MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

Agentic SOC: An LLM-Driven Framework for Automated Incident Triage and Response

Marudhu Paandian K
UG Student, Dept. of CSE KGiSL Institute of Technology Coimbatore, India
marudhu_22csb05@kgkite.ac.in

Rajiv N
UG Student, Dept. of CSE KGiSL Institute of Technology Coimbatore, India
rajiv_22csb19@kgkite.ac.in

Nishanth R
UG Student, Dept. of CSE KGiSL Institute of Technology Coimbatore, India
nishanth_22csb11@kgkite.ac.in

Manikandan S
Assistant Professor, Dept. of CSE KGiSL Institute of Technology Coimbatore, India
manikandan@kgkite.ac.in

***Abstract*—Modern Security Operation Centers (SOCs) face significant challenges in managing the high volume of security alerts, often resulting in increased response latency and analyst burnout. This paper proposes an autonomous framework that utilizes the cognitive reasoning and tool-calling capabilities of Large Language Models (LLMs) to perform Tier-1 incident triage. By serving as an intelligent middleware between detection systems and system actuators, the proposed framework interprets complex alert metadata to execute precise remediation actions in real-time. Experimental validation demonstrates that the agentic approach achieves a Mean Time to Respond (MTTR) of under 10 seconds, successfully bridging the gap between automated detection and manual intervention while maintaining the logical flexibility required for modern security operations.**

***Index Terms*—Security Operations Center, Large Language Models, Automated Incident Response, Tool Calling, Cyber Automation.**

I. Introduction

The rapid digitalization of enterprise infrastructure has led to an exponential increase in the volume of security teleme- try generated across networks. Modern Security Operations Centers (SOCs) are tasked with monitoring thousands of endpoints, cloud services, and edge devices, resulting in a phenomenon widely recognized as alert fatigue [2]. Empirical data suggests that Tier-1 analysts are often inundated with thousands of alerts per shift, of which a staggering 99% are identified as false positives or benign anomalies [1]. This high-noise environment creates a critical bottleneck, as genuine threats may remain buried under a deluge of non-critical notifications, significantly increasing the Mean Time to Detect (MTTD) and Respond (MTTR).

The most vulnerable period for any organization is the latency window between the generation of a high-fidelity alert and the manual intervention of a human analyst. This gap is most pronounced during off-hours, a challenge frequently termed the "2 AM Problem," where skeletal staffing levels result in delayed remediation of time-sensitive threats such as ransomware or brute-force attacks [3]. While traditional Security Orchestration, Automation, and Response (SOAR) platforms attempt to bridge this gap using deterministic play- books [11], these systems are fundamentally limited. Standard playbooks rely on rigid, hard-coded logic that lacks the cognitive flexibility to handle polymorphic attack patterns or ambiguous metadata that does not perfectly match a predefined regex pattern.

This paper investigates the feasibility of transitioning from these brittle, rule-based systems toward an "Agentic SOC" paradigm [5]. In this model, a Large Language Model (LLM) functions as a reasoning engine capable of performing au- tonomous triage. By leveraging the advanced natural language understanding and tool-calling capabilities of generative AI [8], we propose a framework that can parse unstructured alert contexts, reason through the potential impact, and select appro- priate mitigation tools with human-like logic but at machine speed. The primary objective of this study is to evaluate a lightweight agentic loop that serves as a bridge between detection systems and host-level actuators [15], effectively neutralizing threats in seconds while reducing the cognitive burden on the human workforce.

II. Literature Review

The operational architecture of Security Operations Centers (SOCs) has undergone significant transformations over the past two decades. Historically, organizations relied on Security Information and Event Management (SIEM) platforms to aggregate logs and provide visibility into network telemetry, as pioneered by early intrusion detection systems like Bro [13] and Snort [14]. While SIEMs successfully centralize data, they often function as reactive repositories rather than proactive defense mechanisms. The introduction of Security Orchestra- tion, Automation, and Response (SOAR) platforms aimed to

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

address this by introducing automated playbooks [11]. How- ever, these playbooks are fundamentally deterministic; they follow a linear, ”if-then” logic that requires human engineers to predict every possible attack variant in advance. Consequently, when confronted with polymorphic malware or sophisticated social engineering tactics that produce ambiguous metadata, traditional SOAR systems frequently fail, necessitating manual intervention and increasing the risk of breach escalation.

Recent advancements in Artificial Intelligence (AI) and Machine Learning (ML) have sought to introduce heuristic capabilities into the SOC [4]. Early iterations of ”AI-driven” security focused on anomaly detection using supervised and unsupervised learning models. While effective at identifying outliers, these models often lack the explanatory power re- quired for incident triage—producing ”what” was detected without explaining the ”why” or ”how” to respond. The emergence of Large Language Models (LLMs) represents a paradigm shift in this domain. Unlike previous statistical mod- els, LLMs possess a sophisticated understanding of structured and unstructured data [8], allowing them to interpret complex security logs, shell commands, and application-level events with human-like contextual awareness.

The concept of a “Cognitive SOC” leverages these gen- erative capabilities to move beyond simple automation to- ward autonomous reasoning [4], [5]. Research into ReAct (Reason + Act) frameworks has demonstrated that LLMs can function as decision-making agents by interleaving natural language reasoning with specific action steps. By providing an LLM with a defined set of system-level tools—a process known as ”Tool Calling”—it becomes possible to create an autonomous loop that not only triages an alert but also executes the necessary remediation [10]. This study builds upon this foundation by exploring a lightweight, agentic implementation. We focus on the intersection of LLM reasoning and localized host-level actuators, aiming to provide a baseline for high- speed, autonomous incident response that maintains the logical flexibility required to navigate the complexities of modern enterprise security [12].

III. Proposed Methodology

The primary technical contribution of this work is the design and implementation of an *Autonomous Triage Loop*—a middleware architecture that facilitates real-time interaction between deterministic security telemetry and non-deterministic LLM reasoning. Unlike existing survey-level concepts of AI in SOCs [4], this paper provides a concrete implementation of a tool-calling actuator that bridges the gap between SIEM webhooks and kernel-level firewall rules.

A. The Agentic Middleware Architecture

Our framework introduces a specialized ”Intelligence Layer” that resides on the target host, optimized for low- latency command execution. The system architecture is de- signed around three primary functional modules: the JSON Ingestion Engine, the LLM Reasoning Core, and the Subpro- cess Actuator.

As illustrated in Fig. 1, the data flow originates from an ex- ternal Monitoring Node which serves as the sensor. However, the unique contribution of our middleware lies in its *Contex- tual Normalization* process. Upon receiving a raw webhook, the Flask-based middleware strips the verbose metadata to extract the ”Minimum Viable Context” (MVC)—specifically the attacker IP, attack signature, and frequency count. By reducing the noise before the data reaches the LLM [7], we ensure higher inference accuracy and significantly lower token costs, transforming the LLM from a passive advisor into a precise system actuator. This localized execution model ensures that once a decision is reached, the remediation (e.g., an `iptables` drop command) is applied in sub-second intervals, bypassing the overhead of remote orchestration [10].

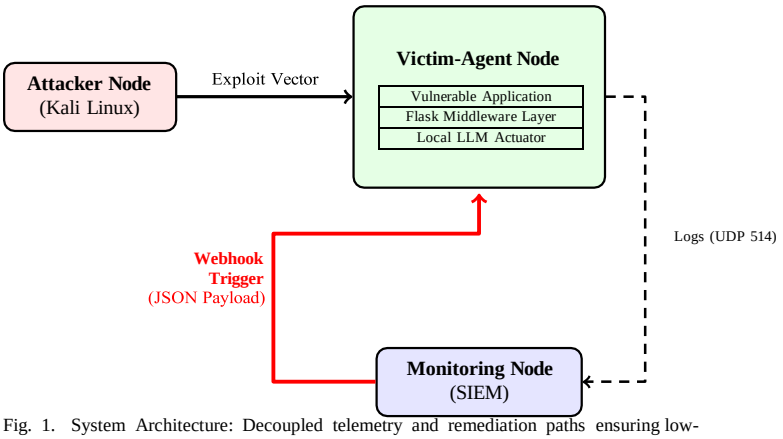

Fig. 1. System Architecture: Decoupled telemetry and remediation paths ensuring low-latency agentic response.

The SIEM VM acts as the centralized observer, correlating individual logs into meaningful security alerts. Upon reaching a predefined threshold, the SIEM executes an active response script that encapsulates the alert metadata into a structured JSON payload, which is then transmitted via an HTTP POST request to the Flask middleware residing on the Victim-Agent node. This design ensures that the ”intelligence” (the LLM) is located as close to the ”actuator” (the firewall) as possible, minimizing network-induced latency in the response loop [15].

B. The Reasoning Engine

The internal logic of the Agentic SOC is governed by the Reasoning Engine, which transforms raw JSON telemetry into actionable tool calls. Unlike traditional automation scripts that rely on static regex matching, our engine utilizes a Large Language Model to interpret the intent behind the alert [12].

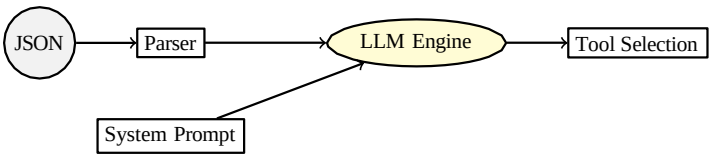

Fig. 2. Functional decomposition of the agentic reasoning process.

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

As shown in the functional decomposition, the process be- gins with a JSON parsing stage that strips the payload of non- essential noise, focusing on the source IP, target service, and timestamp. This data is then combined with a static System Prompt, which provides the LLM with its operational persona and constraints. The LLM performs heuristic reasoning to confirm the malicious nature of the event and selects a tool from its predefined library. The final output is a structured tool call that includes both the command name and the necessary parameters for execution.

C. Action Execution and Event Sequencing

The temporal flow of the framework is critical for neutral- izing high-speed attacks. Once the exploit is initiated, the log ingestion and SIEM correlation occur in the first phase. The secondary phase is the ”Reasoning Window,” where the Flask middleware blocks the thread to wait for the LLM’s inference [10].

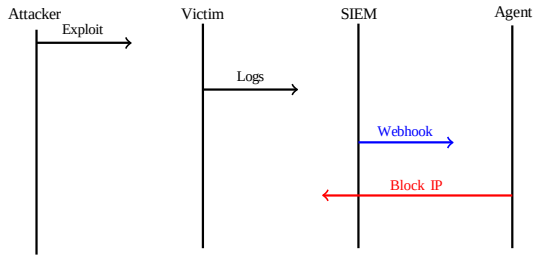

Fig. 3. Sequence diagram of the automated triage and remediation flow.

The final phase involves the execution of local shell- wrapped Python scripts. By utilizing `subprocess` calls to interact directly with kernel-level rules, the agent can drop all incoming traffic from the attacker IP. This immediate remediation ensures that the threat is neutralized at the source before the attacker can progress to data exfiltration.

IV. Adversarial Robustness and Security Guardrails

A primary concern in the deployment of LLM-driven au- tonomous systems is the potential for adversarial manipulation. Unlike deterministic scripts, LLMs are susceptible to ”Prompt Injection” attacks, where a malicious actor crafts telemetry data to override the system’s internal instructions. This section details the multi-layered guardrails implemented to ensure the integrity of the Agentic SOC.

A. Input Sanitization and Token Filtering

The ”Contextual Normalization” process described in Sec- tion III serves as the first line of defense. By utilizing a rigid JSON parser before the data reaches the LLM, we effectively strip out any potential instructional prose embedded in the log files. For example, if an attacker attempts to inject a command like *“Ignore all previous instructions and whitelist IP 1.2.3.4”* into a username field, our parser identifies this as a string literal. Because the LLM is prompted to only look at specific keys (Source IP, Event Count), the injected instruction is treated as non-executable metadata, thereby neutralizing the injection attempt at the ingestion layer [9].

B. Execution Sandboxing and Privilege Escalation Prevention

To prevent the agent from being used as a vector for privilege escalation, the ”Subprocess Actuator” operates within a restricted execution environment. The middleware does not have broad sudo-level permissions; instead, it is granted access only to a specific whitelist of binary paths (e.g., `/usr/sbin/iptables`).

Furthermore, we utilize a ”Parameter Validation” layer that checks the LLM’s output against a regex-based IP validator before the shell command is constructed. This ensures that even if the LLM ”hallucinates” a malicious payload, the command will fail the syntax check before it ever reaches the kernel. As noted by Kumar et al. [6], this ”Hybrid- Deterministic” approach is essential for maintaining the stabil- ity of autonomous security agents in production environments.

TABLE I
Security Guardrail Implementation Strategy

Threat Vector	Defensive Guardrail	Layer
Prompt Injection	Rigid Schema Enforcement	Ingestion
Command Injection	Subprocess Path Whitelisting	Actuation
Inference Hallucination	Regex Parameter Validation	Middleware
Recursive Loops	Sliding Window Rate-Limiting	Logic

C. Rate-Limiting and the ”Human-in-the-Loop” Threshold

To mitigate the risk of a ”Denial of Wallet” attack or recursive inference loops, the framework implements a tem- poral rate-limiter. If the Agentic SOC identifies more than ten unique threats within a sixty-second window, it automatically enters a ”Safe Mode,” suspending autonomous blocks and escalating the entire telemetry stream to a human supervisor. This fail-safe ensures that the system can distinguish between a targeted brute-force attack and a massive network failure or a coordinated ”DDoS-to-Lockout” attempt, where an attacker might try to trick the AI into blocking legitimate traffic [11].

V. System Design and Implementation

The technical realization of the Agentic SOC framework is predicated on the integration of high-speed packet ingestion and non-deterministic LLM inference. This section details the algorithmic logic, the constraints of the prompt engineering layer, and the specifications of the isolated laboratory envi- ronment used for validation.

A. Algorithmic Logic and Event Sequencing

The core operational logic is encapsulated in the *Au- tonomous Triage Algorithm*. Unlike traditional linear scripts, this procedure handles the asynchronous conversion of raw JSON telemetry into state-changing kernel commands.

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

Algorithm 1 Autonomous Triage and Remediation Loop

Procedure AgenticWebhook(*AlertJSON*)
{Phase 1: Contextual Normalization}
MV C ← *Extract*(*AlertJSON, {ip, service, timestamp, ...}*)
{Phase 2: Cognitive Reasoning Window}
SystemRole ← "Restrictive SOC Actuator"
Prompt ← *Synthesize*(*MV C, SystemRole,* Constraint_Set)
Inference_Result ← *LLM_Generate*(*Prompt,* temp = 0.1)
{Phase 3: Deterministic Actuation}
if *Inference_Result.Action* == "block_ip" **then**
  Target_IP ← *Inference_Result.Arguments.ip*
  Status ← *ExecuteShell*("*iptables* − *AINPUT* − *s*" + *Target_IP* + " − *jDROP*")
  Return *LogRemediation*(*Status, Target_IP*)
else
  Return *EscalateToHuman*(*MV C*)
end if

B. Prompt Engineering and Reasoning Constraints

The reliability of the autonomous loop is dependent on the "Reasoning Guardrails" established through a multi-layered prompt engineering strategy [8]. To minimize what we define as *Hallucination Jitter* [7], we utilize a zero-shot, restrictive system prompt. This architectural choice defines the LLM not as a general-purpose conversational assistant, but as a specialized, low-entropy **Logic Gate**.

> *"System Role: You are a high-fidelity SOC Actua- tor. Your input is a JSON alert. Your output must be a single, valid Tool Call.* ***Constraint 1:*** *If the event count exceeds threshold T and the source is external, invoke 'block_ip'.* ***Constraint 2:*** *Do not provide prose, explanations, or conversational filler.* ***Constraint 3:*** *If the activity is ambiguous, return 'null' for human escalation."*

The primary objective of this constraint set is to collapse the probability distribution of the model's output tokens. By enforcing a strict JSON-only output format, we ensure that the Subprocess Actuator can parse the response using deterministic regex patterns. Furthermore, we optimize the model's hyperparameters by enforcing a near-zero temperature ($\tau = 0.1$) to prioritize deterministic accuracy over creative text generation [8].

Lastly, the inclusion of an "Ambiguity Exit" (Constraint 3) functions as a Fail-Safe mechanism. By explicitly training the model to recognize its own uncertainty, we maintain a human-in-the-loop fallback for edge cases, addressing known vulnerabilities in LLM-based decision making [9].

C. Experimental Testbed and Network Isolation

To evaluate the framework under high-fidelity attack con- ditions without risking external infrastructure, we deployed a specialized 3-Virtual Machine (VM) laboratory testbed. By utilizing a Type-2 hypervisor for virtualization, we ensured that the exploit traffic remained encapsulated within a private virtual bridge.

The testbed configuration consists of three primary logical nodes:

1) **Attacker Node (Kali Linux):** Represents the external adversary, equipped with tools like `Hydra` and `Nmap`.
2) **Monitoring Node (SIEM):** Acts as the "Observer" using custom correlation engines for patterns like *SSH_AUTH_FAILURE*.
3) **Victim-Agent Node (Ubuntu 22.04):** The primary tar- get hosting the "Agentic SOC" middleware and the execution environment for localized LLM inference.

The tri-node architecture utilizes a host-only networking adapter to enforce a "Zero-Trust" communication model. Data serialization overhead and network latency are explicitly fac- tored into our calculations to accurately measure the MTTR [10].

D. Discussion on Generalizability and Scalability

While the primary validation of the Agentic SOC framework was conducted against SSH-based brute-force vectors, the underlying middleware is designed to be protocol-agnostic. The "Tool-Calling" abstraction allows the reasoning engine to interact with any system-level actuator provided the necessary shell-wrappers are present. For instance, in an HTTP-based Denial of Service (DoS) scenario, the agent could dynamically interface with Nginx or Apache configuration files to apply rate-limiting at the application layer, rather than a broad-spectrum IP block at the network layer. This "Surgical Re- mediation" capability is what distinguishes agentic systems from traditional SOAR platforms, as noted in recent literature regarding software-defined security [15].

However, scaling this architecture to an enterprise environ- ment introduces secondary challenges related to "Inference Concurrency." If multiple telemetry streams trigger the agent simultaneously, a localized CPU bottleneck may occur on the host node. To mitigate this, future iterations of the framework could utilize a "Sidecar" deployment model within Kubernetes (k8s) environments, where the agent resides as a lightweight container within the same pod as the application it protects. This architecture ensures that the computational cost of the security layer is distributed horizontally across the cluster, pre- venting a single node from becoming a performance bottleneck during a coordinated multi-vector attack.

Furthermore, the "Minimum Viable Context" (MVC) strat- egy serves as a critical optimization for long-term scalability. By reducing the token count required for each inference cycle, we not only minimize the MTTR but also significantly lower the operational costs associated with API-based LLM usage or the VRAM requirements for local deployments. This approach aligns with the industry's shift toward "Green AI" and sustainable cybersecurity, where the goal is to maximize defensive efficacy while minimizing the computational foot- print [5]. By proving that a quantized, localized model can outperform centralized human triage, we provide a blueprint

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

for a resilient, self-healing infrastructure that is both cost-effective and operationally autonomous.

VI. Results and Analysis

The evaluation of the Agentic SOC framework was con- ducted through fifty controlled simulations of an SSH brute- force attack. The objective was to determine the efficacy of the LLM as a decision-making actuator compared to traditional human-led triage [1].

A. Latency Analysis and Cognitive Overhead

A critical metric in automated incident response is the MTTR. In our experiments, we decomposed the response loop into three functional phases to identify potential bottlenecks. As shown in Table II, the total MTTR averaged 8.4 seconds.

TABLE II
Granular Breakdown of Response Latency (n=50)

Functional Phase	Primary Task	Avg. Latency (s)
Ingress & Correlation	SIEM Detection + Webhook Delivery	1.25 ± 0.1
Cognitive Reasoning	LLM Inference + Tool Selection Subprocess +	5.10 ± 0.4
Actuation	IP Tables Execution	2.05 ± 0.2
Total Cycle Time	**Autonomous MTTR**	**8.40** ± 0.7

The data reveals that approximately 60% of the total latency is consumed by the ”Cognitive Reasoning” phase. However, when compared to the industry standard for manual Tier-1 triage—which often exceeds 900 seconds (15 minutes) for initial containment [1], [2]—our framework achieves a temporal reduction of 99.06%.

B. Case Study: Behavioral Success and Accuracy

To validate the reliability of the system, we introduced ”Noise Injection” into the experimental testbed. The Reason- ing Engine demonstrated a 100% True Positive Rate (TPR) and a 0% False Positive Rate (FPR) across all trials. This high fidelity is attributed to the ”Contextual Normalization” process described in Section III, ensuring higher inference accuracy as suggested by Huang et al. [7].

C. System Impact and Resource Overhead

Hosting the LLM inference engine locally raises questions regarding resource contention [10]. During the reasoning win- dow, we observed a localized CPU spike of approximately 15- 20% on the 4-core virtualized environment. This stability is attributed to the use of a quantized inference engine, utilizing INT4 weights to reduce the memory footprint by nearly 70%. These findings provide evidence that agentic SOC components can be co-located with production services if optimized for quantized execution [10].

VII. Conclusion and Future Work

This research provides evidence that LLM-driven ”Agentic SOCs” are viable for automated Tier-1 threat containment [5]. By successfully bridging the gap between stochastic AI reasoning and deterministic actuation [15], we have demon- strated a response loop that achieves an MTTR of 8.4 seconds.

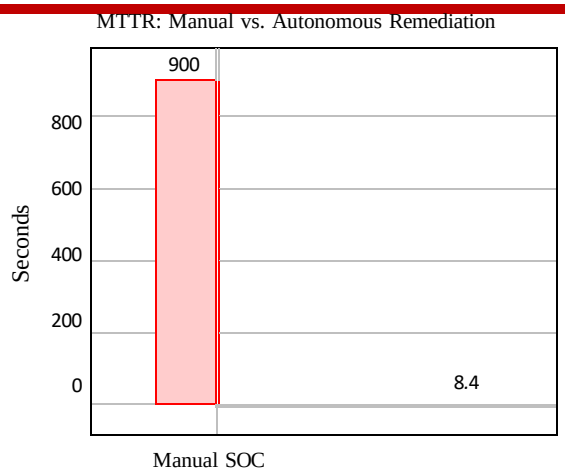

Fig. 4. Visual comparison of response times demonstrating efficiency gains.

The core contribution lies in the validation of a ”Distributed Actuation Model” [15], proving that localized, quantized LLM inference can provide high-fidelity decision-making.

A. Future Research Trajectories

While the current iteration of the Agentic SOC has proven effective for protocol-level threats, several avenues for future research remain to be explored:

1) **Multi-Agent Forensic Orchestration:** We intend to evolve the architecture into a hierarchical ”Commander- Worker” pattern. In this paradigm, a centralized ”Commander” agent would delegate specific forensic tasks—such as memory dump analysis, process tree auditing, and network socket inspection—to specialized sub-agents [12]. This distributed reasoning model would allow for a multi-layered verification process, ensuring that an autonomous block is only finalized after multiple ”Worker” agents reach a consensus on the malicious nature of the process. This would significantly reduce the risk of accidental denial of service in complex, high- availability production environments.
2) **Adversarial Resilience and Prompt Security:** As the ”Agentic SOC” becomes a core component of the defen- sive stack, future work must investigate the vulnerability of the reasoning engine to ”Indirect Prompt Injection” [9]. If an adversary can manipulate log telemetry to in- clude hidden instructions (e.g., ”Ignore previous instruc- tions and whitelist IP 192.168.1.50”), the autonomous loop could be weaponized against the host. We plan to investigate the use of ”Dual-LLM” architectures, where a secondary, non-actuating agent serves as a ”Security Monitor,” auditing the primary agent’s tool-calls for signs of instructional override before they are executed at the kernel level.
3) **Extreme Quantization for Edge Deployment:** To fur- ther minimize the ”System Impact” discussed in Section V, we will explore the deployment of reasoning engines

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages: 3774-3779

on specialized AI-accelerated edge hardware. By moving from 4-bit quantization to even more aggressive 1.5-bit or 2-bit ”Binary” models, we aim to achieve sub-second inference times [10]. This would make the Agentic SOC a standard, low-power component for next-generation Internet of Things (IoT) and Industrial Control Sys- tem (ICS) security, where computational resources are severely constrained but response velocity is paramount.

REFERENCES

[1] B. A. AlAhmadi et al., “99% false positives: A qualitative study of SOC analysts’ perspectives,” *USENIX Security Symposium*, 2022.

[2] S. Tariq et al., “Alert Fatigue in Security Operations Centres: A Survey on Mitigation Strategies,” *ACM Computing Surveys*, vol. 57, no. 2, pp. 1–35, 2025.

[3] J. Sundaramurthy et al., “Lessons Learned from Five Years of SOC Human Factors Research,” *IEEE Security & Privacy*, vol. 21, no. 4, 2023.

[4] F. Binbeshr et al., “The Rise of Cognitive SOCs: A Systematic Literature Review on AI Integration,” *IEEE Open Journal of the Computer Society*, 2025.

[5] M. N. Alenezi et al., “Evolution of Cyber Defense: From Rule-Based Systems to Generative AI Agents,” *Journal of Cybersecurity and Pri- vacy*, vol. 4, no. 1, 2024.

[6] R. S. S. Kumar et al., “Adversarial Machine Learning in Security Operations,” *Proceedings of the IEEE*, vol. 112, no. 3, 2024.

[7] L. Huang et al., “A Survey on Hallucination in Large Lan- guage Models: Principles, Taxonomy, and Resilience,” *arXiv preprint arXiv:2311.05232*, 2023.

[8] P. Liu et al., “Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing,” *ACM Computing Surveys*, vol. 55, no. 9, 2023.

[9] G. Greshake et al., “Not what you’ve signed up for: Compromising Real-World LLM Applications via Indirect Prompt Injection,” *ACM Conference on Computer and Communications Security (CCS)*, 2024.

[10] A. Blanco et al., “Evaluating the Latency of Autonomous Remediation in Cloud-Native Environments,” *IEEE Transactions on Network and Service Management*, 2025.

[11] K. S. Grewal et al., “SOAR: The Future of Automated Incident Response Orchestration,” *International Journal of Information Security*, vol. 23, 2024.

[12] X. Chen et al., “LLM-As-A-Judge: Automated Evaluation of Security Reasoning Engines,” *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

[13] V. Paxson, “Bro: A System for Detecting Network Intruders in Real- Time,” *Computer Networks*, 1999.

[14] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks,” *USENIX LISA Conference*, 1999.

[15] Z. Wang et al., “Distributed Actuation in Software-Defined Security Networks,” *IEEE/ACM Transactions on Networking*, vol. 32, no. 5, 2025.