## 2. Technical Architecture and Workflow

The Sentry solution is deployed across a virtualized lab environment hosted on **WindowsHost** using Hyper-V. The architecture is designed for clear separation of network roles and service responsibilities.

### 2.1 Technology Stack

|  |  |  |
| --- | --- | --- |
| **Component** | **Role** | **Technology** |
| **IDS Sensor** | Network Traffic Analysis & Alert Generation | Suricata (running on SentryServe2) |
| **Log Transport** | Secure, lightweight log shipping from sensor to host | Filebeat |
| **Automation Engine** | Workflow orchestration, normalization, and enrichment | n8n (running in Docker on WindowsHost) |
| **Data Storage** | Structured and vector storage for alerts and RAG data | PostgreSQL with vector extension (Docker) |
| **LLM Inference Engine** | Local deployment of Large Language Models | Ollama (running on WindowsHost) |
| **Operating Systems** | Lab environment virtual machines | Ubuntu Server 24.04.3 LTS (for Sentry Servers), Windows 10 Professional (Host) |

### 2.2 Sentry Data Flow (The 'Loop')

1. **Detection:** Suricata (on SentryServe2) detects suspicious network activity and generates a structured JSON alert.
2. **Log Shipping:** Filebeat sends the raw Suricata JSON log over TCP to the WindowsHost machine.
3. **Data Ingestion:** A **Python Filebeat Listener** script on WindowsHost receives the log, performs initial parsing, and pushes it into the **PostgreSQL suricata\_logs** table.
4. **Workflow Trigger:** The n8n workflow, which interfaces with PostgreSQL, is triggered by new or aggregated logs.
5. **Aggregation & Enrichment (n8n):**
   * Logs are grouped into unique **incident groups** based on similarity in timestamps, source/destination IPs, and alert signatures.
   * External enrichment data (GeoIP, WHOIS) is pulled and stored.
6. **LLM Analysis (Ollama/RAG):**
   * The enriched incident data is formatted into a sophisticated **system prompt**.
   * The prompt is injected into the local LLM (Gemma:4B/1B) via Ollama.
7. **Reporting:** The LLM generates a concise, contextual report that is then outputted (e.g., to a web page or final documentation).

### 2.3 LLM Selection and RAG Justification

The project utilizes a dynamic **Context Injection** approach to provide the LLM with current, external knowledge. Instead of relying on pre-indexed vector stores, which are compute intensive, the system performs real-time data retrieval via HTTP requests to specific APIs—specifically the **NIST National Vulnerability Database (NVD)** for CVE data and **IP Whois** for geolocation and network ownership details. This ensures the model analyzes the most up-to-date security information available, strictly preventing hallucination by grounding the generation in live API responses.

* **Inference Engine:** **Ollama** was selected for its ease of installation and network exposure on Windows, enabling robust, local, and GPU-optional inference.
* **Model Choice:** **Gemma3:4B** and **Gemma3:1B** were chosen due to hardware constraints (CPU-based inferencing).
  + **Justification:** These models are optimized for superior efficiency and instruction-following within their parameter class. Advanced techniques like **Grouped-Query Attention (GQA)** drastically reduce KV-cache memory overhead, resulting in faster local inference speed even when processing large JSON payloads from API responses.
* **Data Retrieval Architecture:**
  + **Direct API Integration:** Rather than a vector database, the system utilizes a standard HTTP client to interface directly with external services.
  + **Context Injection:** Raw JSON data retrieved from the NVD and IP Whois APIs is parsed and injected directly into the LLM’s context window (system prompt). This eliminates the complexity of maintaining embeddings and guarantees that the security analysis is based on exact, real-time registry data rather than similarity-based retrieval.

## 3. Implementation Details

### 3.1 Virtualized Environment (Hyper-V)

The lab environment uses two dedicated Hyper-V network switches:

* **'WAN Switch':** External network type, bridged to the physical adapter (Subnet: 192.168.0.0/24).
* **'LAN Switch':** Internal network type for client-server communication (Subnet: 192.168.50.0/24).

#### A. SentryServe1 (Router/Firewall/Gateway - *Initial Design*)

* **Role:** Initially designed to host Suricata and n8n, and act as the network gateway/router.
* **OS:** Ubuntu Server 24.04.3 LTS.
* **Networking Configuration:**
  + Configured interfaces using netplan for static IP assignment.
  + Enabled **IP Forwarding** via /etc/sysctl.conf.
  + Configured **NAT/Masquerading** and general firewall rules using iptables and made persistent with iptables-persistent.
  + Implemented a **DHCP Server** using ISC DHCP Server to assign IPs on the internal LAN network.

#### B. SentryServe2 (IDS Sensor - *Revised Design*)

* **Role:** Dedicated Suricata IDS sensor. The n8n deployment was moved to WindowsHost due to initial VM configuration issues.
* **OS:** Ubuntu Server 24.03.
* **Networking:** Configured as a network gateway/router to place it in the network path, making the lab setting more realistic.
* **Log Shipping:** **Filebeat** was installed and configured to:
  + Monitor Suricata logs.
  + Output to the output.logstash section in /etc/filebeat/filebeat.yml to send logs over TCP to the WindowsHost machine.

### 3.2 WindowsHost (Central Processing and AI)

WindowsHost runs the orchestration, data storage, and AI components.

#### A. Ollama Installation

* Ollama was installed via the official Windows installer and explicitly **exposed to the network** via its settings interface to allow communication from n8n and the Python service.

#### B. PostgreSQL Database Deployment

* PostgreSQL was deployed using **Docker Compose** to manage structured and vector data.
* **Database Schema:**
  + suricata\_logs: Stores all raw and parsed alert data. Key fields include alert\_id (unique), timestamp, src\_ip, dst\_ip, and incident\_group\_id (for n8n grouping).
  + security\_knowledge: Stores the RAG data. Key fields include title, content, source\_url, and the embedding vector (dimension 512).

#### C. n8n Automation Engine Deployment

* n8n was deployed in **Docker Desktop** using Docker Compose for reproducibility and ease of setup.
* **Workflow Design:** The n8n workflow is responsible for receiving parsed logs from the Python script, enriching them, grouping them into incidents, and ultimately triggering the LLM analysis.

## 4. Demonstrable Vulnerabilities for Testing

To effectively test the Sentry workflow, highly reliable and easily replicable Remote Code Execution (RCE) and Information Disclosure vulnerabilities are necessary. These events generate clear, high-priority Suricata alerts that the LLM must process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Vulnerability** | **CVE ID** | **Type** | **Replication Method** | **Suricata Signature Examples** |
| **Log4Shell** | CVE-2021-44228 | RCE | Deploy a vulnerable Log4j Docker container. Trigger exploit using a malicious JNDI string via curl in an HTTP header (e.g., User-Agent). | ET EXPLOIT Apache log4j RCE Attempt (http ldap) |
| **Shellshock** | CVE-2014-6271 | RCE | Deploy a vulnerable web server/CGI script. Trigger exploit by sending a malicious HTTP header (e.g., User-Agent or Cookie) containing an injected bash function. | ET EXPLOIT Bash CGI environment variable injection attempt |
| **Heartbleed** | CVE-2014-0160 | Info Disclosure | Deploy an unpatched OpenSSL server. Use a Metasploit module or public PoC script to send malformed TLS heartbeat requests. | ET POLICY OpenSSL Heartbleed Attempt |
| **EternalBlue** | CVE-2017-0144 | RCE | Use an unpatched Windows 7 VM as a target. Exploit using the Metasploit framework's exploit/windows/smb/ms17\_010\_eternalblue module. | ET EXPLOIT SMB Multiple Exploits - attempt |