EAI Endorsed Transactions
on Scalable Information Systems

Research Article EAI.EU

Playbook-Guided Executable SOC Automation for Privacy-Preserving Response in Distributed Networked Systems

Jie Zhang[1], Haizhuang Liu[1,*], Le Ren[1], Yuxiang Zhao[1], Zekai Song[1]

[1]Information and Telecommunications Company, State Grid Shandong Electric Power Company, Jinan 250000, China

Abstract

INTRODUCTION: Distributed networks require security operations center (SOC) automation that connects data security monitoring, privacy-aware evidence handling, controlled execution, and measurable evidence. Static playbooks cannot fully handle ambiguous cross-domain incident context.
OBJECTIVES: This paper presents an executable multi-agent framework for data security monitoring and response in distributed networks.
METHODS: LLM-based roles generate event analysis, tasks, actions, commands, execution records, and summaries. Security orchestration, automation, and response (SOAR) playbooks and a virtual security capability layer provide controlled execution and repeatable evaluation.
RESULTS: On 83 labeled incidents, the framework achieved 0.9684 precision, 0.4742 recall, 0.6367 F1-score, and 76.45 s average handling time for tool-call evaluation.
CONCLUSION: The framework makes distributed data-security response auditable and quantitatively evaluable. The main improvement direction is stronger planning verification for complex multi-step incidents.

Keywords: executable SOC automation, playbook-guided response, privacy-preserving response, distributed networked systems, multi-agent SOC, SOAR orchestration

Received on 08 July 2026; accepted on 03 August 2026; published on 03 September 2026

Copyright © 2026 Jie Zhang *et al.*, licensed to EAI. This is an open access article distributed under the terms of the CC BY-NC-SA 4.0, which permits copying, redistributing, remixing, transformation, and building upon the material in any medium so long as the original work is properly cited.

doi: 10.4108/eetsis.13946

1. Introduction

Distributed networked systems carry business data across endpoints, identities, cloud services, edge devices, and inter-domain communication paths. These environments require continuous data security monitoring because a single incident can involve suspicious access, exposed services, malicious indicators, and potential privacy-sensitive evidence at the same time. Security operations centers (SOCs) are expected to process high-volume alerts, maintain visibility across heterogeneous infrastructure, and respond before threats

*Corresponding author. Email: 953943390@qq.com

propagate. This demand is difficult to satisfy with manual triage alone. Incident-response guidance treats response as a lifecycle that connects preparation, detection, analysis, containment, recovery, and continuous improvement [1]. Meanwhile, breach reporting and alert-fatigue research continue to show analyst workload and limited automation depth as persistent SOC challenges [2, 3].

Security orchestration, automation, and response (SOAR) platforms and response playbooks provide a practical foundation for automation. They codify response steps, connect security products, and reduce repeated manual work [4–6]. However, static playbooks usually assume that an incident has already been classified and that the required response path is known. In distributed data-security scenarios, incidents may arrive as partial descriptions with incomplete context, ambiguous affected scope, uncertain business impact, or unclear data-asset exposure. The system must decide which evidence should be enriched, which capabilities should be invoked, which fields should be minimized, and whether the current result is sufficient for closure.

Large language models (LLMs) and agentic workflows provide a potential solution because they can interpret natural-language incident descriptions, decompose tasks, and invoke tools [7–10]. Yet LLM-based automation is risky if it remains a text-only assistant. An automated data-security response system needs executable commands, auditable state transitions, controlled tool interfaces, privacy-aware evidence boundaries, and measurable evaluation results. Otherwise, model output cannot be trusted as operational response. In this paper, privacy-preserving response refers to operational privacy protection through evidence minimization, controlled access, and auditable tool use, rather than a formal cryptographic or differential-privacy guarantee.

This paper presents an executable multi-agent framework for data security monitoring and playbook-guided response in distributed networks. The framework transforms each incident through a chain of event analysis, task assignment, action planning, command execution, result feedback, and response summary. It combines four LLM-based agent roles, a deterministic executor module, a SOAR playbook layer, a virtual security capability layer, and an asynchronous message-and-state mechanism. The prototype is evaluated through tool-call correctness on 83 labeled security incident samples.

The central problem addressed by the framework is the gap between reasoning output and operational response. Many LLM-based security assistants can generate plausible recommendations, but recommendations are not equivalent to executable data-security actions. A deployable system must decide which tool should be used, represent the decision as a controlled command, execute or defer the command according to risk, preserve intermediate state, and expose evidence for later audit. Without these properties, LLM output remains difficult to verify and difficult to integrate into incident-response workflows.

The framework is therefore designed around response artifacts rather than conversation transcripts. Each stage produces a persistent object: an event, task, action, command, execution record, message, or summary. This design makes the response process inspectable. Analysts can review how an incident was interpreted, why a task was created, which command was generated, what tool result was returned, and why the system moved to closure or another round. This is especially important in distributed data-security operations, where accountability, reproducibility, and controlled handling of sensitive evidence are often as important as automation speed.

The contributions are:

1. An executable multi-agent workflow is designed to convert distributed data-security incident descriptions into structured tasks, actions, commands, execution records, and summaries.

2. A message-and-state mechanism is introduced to persist cross-domain response artifacts and synchronize them to the front-end war-room interface in real time.

3. A virtual security capability layer is used to decouple data-security response evaluation from heterogeneous production devices while preserving observable tool-call behavior.

4. A tool-call evaluation method is applied to quantify response behavior using required, allowed, and forbidden tool annotations.

The remainder of this paper is organized as follows. Section 2 discusses background and motivation. Section 3 presents the framework design. Section 4 describes the message and state mechanism. Section 5 reports the experimental evaluation. The final sections discuss implications, limitations, future work, and conclusions.

2. Background and Motivation

2.1. Data Security Monitoring in Distributed Networks

Traditional SOC automation often begins with rules: if an alert matches a known condition, trigger a predefined action. This model is effective for repetitive and well-bounded tasks, but it struggles when alerts require contextual judgment. In distributed networks, data security monitoring must connect endpoint behavior, network indicators, identity events, service exposure, and business ownership. Incident-response guidance and response playbooks therefore emphasize richer input and output models, where response logic must account for event content, affected assets, expected outputs, and recovery objectives [1, 4].

Playbook-assisted response improves structure by representing procedures as executable workflows. Standards and recent playbook studies emphasize repeatability, reporting, interoperability, recommendation, and lifecycle management [5, 6, 11]. However, playbooks still need upstream reasoning to select the correct workflow, construct parameters, and decide whether additional investigation is needed. This is particularly important when the incident may involve protected data assets because a response workflow should collect enough evidence for security decisions without expanding the evidence scope unnecessarily.

This distinction is important for SOC practice. A playbook can reliably query an asset database or block an IP address once the target is known, but it does not necessarily know whether the incident requires asset enrichment, host alert retrieval, threat-intelligence lookup, firewall control, user notification, data-owner coordination, or a combination of these capabilities. Static automation tends to encode this logic for known cases, while real incidents often contain partial evidence. The role of executable reasoning is to bridge this gap by converting ambiguous incident context into a concrete but controlled response plan.

2.2. LLM Agents under Privacy and Execution Constraints

LLMs are increasingly studied for SOC tasks including log explanation, alert triage, incident summarization, threat intelligence processing, vulnerability analysis, and report generation [7–9]. Multi-agent systems further divide work among specialized roles, reducing the burden on a single prompt and allowing staged reasoning [12–14]. For data-security automation, this specialization is attractive: one role can judge the event, another can assign tasks, another can plan actions, and an executor can constrain tool use and evidence access.

The drawback is that agentic systems may be unstable or unsafe without guardrails. Trustworthy-agent studies identify tool misuse, prompt injection, inconsistent planning, and unsafe autonomy as important risks [15]. This paper therefore treats multi-agent reasoning as only one part of the pipeline. The agents must produce structured outputs that can be persisted, mapped to playbooks, executed through controlled capabilities, and evaluated. The same structure also helps prevent privacy-sensitive evidence from becoming an implicit free-form model context without an auditable reason.

The framework avoids giving agents unrestricted access to external systems. Agents operate on incident context and tool descriptions, then produce structured artifacts. The executor and SOAR layer decide how those artifacts are mapped to commands. Unsupported or risky actions can be recorded as manual-handling requirements. This separation allows LLM reasoning to improve flexibility without removing the operational and privacy boundaries required in a SOC.

2.3. Why Tool-Call Evaluation Matters for Data Security

For a text-only assistant, answer quality may be evaluated by correctness or human preference. For an

executable SOC system, the response process itself must be evaluated. If the system calls an irrelevant containment tool, it may disrupt business operations. If it omits a required enrichment or blocking tool, the response may remain incomplete. If it queries evidence that is not needed, it may increase privacy exposure. Tool-use benchmarks in the LLM community have shown that application programming interface (API) selection and function-call correctness are central to evaluating tool-augmented models [16–18].

This motivates the evaluation strategy used in this paper. Each incident sample contains required tools, allowed tools, and forbidden or irrelevant tools. The system is judged by whether its actual tool calls match this annotation. This process-level evaluation is closer to distributed data-security response quality than measuring only final narrative summaries, because the tool-call trace reveals both omitted capabilities and unnecessary operations.

3. Framework Design

3.1. Workflow Overview

The framework treats each distributed data-security incident as an object that advances through staged processing. The core workflow is:

event input → captain analysis → manager tasking → operator action planning → executor command handling → playbook execution → tool feedback → expert summary.

The workflow is designed around separation of responsibility. Agents do not directly manipulate production systems or unrestricted data sources. Instead, they generate structured artifacts that are later checked, mapped, and executed through the SOAR and capability layers. This keeps model reasoning connected to execution while preserving operational and privacy boundaries.

The workflow is also designed for gradual convergence. A complex incident may require several rounds: the first round identifies affected assets, data exposure clues, and suspicious indicators; the second round retrieves additional endpoint, identity, or threat-intelligence evidence; the third round performs containment or notification after the evidence is sufficient. A one-shot assistant can easily hide this process inside a final narrative. In contrast, the proposed workflow records each round and its generated objects, making both progress and incompleteness visible.

3.2. Agent Roles and Executor Module

The captain agent performs event-level analysis. It receives the incident name, description, context, severity, source, and available tool information. Its output identifies the likely incident type, potential data-security impact, response direction, and high-level handling objective. This role is responsible for global judgment rather than detailed tool execution.

The manager agent decomposes the handling objective into tasks. A task describes a stage-level objective such as verifying an affected host, checking whether a data asset may be exposed, querying source-IP reputation, checking whether containment is required, or creating a notification. This level helps organize the response into manageable units.

The operator agent converts tasks into concrete response actions. An action is closer to execution semantics: query asset information, retrieve threat intelligence, block a malicious source address, create a ticket, or request manual confirmation. For privacy-sensitive evidence, the action object can also indicate why a query is needed and which fields are expected. The operator therefore bridges semantic tasking and concrete capability invocation.

The executor module is a deterministic dispatch component rather than an LLM-based reasoning agent. It transforms actions into commands and dispatches them. If an action can be executed through a playbook, the executor creates a playbook command. If human handling is required, it creates a manual-handling record. The executor also writes execution state and results back into the system. This boundary is important for distributed data security because it prevents free-form model output from becoming direct access to production data or high-impact controls.

The expert agent supports domain analysis and final summarization. It can interpret complex evidence, explain tool outputs, and determine whether the event has reached closure. This role is especially important

when results are incomplete or when a response requires another processing round.

The four LLM-based agent roles and the executor module are not intended to imitate human organizational titles exactly. They are engineering boundaries that separate global judgment, task decomposition, action planning, deterministic command dispatch, and domain interpretation. This separation makes it possible to improve one stage without rewriting the entire system. For example, if the experiment reveals that complex incidents are closed too early, the improvement can target captain prompts, manager task templates, operator action schemas, or executor command mapping depending on where the omission occurs.

3.3. Playbook-Guided Execution

The SOAR layer receives structured commands and maps them to playbook workflows. Asset-related commands are mapped to asset-query workflows; threat-analysis commands are mapped to threat-intelligence workflows; containment commands are mapped to firewall control workflows; and collaboration commands are mapped to notification or ticket workflows. For distributed data-security operations, this mapping makes the path from an ambiguous event to a controlled response explicit.

Playbooks are used for two reasons. First, they define executable order, parameter passing, and dependency handling. Second, they produce observable execution records such as node status, input parameters, return values, and error information. These records are essential for both closed-loop reasoning and quantitative evaluation.

The command layer acts as a boundary between agent reasoning and playbook execution. Commands include the intended capability, parameters, source action, event identifier, and processing round. A command can be executed automatically only if it matches a supported playbook and satisfies execution constraints. These constraints include parameter completeness, supported playbook matching, command-risk level, and whether analyst approval is required. Otherwise, it is preserved as a manual-handling command. This design is deliberately conservative. It prevents an agent from creating arbitrary side effects and gives the system a clear location for future policy checks, such as requiring analyst approval before containment or before querying privacy-sensitive records.

3.4. Virtual Security Capability Layer

Production security devices and data-security platforms are heterogeneous. They differ in authentication, API format, permission model, latency, exception handling, result structure, and data-access policy. Directly binding a research prototype to such devices makes controlled evaluation difficult. The proposed framework therefore introduces a virtual security capability layer.

The virtual layer abstracts common capabilities including asset query, threat-intelligence lookup, firewall blocking, notification, and ticket creation. It accepts structured parameters and returns structured execution results. Although it cannot fully replace real devices, it provides a stable environment for verifying whether the multi-agent system selects suitable capabilities and constructs executable response chains without exposing real data assets during experiments.

The virtual layer is also useful for evaluating tool-call behavior independently of vendor-specific instability. Production tools differ in authentication, permission models, data freshness, rate limits, error formats, and privacy constraints. These differences are important during deployment, but they can obscure whether the decision logic selected the correct capability. By fixing the capability interface during evaluation, the experiment can focus on selection and response-chain construction. After the reasoning and planning logic are improved, the same capability interface can be connected to real products with additional access-control and privacy checks.

3.5. Control Points and Human Oversight

The framework contains several control points. The first control point is the tool-description boundary: agents can only reason over the capabilities exposed to them. The second is the command mapping boundary: an action must be converted into a supported command before automated execution. The third is

the playbook boundary: playbooks define execution order, parameters, and output structure. The fourth is the manual-handling path: actions that cannot be safely executed are preserved for analysts instead of being discarded or executed blindly. These boundaries are also useful for enforcing data-minimization and approval requirements in distributed environments.

Human oversight can be introduced at any of these points. In a conservative deployment, all containment commands may require approval while enrichment commands execute automatically. In a more mature deployment, low-risk actions can be executed directly, while high-impact actions such as firewall blocking, account disabling, or host isolation remain gated. The architecture therefore supports incremental adoption rather than requiring full autonomy from the beginning.

4. Message and State Mechanism

4.1. Persistent Response Objects

The framework represents response progress through persistent objects: event, task, action, command, execution record, message, and summary. An event is the root object. Tasks are generated from the event-level objective. Actions are generated from tasks. Commands are generated from actions. Execution records are produced by command handling and playbook execution. Summaries aggregate the current round and support closure decisions.

Persisting these objects allows the system to support replay, audit, recovery, and evaluation. It also avoids treating the multi-agent conversation as an opaque transcript. Each stage becomes a structured part of the response chain. For distributed data-security response, the persisted objects also provide a place to record why a capability was requested and which evidence scope was implied by the incident.

This representation is particularly useful when an incident is not fully handled. If a required tool is missing, the stored objects show whether the omission occurred before task generation, during action planning, or during command mapping. If a tool was called incorrectly, the execution record can be traced back to the source command and source action. If an unnecessary evidence query is produced, the same chain can reveal which reasoning step expanded the evidence scope. This makes the framework suitable for iterative improvement because failures can be mapped to concrete system stages.

4.2. Asynchronous Delivery

The message flow follows a persist-relay-push pattern. When an agent or executor produces a message, the system first writes it into the database. The message is then published to RabbitMQ. A web-service consumer receives the message and pushes it to the appropriate SocketIO room for the event. The front end receives the update and renders it in the war-room interface.

This design separates durable state from real-time presentation. If a browser disconnects, the response process can continue and the saved messages can later be loaded. If an agent service produces messages faster than the front end can display them, RabbitMQ provides asynchronous buffering.

The asynchronous design also supports multiple agent services running in parallel or on separate hosts. Each role can publish messages and update state without assuming that the front end is available. This matters for distributed SOC workflows because analysts may open, close, or switch event views while the back-end response continues. The persistent database remains the source of truth, while real-time sockets are used only for timely presentation. The design therefore separates operational state from display state, which is useful when response evidence and analyst review must remain traceable.

4.3. Multi-Round Closure

Some incidents can be closed after one round of analysis and execution. Others require several rounds. The framework records a processing round number for tasks, actions, commands, messages, and summaries. After tool feedback is returned, the expert agent performs the final closure judgment using executor-side deterministic state checks. Closure requires all mandatory tasks in the current response plan to reach terminal states, all issued commands to return definitive results or be explicitly deferred, and no

execution failure or pending analyst approval to remain unresolved. If the evidence is insufficient, new tasks can be generated using the latest execution context. This mechanism supports gradual convergence rather than forcing a one-shot response. For data-security incidents, the same mechanism helps distinguish between a case that truly needs more evidence and a case where additional collection would be unnecessary.

Round-based state is also important for evaluation. A tool call made in a later round may be correct even if it was not obvious at the beginning. Conversely, if the system closes an event before enough evidence has been gathered, the evaluation can identify the omission as a planning or verification weakness. The prototype currently evaluates the final set of invoked tools for each sample, but the same state model can support finer-grained future metrics such as first-round progress, additional-round improvement, and closure accuracy.

5. Experimental Evaluation

5.1. Dataset and Labels

The experiment uses 83 labeled security incident samples. Each sample contains an incident title, original message, context, severity, source, and response labels. The response labels are divided into required tools, allowed tools, and forbidden tools. Table 1 summarizes the main fields. In this paper, the samples are used to evaluate whether the framework can transform distributed data-security events into controlled response artifacts; they do not contain real personal records.

The dataset covers advanced persistent threats, backdoor trojans, distributed denial-of-service attacks, phishing, hacker tools, insider threats, mining trojans, software supply-chain risks, vulnerability exploitation, and web attacks. These categories are intended to reflect common SOC response scenarios rather than exhaustive production coverage.

The labels are capability-oriented. A required tool represents a response capability that must be invoked for the case to be considered complete. An allowed tool represents a reasonable auxiliary capability that should not be punished as a false positive. A forbidden tool represents an irrelevant or risky capability. This labeling style is suitable for evaluating executable automation because it checks operational behavior rather than only final text similarity. It is also suitable for data-security response because unnecessary capabilities can represent avoidable evidence access or excessive control action.

The sample context was intentionally kept close to SOC work orders. Some cases provide explicit IP addresses or host identifiers; some describe a vulnerability or attack chain; others provide only a high-level incident narrative and expected response direction. This heterogeneity is useful for testing whether the framework can convert different input forms into comparable response artifacts. A system that works only for one structured alert format would be less useful in a real SOC, where event sources include SIEM rules, endpoint detection products, network traffic analysis, vulnerability intelligence, and analyst-created incidents. For distributed data-security monitoring, such heterogeneity reflects the fact that evidence may be scattered across host, network, application, and identity domains.

Table 2 introduces the categories from the perspective of SOC decision-making. The table intentionally describes the meaning of each subset rather than reporting evaluation scores. The goal is to clarify why these cases are useful for testing an executable automation framework: each category stresses a different kind of evidence, response objective, and operational risk.

The table shows that the dataset is designed around operational reasoning patterns rather than around a single benchmark label. Host-centered categories require endpoint evidence and affected-asset confirmation. Network-centered categories require external indicator analysis and traffic interpretation. Business-context categories require the system to reason about ownership, affected scope, and coordination. This structure is useful for evaluating whether the framework can produce response artifacts that are meaningful across different SOC workflows and different data-security risk patterns.

Table 1. Incident sample fields

Field name	JSON field	Function in the experiment
Sample identifier	`case_id`	Matches input incidents with execution results
Event name	`event_name`	Provides the incident title for event analysis
Original message	`message`	Provides the raw alert or incident description
Context	`context`	Provides assets, indicators of compromise (IOCs), host identifiers, accounts, or business context
Severity	`severity`	Provides risk level information
Source	`source`	Identifies the alert or incident source
Required tools	`required_tools`	Defines tools that should be invoked
Allowed tools	`allowed_tools`	Defines acceptable auxiliary tools
Forbidden tools	`forbidden_tools`	Defines tools that should not be invoked
Case type	`case_type`	Provides scenario category

Table 2. Dataset category semantics and operational meaning

Code	Cases	Scenario meaning	Typical evidence in the sample	Why the category is included
APT	12	Multi-stage intrusion, C2 communication, persistence, or lateral movement	External IPs, affected assets, host identifiers, threat context, abnormal traffic or endpoint behavior	Tests attack-chain decomposition and staged investigation rather than one-step response
BT	3	Backdoor trojan or remote-control behavior on a compromised host	Host process clues, C2 indicators, endpoint alert context, asset ownership	Tests whether the system recognizes host compromise and containment needs
DDoS	2	Availability-oriented network attack against a service or gateway	Source traffic, target service, traffic volume, service impact, severity	Tests the distinction between service-availability response and host-intrusion response
FA	2	Phishing or social-engineering incident affecting users or mail systems	Suspicious sender, URL or attachment clue, recipient context, mail-gateway alert	Tests user/account-oriented triage and notification-oriented response
HT	4	Hacker tool, scanner, exploit tool, or suspicious post-exploitation utility	Tool signature, process name, source host, endpoint or SIEM alert	Tests recognition of suspicious tool usage and follow-up investigation
IT	10	Insider threat, abnormal account behavior, or business misuse scenario	User identity, asset ownership, access behavior, business context	Tests reasoning that depends on identity and business context, not only technical indicators
MC	7	Mining trojan or resource-abuse incident on hosts or servers	High CPU behavior, mining process, wallet or pool clue, host alert, asset context	Tests host-level investigation and remediation planning
SC	21	Software supply-chain risk, vulnerable component, or dependency exposure	Vulnerability notice, affected software, asset scope, patch or exposure context	Tests broad impact assessment and coordination for non-single-host risks
VR	17	Vulnerability exploitation, exploit attempt, or exposed weakness	CVE or vulnerability name, affected service, exploit path, host or network alert	Tests whether the system can connect vulnerability context with response actions
WA	5	Web attack such as webshell, SQL injection, command execution, or reverse shell	HTTP request details, attacker/victim IPs, web path, response status, payload evidence	Tests complex web-attack analysis and attack-path reconstruction

5.2. Experimental Procedure

Each sample was processed through the full framework. The system created an event, invoked the role-based workflow, generated tasks and actions, mapped supported actions to commands, executed commands through playbooks and virtual capabilities, and stored execution records. The resulting predicted tool set was then compared with the required, allowed, and forbidden annotations.

The evaluation is end-to-end. A missing required tool can be caused by several stages: event-level interpretation may be incomplete, task decomposition may omit an investigation objective, action planning may not select the right capability, or command mapping may fail to connect an action to an executable playbook. This makes the evaluation stricter than prompt-only testing, but it better reflects the requirements of an operational SOC system.

5.3. Metrics

Let P denote the set of tools invoked by the system, R denote the required tool set, and A denote the allowed tool set for an incident. The experiment defines:

$$TP = |P \cap R|, \tag{1}$$

$$FP = |P \setminus (R \cup A)|, \tag{2}$$

$$FN = |R \setminus P|. \tag{3}$$

Allowed tools are not counted as false positives because they may be reasonable auxiliary calls even if they are not mandatory. Precision, recall, and F1-score

Table 3. Aggregate evaluation results

Metric	Value
True-positive tool calls	92
False-positive tool calls	3
False-negative tool calls	102
Precision	0.9684
Recall	0.4742
F1-score	0.6367
Average handling time	76.45 s

are computed as:

$$\text{Precision} = \frac{TP}{TP + FP}, \tag{4}$$

$$\text{Recall} = \frac{TP}{TP + FN}, \tag{5}$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{6}$$

The experiment also records average handling time to provide a basic view of runtime feasibility. The timing starts when the event object is created and ends when the final summary is stored. It includes role reasoning, database persistence, RabbitMQ queue waiting and delivery, command generation, playbook execution, virtual-capability execution, and result aggregation, but excludes human waiting time.

5.4. Results

The prototype completed batch processing on all 83 samples. Table 3 presents the aggregate results.

The high precision indicates that the framework strongly constrains unreasonable tool calls. Only three tool calls were counted as false positives under the required/allowed/forbidden labeling scheme. This is important for SOC automation because irrelevant tool execution can generate noise or cause operational side effects.

The recall is much lower than precision. The system missed 102 required tool calls, showing that it often produced conservative but incomplete response plans. This pattern suggests that the current planning process is better at avoiding wrong tools than at covering all necessary tools. Complex incidents requiring multi-step investigation, chained enrichment, or coordinated containment are the most likely cases where required tools are omitted.

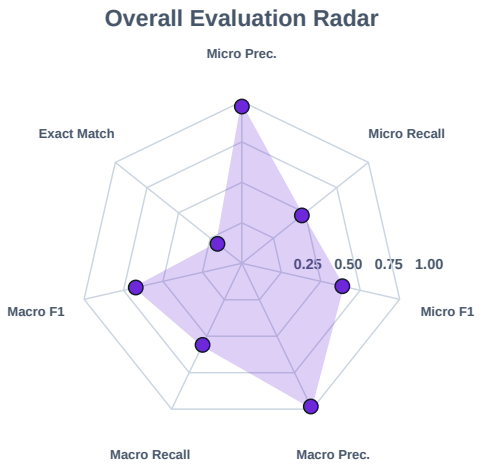


Figure 1. Overall tool-call evaluation metrics for the executable SOC framework

The F1-score of 0.6367 reflects this imbalance. The framework is already executable and measurable, but the response-completeness problem remains open. The average handling time of 76.45 seconds indicates that multi-agent execution is feasible in the current experimental setup, though latency should be re-evaluated when real devices and larger datasets are introduced.

Figure 1 summarizes the overall metrics. The framework achieves high micro precision and macro precision, which indicates that most generated tool calls remain within the expected response space. The exact-match rate is lower because exact match requires the whole response plan to satisfy the case annotation. This distinction is important: the framework can be operationally cautious while still needing stronger completion verification.

The error composition in Figure 2 shows that the framework's weakness is primarily recall. There are only 3 false-positive calls, but there are 102 false-negative calls. In operational terms, the system rarely invokes unreasonable capabilities, but it often stops after a partial response plan. This result matches the framework's guarded design: agents produce structured actions that are constrained by commands and playbooks, but no verification layer yet checks whether the response objective has been fully satisfied.

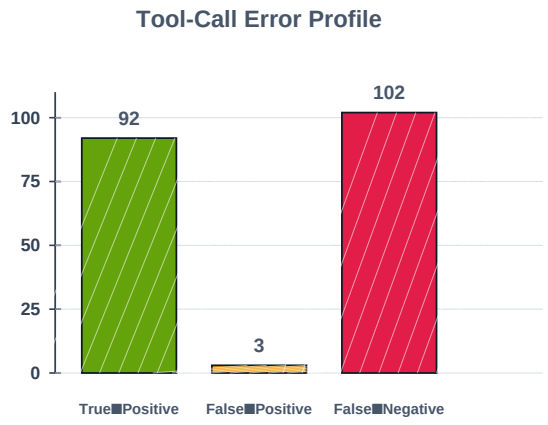


Figure 2. True-positive, false-positive, and false-negative tool-call counts

Figure 3 presents category-level differences. Software supply-chain cases obtain relatively balanced results, while web-attack, APT, and vulnerability-exploitation cases show lower recall. These lower-recall categories often require vulnerability context, cross-source evidence, and follow-up coordination. A single missing reasoning step therefore produces a false negative even when the final textual summary appears reasonable.

The category-level pattern indicates that the system is better at bounded response objectives than at open-ended investigation paths. This does not undermine the executable framework, but it shows where additional control logic should be introduced. A future planning verifier should inspect the generated response plan against the incident objective, affected scope, risk level, and closure condition before execution begins.

The runtime distribution in Figure 4 shows that the framework has a stable prototype-level processing cost. The average handling time is 76.45 seconds, and the median is close to the average. This runtime includes role reasoning, database persistence, RabbitMQ queue waiting and delivery, command generation, playbook execution, virtual tool feedback, and aggregation. For production use, latency can be reduced through model routing, prompt caching, and fast paths for low-risk enrichment-only cases, while high-risk cases can keep the full auditable workflow.

5.5. Ablation Comparison

An additional single-agent ablation run is used to examine the effect of role-specialized orchestration. The comparison uses 83 common case identifiers between the main framework run and the single-agent run. Figure 5 shows that the single-agent workflow obtains higher recall and F1, but at the cost of lower precision and more false-positive calls. The multi-agent framework is more conservative and slower, but it provides stronger control over execution and exposes more intermediate state for audit.

This result does not imply that a single-agent workflow is preferable for SOC deployment. It shows a trade-off between broad generation and controlled execution. The single-agent configuration tends to generate more actions, which improves recall but increases the risk of unnecessary operations. The multi-agent framework should therefore be improved through planning verification and response-completion checks, rather than by removing role separation.

6. Discussion

6.1. Operational Implications

The main value of the framework is that it makes agentic data-security automation operationally inspectable. Instead of producing only a final recommendation, the system exposes intermediate tasks, actions, commands, execution records, and summaries. This is useful for analyst oversight, system debugging, post-incident audit, and review of whether sensitive evidence was handled through controlled interfaces.

The results also show why process-level evaluation is necessary. A final text summary might appear reasonable even when important tools were omitted. Tool-call metrics reveal this weakness directly. In operational SOC settings, similar metrics could be extended to include parameter correctness, result consistency, containment success, analyst approval, and privacy-impact indicators such as unnecessary data-field access.

6.2. Limitations

The current prototype has four main limitations. First, complex incidents still need stronger planning

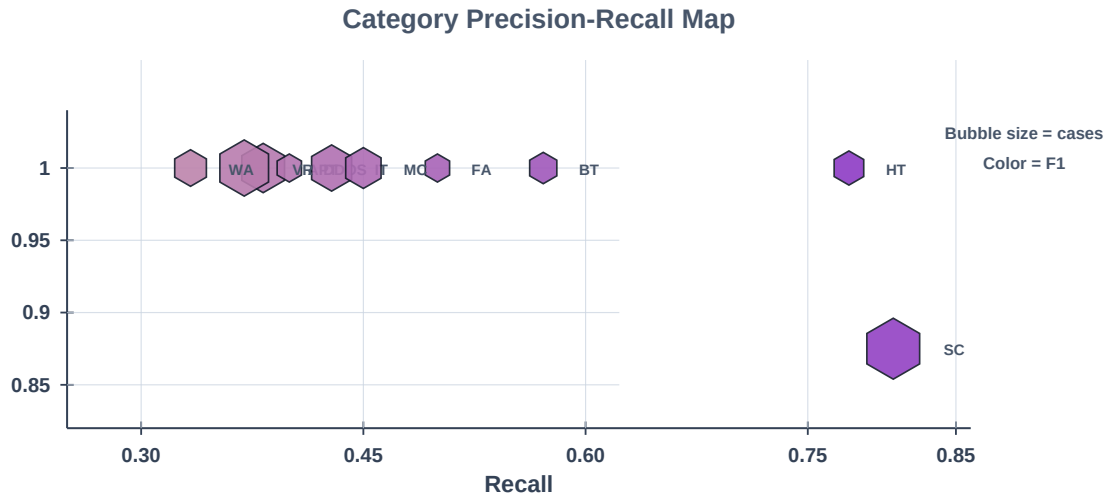


Figure 3. Evaluation metrics grouped by incident category

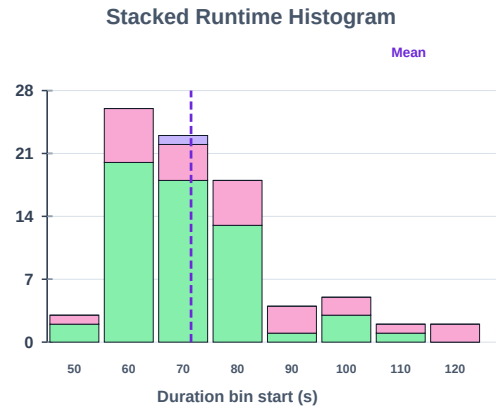


Figure 4. Runtime distribution and runtime by severity group

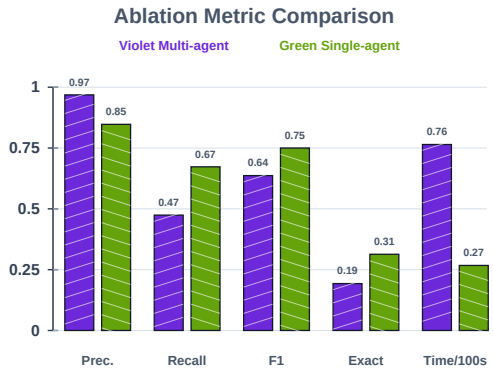


Figure 5. Comparison between the multi-agent framework and a single-agent ablation on common cases
Runtime is normalized by 100 seconds for plotting

verification and response-completion checks, especially when data exposure must be inferred across several systems. Second, the virtual security capability layer simplifies production conditions. Real products include authentication failures, permission constraints, rate limits, unstable output formats, data-access policies, and operational risk. Third, the dataset contains 83 samples, which is useful for prototype evaluation but limited compared with real SOC diversity. Fourth, the current metrics focus on tool selection and do not fully measure parameter quality, response effectiveness, privacy preservation, or business impact.

6.3. Future Work

Future work will add a planning verification stage before playbook execution. This stage can compare generated actions against incident type, data-asset exposure assumptions, scenario templates, closure conditions, and historical response cases. A richer tool schema can also help agents understand preconditions, required parameters, side effects, expected outputs, and privacy constraints.

Another direction is hybrid virtual-real validation. Virtual tools are suitable for repeatable experiments,

while real products are necessary for deployment realism. Combining both environments would support controlled regression tests, privacy-aware access checks, and production-readiness assessment.

Finally, the evaluation framework should be expanded. In addition to precision, recall, and F1-score, future metrics should include parameter accuracy, execution-result consistency, closed-loop completion rate, response latency, manual intervention count, expert judgment, and data-minimization quality.

7. Conclusion

This paper presented an executable multi-agent framework for data security monitoring and playbook-guided response in distributed networks. It integrates LLM-based role collaboration, SOAR playbook orchestration, asynchronous message delivery, persistent state tracking, privacy-aware response boundaries, and virtualized security capabilities. The framework converts security incidents into structured response artifacts and evaluates tool-call behavior quantitatively.

Experiments on 83 labeled incident samples showed 0.9684 precision, 0.4742 recall, 0.6367 F1-score, and 76.45 seconds average handling time. These results demonstrate that the framework can constrain irrelevant tool calls and support measurable automation, while also showing the need for stronger planning verification in complex incidents. The proposed design provides a practical basis for developing distributed data-security response systems that are executable, observable, and evaluable.

Acknowledgements

Funding: This work is supported in part by the Science and Technology Project of State Grid Shandong Electric Power Company: 'Research on Key Technologies of Attack Mechanism Mining, Analysis and Decision-Making for Power System Cross-Domain Security Boundary Protection' under Grant 520627250002.

References

[1] Cichonski P, Millar T, Grance T, Scarfone K. Computer Security Incident Handling Guide. National Institute of Standards and Technology; 2012. Available from: https://doi.org/10.6028/NIST.SP.800-61r2.

[2] Verizon Business. 2026 Data Breach Investigations Report. Verizon Business; 2026. Available from: https://www.verizon.com/business/resources/reports/dbir/.

[3] Tariq S, Chhetri MB, Nepal S, Paris C. Alert Fatigue in Security Operations Centres: Research Challenges and Opportunities. ACM Computing Surveys. 2025;57(9):224.

[4] Cybersecurity and Infrastructure Security Agency. Federal Government Cybersecurity Incident and Vulnerability Response Playbooks. CISA; 2021.

[5] OASIS. CACAO Security Playbooks Version 2.0; 2023. Available from: https://docs.oasis-open.org/cacao/security-playbooks/v2.0/security-playbooks-v2.0.html.

[6] Hu H, Zhang L, Zhang Z, Yao X, Wu X. An Intelligent Playbook Recommendation Algorithm Based on Dynamic Interest Modeling for SOAR. Symmetry. 2025;17(11):1851.

[7] Habibzadeh A, Feyzi F, Atani RE. Large Language Models for Security Operations Centers: A Comprehensive Survey; 2025. arXiv preprint arXiv:2509.10858. Available from: https://arxiv.org/abs/2509.10858.

[8] Jaffal NO, Alkhanafseh M, Mohaisen D. Large Language Models in Cybersecurity: A Survey of Applications, Vulnerabilities, and Defense Techniques; 2025. arXiv preprint arXiv:2507.13629. Available from: https://arxiv.org/abs/2507.13629.

[9] Xu H, et al. Large Language Models for Cyber Security: A Systematic Literature Review. ACM Transactions on Software Engineering and Methodology. 2025.

[10] Xi Z, Chen W, Guo X, et al. The Rise and Potential of Large Language Model Based Agents: A Survey. Science China Information Sciences. 2025;68(2):121101.

[11] Akbari Gurabi M, Fysarakis K, Mavroeidis V, et al. From Legacy to Standard: LLM-Assisted Transformation of Cybersecurity Playbooks into CACAO Format. In: Computer Security. ESORICS 2025 International Workshops. Springer Nature Switzerland; 2026. p. 491-510.

[12] Wu Q, Bansal G, Zhang J, et al.. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework; 2023. arXiv preprint arXiv:2308.08155. Available from: https://arxiv.org/abs/2308.08155.

[13] Yao S, Zhao J, Yu D, et al. ReAct: Synergizing Reasoning and Acting in Language Models. In: International Conference on Learning Representations; 2023. Available from: https://arxiv.org/abs/2210.03629.

[14] Liu Z. Multi-Agent Collaboration in Incident Response with Large Language Models; 2024. arXiv preprint arXiv:2412.00652. Available from: https://arxiv.org/abs/2412.00652.

[15] Yu M, Meng F, Zhou X, Wang S, et al.. A Survey on Trustworthy LLM Agents: Threats and Countermeasures; 2025. arXiv preprint arXiv:2503.09648. Available from: https://arxiv.org/abs/2503.09648.

[16] Li M, Zhao Y, Yu B, et al.. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs; 2023. arXiv preprint arXiv:2304.08244. Available from: https://arxiv.org/abs/2304.08244.

[17] Qin Y, Liang S, Ye Y, et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. In: International Conference on Learning Representations; 2024. Available from: https://arxiv.org/abs/2307.16789.

[18] Wang J, Zhou J, Wen M, et al.. HammerBench: Fine-Grained Function-Calling Evaluation in Real Mobile Device Scenarios; 2024. arXiv preprint arXiv:2412.16516. Available from: https://arxiv.org/abs/2412.16516.