SOC-in-a-Box: A Multi-Agent LLM-Based Security Operations Center for Threat Detection and Automated Incident Response

Disha S[1], *Pallavi U*[2], *Devika K A*[1]

[1]*UG, Dept. of CSE, Atria Institute of Technology, Bengaluru, Karnataka, India – 560024*

[2]*Assistant Professor, Dept. of CSE, Atria Institute of Technology, Bengaluru, Karnataka, India – 560024*

Emails: 1at23cs046@visioneer.atria.edu, pallavi_cse@atria.edu.in, 1at23cs044@visioneer.atria.edu

Abstract

Modern Security Operations Centres struggle with overwhelming alert volumes, chronic analyst shortages, and slow incident response times. This paper presents SOC-in-a-Box, a multi-agent prototype that automates the three core SOC functions—detection, investigation, and response—using specialised AI agents powered by a large language model (LLM). The Sentry agent monitors log files and flags suspicious events using either LLM classification or a built-in rule engine. The Investigator agent gathers related evidence from across the log corpus and asks the LLM to produce a structured root-cause analysis. The Responder agent selects a policy-approved containment action, executes it in a simulated or live environment, and generates a structured Markdown incident report. All three agents run as lightweight Python threads connected through in-memory queues with no external message broker. When the LLM is unavailable, a deterministic fallback engine ensures the pipeline continues to operate. Evaluation across five attack categories—brute-force, data exfiltration, privilege escalation, port scanning, and malware deployment—shows complete detection coverage with end-to-end latency below 60 seconds on a standard laptop. The system demonstrates that a self-contained, locally deployable multi-agent architecture can meaningfully reduce manual effort in routine SOC workflows while preserving human oversight on critical decisions.

Keywords: Autonomous monitoring; incident response; large language models; multi-agent systems; threat detection

1. INTRODUCTION

The cybersecurity landscape has evolved dramatically over the past decade. Organisations of all sizes depend on Security Operations Centres (SOCs) to defend their digital infrastructure against an increasing volume and variety of threats. A SOC typically operates around the clock, with analysts monitoring dashboards, triaging alerts, investigating suspicious activity, and coordinating incident response [1]. However, the effectiveness of this model is being eroded by three converging pressures.

First, the volume of security telemetry has grown beyond what human analysts can reasonably process. A mid-sized organisation may generate tens of thousands of log events per day from firewalls, authentication servers, endpoint agents, and network devices [6]. A large portion of these events trigger alerts, the majority of which are false positives. Industry surveys consistently report that SOC analysts spend upwards of 70% of their time investigating alerts that turn out to be benign, a condition commonly referred to as alert fatigue [6]. Second, there is a well-documented shortage of skilled cybersecurity professionals. The global cybersecurity workforce gap was estimated at 3.4 million in 2024, meaning that many organisations simply cannot hire enough analysts to staff their SOCs adequately. Small and medium enterprises and educational institutions are particularly affected, as they lack the budgets to attract experienced personnel or to purchase expensive commercial tools like Splunk or IBM QRadar [6]. Third, the speed at which attackers operate has increased. Automated toolkits and AI-assisted attack techniques allow adversaries to move from initial compromise to data exfiltration in a matter of hours. Meanwhile, the median time for an organisation to detect and contain a breach remains measured in weeks [6]. This gap between attack speed and response speed is the central problem that modern SOCs must address.

Recent advances in large language models (LLMs) and multi-agent systems present a potential path forward. LLMs have demonstrated the ability to read unstructured text, identify patterns, and produce coherent natural-language explanations [1], [6]. Multi-agent architectures allow complex workflows to be decomposed into specialised roles, each handled by a dedicated agent [2], [5]. When combined, these technologies can form a system that detects threats, investigates their root cause, and takes bounded response actions with minimal human involvement.The objective of this work is to develop a lightweight, autonomous Security Operations Centre (SOC) framework that can detect, investigate, and respond to web application security threats through a coordinated multi-agent architecture.Unlike conventional SOC approaches that often rely on separate tools and human-driven workflows for detection, investigation, and response, this work integrates these stages into a coordinated multi-agent architecture. The proposed approach combines real-time log monitoring, evidence correlation, threat-intelligence enrichment, and bounded response actions within a lightweight, locally deployable framework.

This paper presents SOC-in-a-Box, a prototype that realises this vision through three cooperating AI agents:

1) **Sentry**—monitors a log directory using filesystem notifications and classifies each new or modified file as suspicious or benign, using the LLM or a set of 14 built-in rules.
2) **Investigator**—gathers related evidence from other log files and asks the LLM to explain what happened, why, and what the potential impact is.
3) **Responder**—selects a containment action (such as blocking an IP address or locking a user account) and writes a structured incident report.

The system is designed as a self-contained, locally de-

ployable architecture. In the current prototype, NVIDIA NIM provides the LLM inference endpoint, while the orchestration, agents, log processing, and response pipeline run locally. The architecture can be adapted to use a locally hosted LLM for fully offline deployment, and includes a synthetic log generator for repeatable testing. All response actions are simulated by default, making the system safe to demonstrate on any standard computer without risking unintended changes. The remainder of this paper describes the system design, evaluates it against five attack scenarios, and discusses its strengths and limitations.

2. RELATED WORK

Research into intelligent security automation spans several areas, including multi-agent architectures for SOC workflows, LLM-based threat detection, reinforcement learning for autonomous defence, and infrastructure for scalable event processing. This section reviews the most relevant prior work and identifies the gaps that SOC-in-a-Box addresses.

2.1. Multi-Agent SOC Architectures

The idea of decomposing SOC workflows into cooperating agents has gained traction in recent years. Hmimou et al. [1] built a multi-agent cybersecurity framework integrated with LLMs for intelligent threat detection and event correlation, achieving 93.6% accuracy on combined ELK Stack and Suricata telemetry but requiring large, computationally expensive models that did not support offline deployment.

Tang et al. [2] introduced LiaisonAgent, an LLM-based multi-agent framework (built on the QWQ-32B reasoning model) designed to bridge technical risk detection with business-level risk governance in SOC settings. LiaisonAgent uses autonomous planning and tool-calling to coordinate evidence collection and response execution, demonstrating that structured coordination between agents can significantly reduce analyst workload. Mitra et al. [5] examined the security of LLM-powered multi-agent systems, decomposing the attack surface of agent-to-agent integration into component, coordination, and protocol layers and applying the framework to an MCP-based Security Orchestration, Automation, and Response (SOAR) architecture. This is a complementary rather than directly analogous concern to SOC-in-a-Box, which assumes agent-to-agent communication is trusted and instead focuses on the detect–investigate–respond division of labour. Nwukor [4] approached the problem from a reasoning perspective, proposing a role-based multi-agent framework with a five-stage reasoning engine and credibility-weighted knowledge sharing.

Srinivas et al. [6] surveyed the broader landscape of LLMs and agents in SOC environments, covering alert triage, investigation assistance, and automated response. They identified coordination complexity, explainability, and real-time deployment as the primary open challenges. Roy [13] extended this direction with AgenticCyber, a GenAI-powered system for multimodal threat detection across logs, video, and audio streams, while Vinay [14] provided a broader taxonomy tracing the evolution from single LLM reasoners to fully autonomous multi-agent pipelines in cybersecurity.

2.2. Reinforcement Learning for Cyber Defence

An alternative approach to intelligent SOC automation uses reinforcement learning (RL), where agents learn defence strategies by interacting with their environment. Girei et al. [3] demonstrated multi-agent RL for real-time threat mitigation, though their evaluation was limited to simulated environments. Landolt et al. [7] surveyed the theoretical foundations of multi-agent RL applied to intrusion detection and moving-target defence. Singh et al. [8] proposed hierarchical RL that decomposes defence tasks into sub-policies, enabling more scalable multi-agent coordination. Alom et al. [17] proposed an adaptive multi-agent reinforcement learning framework for intrusion mitigation in smart-city environments. Finistrella et al. [16] classified available MARL frameworks according to their suitability for different network configurations, providing guidance on framework selection that informed the design choices behind SOC-in-a-Box.

These works inform the design thinking behind SOC-in-a-Box, though the current prototype relies on LLM classification rather than learned policies.

2.3. Explainability and Trust

A recurring concern in automated SOC systems is that analysts need to understand why the system flagged something, not just that it did. Panchal et al. [9] addressed this by introducing explainable AI overlays for automated SOC playbooks and threat response automation. In an adjacent application domain, Alkahtani et al. [15] developed a trust-based, explainable multi-agent RL framework for secure UAV swarm communication in Flying Ad Hoc Networks (FANETs); while not a SOC system, it illustrates that trust-aware explainability mechanisms for multi-agent RL are being actively developed outside the SOC context as well. SOC-in-a-Box responds to the same underlying concern by requiring the Investigator agent to produce readable root-cause explanations rather than simple severity scores.

2.4. Event-Driven Infrastructure

Mandloi [10] validated Kafka-based event streaming for multi-agent security data processing and coordination, addressing scalability concerns in real-time systems. Punniyamoorthy et al. [11] proposed a four-plane cognitive platform for autonomous cloud operations using Kubernetes and real-time anomaly detection. Mgbemele et al. [12] combined multi-agent RL with federated learning for autonomous threat response and recovery. Rajgopal [20] explored AI copilots as force multipliers in resource-constrained SOCs, aligning with the practical motivation behind SOC-in-a-Box.

2.5. Identified Gaps

The reviewed literature reveals several recurring limitations: (i) high computational requirements that prevent lightweight or offline deployment; (ii) limited real-time validation beyond proof-of-concept stages; (iii) lack of integrated end-to-end

pipelines spanning detection through response; (iv) insufficient explainability in automated decisions; and (v) absence of built-in safety models to prevent unintended system modifications. SOC-in-a-Box addresses each of these gaps through its lightweight architecture, rule-based fallback, structured narrative outputs, and layered safety design.

3. PROBLEM FORMULATION

The operational challenge addressed by this work can be stated as follows. Given a continuous stream of log events $E = \{e_1, e_2, \ldots, e_n\}$ generated by an IT infrastructure, the objective is to design a system S that performs three functions:

1) **Detection:** Classify each event e_i as suspicious or benign, filtering out routine activity to reduce noise.
2) **Investigation:** For each suspicious event, collect related evidence from the broader log corpus and produce an explanation of the root cause, origin, method, and potential impact.
3) **Response:** Based on the investigation, select and execute a containment action from a predefined set and document the entire chain in a structured report.

The system must satisfy four constraints:

1) **Self-contained:**Self-contained: The architecture is designed to run on a single machine without dependence on external SOC platforms or cloud infrastructure. The current prototype uses an LLM inference endpoint, while a locally hosted LLM can be substituted for fully offline deployment.
2) **Safe by default:** Response actions must not modify the host system unless explicitly authorised.
3) **Resilient:** If the LLM is unavailable, the system must continue operating using built-in rules.
4) **Extensible:** Adding new agents or new log sources should not require changes to existing components.

These constraints reflect the target deployment scenario: an academic or small-team environment where the primary goal is demonstrating and evaluating autonomous SOC capabilities without the infrastructure overhead of a production system.

4. SYSTEM DESIGN

4.1. Architecture Overview

The system implements a linear three-stage pipeline where each agent runs as an independent Python daemon thread. A single entry point (main.py) creates the threads and two shared queues. The Sentry watches a designated log directory through filesystem callbacks provided by the watchdog library. When a log file is created or modified, the Sentry processes it and, if suspicious, places a structured alert on alert_queue. The Investigator consumes this alert, gathers evidence, and pushes an investigation report onto report_queue. The Responder then selects a response action, executes it, and writes a Markdown report to disk. Fig. 1 illustrates this pipeline.

No external message broker (such as Kafka or Redis) is used. The in-process queue approach keeps the system self-

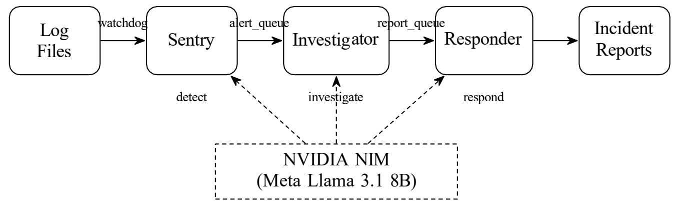

FIGURE. 1. System architecture. Three agents run in separate threads and share a single LLM endpoint hosted on the NVIDIA NIM platform.

contained and The prototype was developed and evaluated using Python 3.11. Adding a fourth agent in the future requires only creating a new class, a new queue, and a new thread in main.py, without modifying the existing agents.

4.2. Sentry Agent

The Sentry is the first layer of defence. It monitors the logs/ directory using the watchdog library, which provides operating-system-level callbacks for file creation, modifica-tion, and deletion. When a file event occurs, the Sentry executes the following sequence:

1) **Debounce:** A per-file tracking set prevents the same file from being processed multiple times while an analysis is already in progress. A configurable cooldown (default 3 seconds) throttles the rate of LLM calls.
2) **Tail read:** The last 60 lines of the modified file are read. This captures the most recent activity, which is typically where burst attacks (like repeated failed logins) appear.
3) **Classification:** The extracted lines are sent to the LLM with a system prompt that asks: “Is this content suspicious? If so, what type of event is it, how severe is it, and what indicators support your classification?” The LLM must respond with a structured JSON object.
4) **Fallback:** If the LLM is unreachable or returns malformed output, a 14-rule regex engine evaluates the content against known attack patterns.
5) **Alert emission:** If the event is classified as suspicious, an alert dictionary is placed on the alert_queue for the Investigator.

If a log file is deleted, the Sentry treats it as evidence tampering and raises a high-severity LOG_TAMPERING alert without consulting the LLM. Table 2 shows a sample of the fallback rules used when the LLM is unavailable.

4.3. Investigator Agent

The Investigator receives an alert from the Sentry and gathers additional context before forming a judgement. It has access to eight tool functions that enable it to search, extract, and correlate information across the log directory:

For a brute-force alert, the Investigator would: (1) extract the source IP from the alert; (2) search all log files for that IP to find related events; (3) check whether the IP appears in the local blocklist of known malicious addresses; (4) count how often the IP has appeared in the last 10 minutes; (5) read surrounding log lines for additional context; and (6) send the

TABLE 1
COMPARATIVE POSITIONING OF SOC-IN-A-BOX AGAINST RELATED SYSTEMS

System	Multi-Agent	LLM-Based	Runs Offline	Automated Response	Full Pipeline	Safety Model
Hmimou et al. [1]	✓	✓	—	—	✓	—
LiaisonAgent [2]	✓	✓	—	✓	✓	—
Mitra et al. [5]	✓	✓	—	✓	partial	—
Girei et al. [3]	✓	—	—	✓	partial	—
Panchal et al. [9]	—	—	—	partial	partial	✓
Mandloi [10]	✓	—	✓	—	—	—
Rajgopal [20]	—	✓	—	partial	partial	—
SOC-in-a-Box	✓	✓	✓	✓	✓	✓

LiaisonAgent [2] is explicitly built on the QWQ-32B model and Mitra et al. [5] explicitly targets LLM-powered multi-agent systems; both are LLM-based.

TABLE 2
SAMPLE FALLBACK DETECTION RULES

Pattern	Event Type	Sev.
Failed password	Brute Force	Med
Invalid user	Brute Force	High
POSSIBLE BREAK-IN	Brute Force	Crit
sudo.*FAILED	Priv. Escalation	High
/etc/shadow	Unauth. Access	Crit
rm\s+-rf	Malware	High
base64\s+-d	Malware	High
nc\s+-e	Malware	Crit
chmod\s+777	Priv. Escalation	Med
\bnmap\b	Unauth. Access	Med

TABLE 3
INVESTIGATOR TOOL FUNCTIONS

Function	Purpose
search_logs	Full-text search across all log files
extract_ips	Find IP addresses in text extract_users
	Find usernames (sshd/sudo/PAM)
count_occur	Count keyword frequency in recent files
check_blocklist	Look up IP in local blocklist get_metadata
	File size and timestamps
open_file	Read entire file (scoped to logs/)
read_lines	Read specific line range

collected evidence to the LLM with a prompt asking for a structured root-cause analysis.

The LLM is expected to return an explanation covering four dimensions: why the event occurred (root cause), where it originated (IP, user, or file path), how it was carried out (technique), and what the potential impact is. It also assigns a confidence level (high, medium, or low) and recommends an action for the Responder.

4.4. Responder Agent

The Responder receives the Investigator's analysis and decides what action to take. It queries the LLM to select from a fixed set of response actions and to produce a complete incident report in Markdown format.

After executing the chosen action, the Responder writes a Markdown report to the reports/ directory. The re-port includes: an incident ID and timestamp, the severity and event type, a summary of what happened, the evi-

TABLE 4
AVAILABLE RESPONSE ACTIONS

Action	What It Does
NO_ACTION	Event is benign; take no action
MONITOR	Log the event for future reference
BLOCK_IP	Block the source IP address
LOCK_USER	Disable the affected user account
QUARANTINE_FILE	Move a suspicious file to quarantine
DELETE_FILE	Remove a confirmed malicious file
ESCALATE_HUMAN	Alert a human operator for manual review

dence collected by the Investigator, the root-cause analysis (WHY/WHERE/HOW/IMPACT), the action taken, and recommendations for preventing recurrence. If the action is ESCALATE_HUMAN, the Responder additionally prints a vis-ible warning banner to the terminal and writes a notification to a separate log file.

By default, all actions are simulated: the system logs what it would do (including the exact shell command it would run) without actually executing it. This makes the system safe to test and demonstrate on any machine.

4.5. LLM Integration

All three agents share a single connection to the NVIDIA NIM platform, which hosts the Meta Llama 3.1 8B Instruct model through an OpenAI-compatible REST API. The model is configured with a low temperature (0.2) for consistent, structured output. If the LLM returns an error or invalid JSON, each agent activates its built-in fallback: the Sentry uses its regex rules, the Investigator returns a low-confidence analysis with a recommendation to escalate, and the Responder defaults to ESCALATE_HUMAN.

4.6. Safety Design

Because the Responder can take actions like blocking IPs and locking accounts, the system includes several safeguards to prevent unintended consequences:

1) **Simulate-only mode** is the default; no real system changes are made unless explicitly authorised via a command-line flag.
2) **Command whitelist:** only about 40 approved shell commands are permitted; anything else is blocked auto-matically.

TABLE 5
BUILT-IN ATTACK SCENARIOS FOR TESTING

Scenario	What It Generates
Brute Force	15 failed SSH login attempts from one IP, followed by one success
Exfiltration	User login, 500 MB outbound transfer, access to /etc/shadow
Priv. Esc.	Sudo failure, break-in warning, chmod 777 on a suspicious file
Port Scan	Nmap detection plus SYN probes on common ports
Malware	Download from attacker domain, encoded payload decode, reverse shell

3) **Filesystem scoping:** file reads are limited to the logs/ directory; file deletions are limited to the reports/ directory.
4) **Input validation:** IP addresses and usernames are checked for correct format before any action is attempted.
5) **Timeout control:** all shell commands run with a 10-second timeout.

Securing the identity and intent of autonomous agents in distributed environments is an active area of research; Bhushan [18] proposed intent-aware identity management for always-on agents, a concern that becomes relevant as the system scales beyond a single host.

4.7. Test Log Generator

A standalone script generates realistic log entries for testing and demonstration. It supports five attack scenarios and three operating modes:

In normal mode the script emits routine log lines only. In attack mode it repeats a chosen scenario. In mixed mode it interleaves normal logs with random attacks every few cycles, producing a more realistic testing environment.

5. IMPLEMENTATION DETAILS

The prototype is implemented in Python 3.11 and requires only two pip dependencies: watchdog for filesystem monitoring and requests for HTTP communication with the NIM API. An optional .env file stores the NVIDIA API key, loaded at startup via python-dotenv.

5.1. Threading and Communication

Each agent is implemented as a Python class. The Sentry starts a watchdog observer in a daemon thread; the Investigator and Responder each run in their own daemon threads, looping on their respective queues with a 1-second timeout. A shared threading.Event coordinates graceful shutdown: when the main process receives a keyboard interrupt, it sets the event, and each agent's stop() method is called with a 5-second join timeout.

Queue payloads are Python dictionaries containing structured data. An alert dictionary from the Sentry includes the raw log content, the detected severity, the event type, and any extracted indicators. An investigation report from the Investigator includes the original alert, all collected evidence, and the LLM-generated analysis.

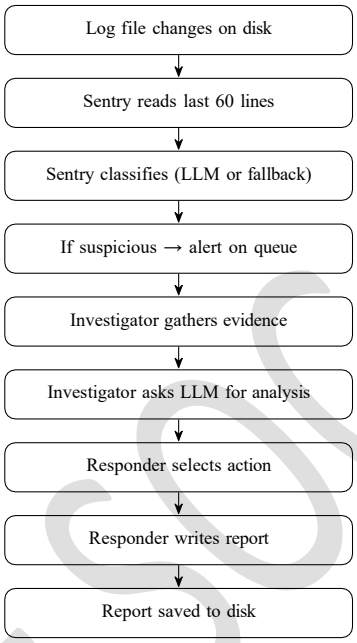

FIGURE. 2. Step-by-step flow from a log file event to a saved incident report.

5.2. LLM Client Design

The NvidiaNimClient class wraps the NIM REST API using the OpenAI-compatible /v1/chat/completions endpoint. Key design decisions include: exponential backoff on HTTP 429 errors (2 s, 4 s, 8 s with a 30-second cap and 3 retries); a JSON extraction routine that strips markdown fences and finds the first complete JSON object; and a health-check endpoint (GET /v1/models) with a 5-second timeout that is tested at startup.

5.3. Responder Execution Model

The Responder's action functions (block_ip, lock_user, etc.) accept a simulate flag. When simulating, the func-tion logs the command it would run (e.g., iptables -A INPUT -s 192.168.1.105 -j DROP) without execut-ing it. When running in live mode, the command is executed through subprocess.run() with array-based invocation (no shell interpolation), a 10-second timeout, and output capture. The command whitelist is checked before execution regardless of the mode.

6. EVALUATION

6.1. Test Environment

The system was tested on a standard laptop with an 8-core processor and 16 GB RAM, running Python 3.11 on Linux. The NVIDIA NIM platform with Meta Llama 3.1 8B Instruct served as the LLM backend. The log generator was run separately for each of the five attack scenarios and once in mixed mode for a combined test.

TABLE 6
DETECTION AND RESPONSE RESULTS BY ATTACK CATEGORY

Attack	Det.	Rep.	Action Taken
Brute Force	✓	✓	BLOCK_IP
Exfiltration	✓	✓	LOCK_USER + QUARANTINE_FILE
Priv. Esc.	✓	✓	LOCK_USER
Port Scan	✓	✓	MONITOR
Malware	✓	✓	BLOCK_IP + DELETE_FILE

TABLE 7
PIPELINE TIMING MEASUREMENTS

Pipeline Stage	Time
Sentry (detection)	<10 s
Investigator (evidence + LLM)	15–25 s
Responder (action + report)	15–30 s
End-to-end total	<60 s

6.2. Detection Results

Table 6 summarises the outcome for each attack scenario. Every attack was correctly identified by the Sentry, and every incident report was successfully generated by the Responder. The brute-force scenario produced the most complete chain of evidence. The Sentry flagged 15 failed SSH logins as a brute-force attempt. The Investigator found the same IP address in the local blocklist (listed as a known Tor exit node) and counted its recent appearances. The Responder generated a simulated iptablesDROP rule and documented the full chain in the incident report.

The exfiltration scenario was the most demanding because no fallback regex rule covers netflow anomalies. Detection relied entirely on the LLM, which correctly identified the 500 MB outbound transfer combined with prior access to /etc/shadow as indicative of data exfiltration. The Inves-tigator correlated the user session and the file access, and the Responder took two actions: locking the user account and quarantining the accessed file.

6.3. Pipeline Timing

Table 7 reports the time taken for each stage of the pipeline. Measurements were taken over 10 runs for each scenario and the ranges represent the minimum and maximum observed values.

The Sentry stage was consistently the fastest because it involved only one short LLM call on a 60-line text fragment. The Investigator stage was the most variable because the number of tool invocations depended on how many IPs and usernames were found in the alert. The Responder stage was moderately consistent, dominated by the LLM call to generate the Markdown report.

6.4. Fallback Reliability

To test the fallback engine, the LLM endpoint was deliberately made unavailable by setting an invalid API URL. In all five attack scenarios, the fallback engine activated automatically in all three agents. The Sentry's regex rules correctly classified brute-force, privilege-escalation, port-scan, and malware events. The exfiltration scenario was not caught by the fallback because it requires reasoning about netflow volume, which no regex rule covers. In this case, the Sentry classified the event as benign, and no alert was raised—a known limitation of the fallback approach.

When the Sentry did raise an alert (the other four scenarios), the Investigator returned a low-confidence analysis with a recommendation to escalate, and the Responder defaulted to ESCALATE_HUMAN. No crashes or unhandled errors occurred across 50 fallback trials.

TABLE 8
DETECTION METHOD COVERAGE BY ATTACK TYPE

Attack	LLM	Fallback	Correlation
Brute Force	✓	✓	✓
Exfiltration	✓	—	✓
Priv. Esc.	✓	✓	✓
Port Scan	✓	✓	—
Malware	✓	✓	—

6.5. Report Quality

Every generated report contained the expected sections: incident ID, timestamp, severity, event type, summary, evidence listing, root-cause analysis table, actions-taken JSON block, and recommendations. Reports produced with LLM support consistently included MITRE ATT&CK tactic labels (such as credential_access for brute-force attacks) and detailed explanations. Reports generated during fallback mode lacked these labels and had less detailed explanations, but were still structurally complete.

7. DISCUSSION

7.1. Design Rationale

Several design choices were made deliberately to suit the target deployment scenario.

Why three agents? Separating detection, investigation, and response into distinct agents mirrors how a real SOC team operates, where different analysts handle different stages. This separation also makes the system easier to extend: a new agent can be added by creating a new class and queue without modifying existing code.

Why use an LLM? Pattern-matching rules can catch known attack signatures, but they cannot explain why something is malicious or assess its likely impact. The LLM gives the Investigator the ability to produce a readable explanation that a human reviewer can act on. This is particularly valuable for complex attack patterns like data exfiltration, where simple regex rules are insufficient.

Why simulate responses? Taking real actions (blocking IPs, locking accounts) on a production system carries significant risk. The simulate-only default allows the system to be tested, demonstrated, and evaluated safely, while the -live

flag provides a clear upgrade path for environments where real response is appropriate.

Why in-process queues? Using Python's built-in queue.Queue instead of an external message broker keeps the system simple and self-contained. The trade-off is that the system cannot scale across multiple machines, but this is acceptable for a single-host prototype.

7.2. Limitations

The current prototype has several limitations that should be considered when interpreting the results:

1) **Single-directory monitoring:** The Sentry watches only one local directory. Real environments aggregate logs from many sources, which would require an additional ingestion layer.
2) **Raw text processing:** Logs are read as plain text without parsing structured formats like JSON or CEF. This limits the precision of field extraction.
3) **LLM reliability:** At low temperature, the LLM occasionally produces invalid JSON or incorrect conclusions. The fallback engine catches most failures but cannot match LLM-level reasoning for complex patterns.
4) **Throughput:** The 3-second cooldown between API calls limits the system to about 20 events per minute, which is adequate for testing but would need optimisation for higher-volume environments.
5) **No persistent state:** Agent state is held in memory. A crash would lose any in-flight alerts that have not yet been written to a report.
6) **Exfiltration gap:** The fallback engine cannot detect data exfiltration because it relies on regex patterns that do not cover netflow volume anomalies.

7.3. Implications

Despite these limitations, the evaluation demonstrates that even a simple multi-agent LLM-backed system can handle the core SOC workflow effectively. The consistent detection of all five attack categories and the production of readable, structured reports suggest that this approach is viable as a teaching tool, a research platform, or a starting point for a more production-ready system. The layered safety model makes it particularly suitable for environments where experimentation must not affect live systems.

8. CONCLUSION AND FUTURE WORK

This paper presented SOC-in-a-Box, a multi-agent prototype that automates the detect–investigate–respond cycle at the heart of SOC operations. Three cooperating agents—Sentry, Investigator, and Responder—handle detection, evidence gathering, and response execution respectively, using a shared LLM endpoint for classification and narrative generation. The system correctly identifies five common attack patterns, produces structured incident reports with root-cause explanations, and completes each cycle in under 60 seconds on a standard laptop. A rule-based fallback engine ensures that the pipeline continues operating when the LLM is unavailable, and a layered safety model prevents unintended system modifications. Several directions for future work have been identified:

1) **Streaming log ingestion:** Replacing the watchdog-based monitoring with a streaming pipeline (e.g., Filebeat to Kafka to Sentry) would allow the system to handle logs from multiple machines simultaneously.
2) **Historical pattern memory:** Storing past incidents in a vector database would let the Investigator compare new alerts against previous cases, improving accuracy over time.
3) **Confidence-gated response:** Restricting high-impact actions (like blocking IPs) to cases where the Investigator's confidence is high would reduce the risk of false-positive responses.
4) **Web dashboard:** A real-time monitoring interface would make the system more accessible to non-technical users and support live demonstrations.
5) **Multi-model support:** Allowing different agents to use different models (for example, a smaller model for the Sentry's quick classification and a larger model for the Investigator's detailed analysis) could improve both speed and accuracy. Yang and Shami [19] demonstrated a related direction through an AutoML framework that automatically selects and tunes intrusion detection models, suggesting that automated model selection could further reduce the configuration effort required to deploy the system in new environments.
6) **Human feedback loop:** Allowing analysts to label the quality of generated reports could be used to refine the LLM prompts over time.

REFERENCES

[1] Y. Hmimou, M. Tabaa, A. Khiat, and Z. Hidila, "A multi-agent system for cybersecurity threat detection and correlation using large language models," *IEEE Access*, vol. 13, pp. 150199–150215, 2025.

[2] C. Tang, L. Qing, and S. Chen, "LiaisonAgent: an multi-agent framework for autonomous risk investigation and governance," arXiv:2603.00200, 2026.

[3] A. A. Girei, F. Abraham, A. O. Majekodunmi, and J. Alebiosu, "Autonomous cyber defense agents: a reinforcement learning approach to real-time threat mitigation," *Int. J. Comput. Appl.*, vol. 187, no. 46, pp. 32–41, Oct. 2025.

[4] I. Nwukor, "Role based multi-agent reasoning frameworks," *Int. J. Comput. Appl.*, vol. 187, no. 78, Feb. 2026.

[5] S. Mitra, R. Patel, S. Mittal, M. R. Rahman, and S. Rahimi, "AgenticCyOps: securing multi-agentic AI integration in enterprise cyber operations," arXiv:2603.09134, 2026.

[6] S. Srinivas, B. Kirk, J. Zendejas, M. Espino, M. Boskovich, A. Bari, K. Dajani, and N. Alzahrani, "AI-augmented SOC: a survey of LLMs and agents for security automation," *J. Cybersecur. Priv.*, vol. 5, no. 4, art. 95, 2025.

[7] C. R. Landolt, C. Würsch, R. Meier, A. Mermoud, and J. Jang-Jaccard, "Multi-agent reinforcement learning in cybersecurity: from fundamentals to applications," arXiv:2505.19837, 2025.

[8] A. V. Singh, E. Rathbun, et al., "Hierarchical multi-agent reinforcement learning for cyber network defense," arXiv:2410.17351, 2024.

[9] R. V. Panchal, R. Snehkunj, and V. V. Panchal, "Explainable artificial intelligence (XAI) for intelligent intrusion detection systems and threat response automation," *Int. J. Comput. Appl.*, vol. 187, no. 74, pp. 51–55, Jan. 2026.

[10] A. Mandloi, "Event-driven architectures with Apache Kafka: supporting agentic AI and big data analytics in banking transformations," *Int. J. Comput. Appl.*, vol. 187, no. 79, pp. 5–13, Feb. 2026.

[11] V. Punniyamoorthy, N. Saksena, S. R. Sankiti, N. Chockalingam, A. M. Kirubakaran, S. K. R. Carimireddy, and D. Maruthavanan, "Cognitive platform engineering for autonomous cloud operations," *Int. J. Comput. Appl.*, vol. 187, no. 72, pp. 17–23, Jan. 2026.

[12] A. Mgbemele et al., "Autonomous AI defense with MARL coordination and federated learning for real-time threat response," *All Multidiscip. J.*, 2025.

[13] S. Roy, "AgenticCyber: a GenAI-powered multi-agent system for multimodal threat detection and adaptive response in cybersecurity," arXiv:2512.06396, 2025.

[14] V. Vinay, "The evolution of agentic AI in cybersecurity: from single LLM reasoners to multi-agent systems and autonomous pipelines," arXiv:2512.06659, 2025.

[15] H. K. Alkahtani et al., "Explainable multi-agent reinforcement learning framework for secure and adaptive communication in UAV swarm-based FANETs," *Sci. Rep.*, 2026.

[16] S. Finistrella, S. Mariani, and F. Zambonelli, "Multi-agent reinforcement learning for cybersecurity: classification and survey," *Intell. Syst. Appl.*, vol. 26, art. 200495, 2025.

[17] J. Alom, M. S. Ullah, M. T. Islam, M. Niloy, R. Islam, and S. Firdaus, "Adaptive multi-agent reinforcement learning for intrusion mitigation aligned with smart city," in *Proc. 2025 Int. Conf. Quantum Photonics, Artificial Intelligence, and Networking (QPAIN)*, IEEE, Jul. 2025, pp. 1–6, doi: 10.1109/QPAIN66474.2025.11172093.

[18] B. Bhushan, "Intent-aware identity management for autonomous IIoT: a decentralized, trust-driven security architecture," *Int. J. Comput. Appl.*, vol. 187, no. 53, pp. 30–41, Nov. 2025.

[19] L. Yang and A. Shami, "Towards autonomous cybersecurity: an intelligent AutoML framework for autonomous intrusion detection," in *Proc. Workshop on Autonomous Cybersecurity (AutonomousCyber), ACM CCS*, 2024.

[20] P. R. Rajgopal, "SOC talent multiplication: AI copilots as force multipliers in short-staffed teams," *Int. J. Comput. Appl.*, vol. 187, no. 48, pp. 46–62, Oct. 2025.