AI Sentinel: Technical Architecture & System Flow

This document provides a comprehensive technical breakdown of the final AI Sentinel platform, covering the logic, tools, and architectural decisions made throughout development.

System Overview

AI Sentinel is a Security Orchestration, Automation, and Response (SOAR) platform designed to detect and mitigate "Shadow AI" usage. It combines traditional rule-based detection with modern agentic AI orchestration.

The Technical Flow

1. Request Ingestion & Pre-Processing

When a log entry is analyzed, it flows through a multi-stage pipeline:

- Static Analysis: The 'ShadowAIDetector' ('src/detector.py') immediately runs regex-based scanners ('SENSITIVE_PATTERNS') to identify IBANs, account numbers, and PII.
- Policy Lookup: The domain is checked against 'APPROVED_DOMAINS' and 'MALICIOUS_DOMAINS' defined in 'src/policies.py'.
- Threat Intelligence: A simulated VirusTotal check is performed for external URLs to determine reputation.

2. Agentic Orchestration (LangGraph)

If a request is flagged as external or suspicious, it is handed off to the 'SecurityAgent' ('src/agents.py').

- Stateful Graph: Built using LangGraph, the agent maintains a state containing the log, the evolving analysis, and a list of taken actions.
- Analyzer Node: Utilizes GPT-4 (or Llama 3.2 via Ollama) to assess the "Intent" and "Risk Category". It evaluates context that static rules might miss.
- Mitigator Node: Decides on automated side-effects.
- CRITICAL Risk: Triggers the 'WebhookManager' to block the source IP and creates a SOC incident.
- HIGH Risk: Creates a SOC incident and broadcasts notifications.
- Deterministic Overrides: A fail-safe logic ensures that if pre-screening confirmed a malicious domain or VT hit, the Agent *must* return a 'CRITICAL' status, bypassing LLM uncertainty.

3. Automated Response (SOAR)

- Webhooks ('src/webhooks.py'): Simulates API calls to corporate firewalls for IP blocking.
- Notifications ('src/notifications.py'): A multi-channel broadcast system that mocks alerts to Slack and Microsoft Teams.

4. Behavioral Analytics & Intelligence

- Analytics Engine: The 'get_analytics' method in 'src/detector.py' uses Pandas to process batch results, generating:
- Risk heatmaps by Department.

- Sensitive data exfiltration breakdowns.
- Temporal risk trends.

5. Virtual Security Assistant (VSA)

- SecurityAdvisor ('src/agents.py'): A separate LLM-driven class that provides conversational support.
- RAG-lite Context: Unlike general chatbots, the VSA is injected with a JSON snapshot of the *current* analysis results, allowing it to answer specific questions like "Why was this specific user blocked?" accurately.

Technology Stack

Layer	Component	Description
Orchestration	LangGraph	State machine for security logic flow
Logic/AI	LangChain + OpenAI/Ollama	LLM integration and prompt management
Frontend	Streamlit	Real-time interactive dashboard
Data Processing	Pandas	Backend analytics and statistics
Automation	Custom Webhook/Notification Modules	Mocked side-effects for SOAR
Security	Custom Policy Engine	Regex and domain-based safety rules

Logic Principles

1. Defense in Depth: Multiple layers of detection (Regex -> Domain -> VT -> Agent).
2. Human-in-the-loop: High-risk actions are logged in the "Action Center" for manual confirmation, balancing automation with oversight.
3. Explainability: Every risk assessment includes a 'reasoning' field generated by the agent, ensuring security teams understand the "Why" behind every alert.