Unified Agentic SOC Platform: AI-Driven Threat Hunting & Incident Response

Project Overview

The **Unified Agentic SOC Platform** is a comprehensive Python-based desktop application designed to streamline the lifecycle of security operations. It bridges the gap between manual log analysis, automated threat hunting, and formal incident reporting.

Built to assist both in **Capture The Flag (CTF)** competitions and real-world **Azure Sentinel** investigations, this tool acts as a "Force Multiplier" for SOC analysts. It creates a *Single Pane of Glass* where an analyst can ingest raw logs (PDF/DOCX/TXT), query live cloud environments (Azure Log Analytics), managing investigation state, and generate professional reports without context switching.

Key Capabilities

1. Agentic Azure SOC Analyst

The core of the platform is an AI agent capable of translating natural language intent into executable **Kusto Query Language (KQL)**.

- **Text-to-KQL:** Translates "Find failed logins from external IPs" into precise KQL syntax tailored to the specific schema (e.g., `SigninLogs`, `DeviceProcessEvents`).
- **Self-Healing Logic:** If a query fails (syntax error or zero results), the agent analyzes the error message, adjusts the time range or syntax, and suggests a "fixed" query automatically.
- **Batch Analysis:** To bypass LLM token limits, large result sets from Azure are retrieved and processed in intelligent batches (e.g., 50 records at a time) to identify anomalies.

2. Threat Hunter (CTF Mode)

A module designed for offline log analysis or static forensic artifacts.

- **Multi-Format Ingestion:** Extracts text from PDF logs, Word documents, and raw text files.
- **Context-Aware Hunting:** The AI "remembers" previously found flags and active hints to avoid duplicate findings.
- **Verification Loop:** Unlike "black box" tools, this keeps the analyst in the loop. The AI proposes a finding, and the analyst must **Verify**, **Edit**, or **Discard** it before it enters the evidence chain.

3. "Flag Bank" & Context Memory

A centralized state management system that mimics how a human analyst builds a mental model of an attack.

- **Accumulated Knowledge:** As flags are verified, they are stored in a "Flag Bank."
- **Narrative Generation:** A dedicated background thread uses AI to constantly rewrite a "Current Incident Understanding" narrative based on confirmed facts.
- **Context Injection:** This narrative is dynamically injected into future queries. *Example: If we confirmed the attacker is using IP `10.10.10.5`, the SOC Agent knows to correlate future queries against that IP.*

4. Automated Reporting

Eliminates the tedious paperwork of Incident Response.

- **One-Click Report Generation:** Compiles the timeline, confirmed flags, methodology, and AI-generated narrative into a professional Word (`.docx`) document.
- **Evidence Management:** Allows users to attach screenshots and captions to specific findings.

Technical Architecture

- **Language:** Python 3.10+
- **GUI Framework:** Tkinter (Custom styled with `ttk` for a modern dashboard look)
- **AI Integration:** OpenAI API (`gpt-4o` / `gpt-4o-mini`) using extensive prompt engineering and context management.
- **Cloud Integration:** Azure SDK (`azure-identity`, `azure-monitor-query`) for direct Log Analytics interaction.
- **Document Processing:** `pypdf` for PDF parsing, `python-docx` for Word document generation.
- **Concurrency:** Heavy use of Python `threading` to ensure the GUI remains responsive while running complex KQL queries or long-polling AI tasks.

Key Challenges Solved

The "Hallucination" Problem in KQL

Challenge: LLMs often invent KQL operators or misremember table schemas. **Solution:** I implemented a **Self-Healing Loop**. The tool captures the exception from the Azure SDK, feeds the specific error and the failed query back to the AI, and requests a syntax correction. This reduced query failures by over 80%.

Token Limits vs. Big Data

Challenge: SOC logs can be millions of lines long. Passing them all to GPT-4 is impossible. **Solution:** I engineered a **Batch Processor**. The tool retrieves raw data, splits it into chunks of 50-100 records, and summarizes findings per chunk. These summaries are then aggregated into the master timeline.

State Management

Challenge: Keeping track of what the analyst has already found versus what they are looking for. **Solution:** I built a robust **Session Manager** that serializes the entire application state (flags, hints, API keys, chat history) to JSON. This allows analysts to pause an investigation and resume exactly where they left off days later.

Usage Workflow

1. **Load Data:** Drag and drop PDF logs or connect to Azure Workspace.
2. **Set Focus:** Select "Flag 1" from the dropdown to focus the AI's attention on a specific clue.
3. **Investigate:** Use the SOC Agent to run queries. "Show me processes running from Temp."
4. **Verify:** When the AI spots `malware.exe`, click **Verify**. Add a note: "Confirmed C2 beacon."
5. **Synthesize:** The Flag Bank updates the narrative: *"Attacker established persistence via malware.exe..."*
6. **Report:** Click **Generate Report** to produce the final deliverable.

Future Improvements

- **Graph Visualization:** Integrating `networkx` to visualize relationships between IPs and Processes.
- **Local LLM Support:** Adding adapters for Ollama/Llama 3 for air-gapped investigations.
- **STIX/TAXII Export:** Exporting findings in standardized threat intelligence formats.