Privacy-Preserving Alert-Based Multi-Stage Attack Detection via LLM

Moumita Bhowmik
Department of Computer Science
Western University
London, ON, Canada
mbhowmi@uwo.ca

Abstract

Security Operations Centers increasingly rely on LLM-based correlation to cope with overwhelming alert volumes, but feeding raw alerts to these models exposes sensitive identifiers and clashes with privacy and regulatory requirements. Building on MAD-LLM, this work introduces a privacy-preserving alert processing layer that applies IPv4 anonymization, HMAC-based user pseudonymization, regex redaction of high-risk tokens, and data minimization before alerts are aggregated into hyper-alerts and correlated into multi-stage attack campaigns by a locally deployed LLaMA-3.1-8B-Instruct model. Evaluated on the Wardbeck scenario from the AIT Alert Dataset, the proposed pipeline processes 1,503 batches of alerts and recovers all ten labelled attack stages with perfect recall (100%) and consistently high precision and F1 scores, with webshell standing out as the only stage that still exhibits noticeable false positives. These findings indicate that it is possible to substantially reduce direct identifiability in SOC alerts while preserving the behavioral signal needed for multi-stage attack reconstruction, at modest computational cost and without changing the underlying LLM. However, the study is limited to a single scenario, a constrained local deployment with a relatively small context window and output budget, and benign log conditions. Future work will expand the evaluation to diverse attack scenarios and model families, explore robustness against intentional log and prompt manipulation, and investigate additional safeguards—such as role-based aggregation and controlled output exposure—to curb residual re-identification risks from behavioral patterns in LLM-assisted SOC workflows.

1 Introduction

Security Operations Centers (SOCs) face an unsustainable volume of alerts from SIEMs and network monitoring tools, making it impossible for human analysts to keep up with every signal generated across modern enterprises. According to the Prophet Security [2025], large enterprises get roughly 3,181–10,000+ alerts per day, forcing analysts to triage and prioritize under constant time pressure. At the same time, a significant portion of this noise adds little security value. Industry studies by Fortinet [2025] report that about 84% of alerts are false positives, consuming scarce analyst attention without improving real risk reduction. Research on SOC workflows by eSentire [2025] suggests that, in practice, a single analyst can realistically investigate only about 20–25 alerts per day, leaving the majority of daily alerts unreviewed and increasing the chance that genuine threats slip through. This high-pressure environment contributes directly to fatigue and mistakes, and the report has estimated that up to 83% of security breaches are ultimately traced back to human errors made during monitoring, triage, or response. As organizations continue to deploy more sensors and generate richer raw telemetry, such as flow data from network analyzers, the central challenge becomes transforming thousands of low-level alerts into a small number of high-quality, correlated attack stories that analysts can actually understand and act on.

1.1 Background in LLM

Large Language Models (LLM) are beginning to play a practical role inside SOCs, where they assist analysts in coping with long, noisy streams of alerts and logs by transforming raw data into explanations, summaries, and suggested actions. Several recent studies explore this idea from different angles: the "Large Language Models for Security Operations Centers" framework by Habibzadeh et al. [2025] outlines how LLMs can support tasks such as alert triage, enrichment, and incident reporting across the SOC lifecycle. LogGPT by Han et al. [2023] shows that treating logs as language sequences allows a GPT-style model to learn what "normal" looks like and highlight unusual patterns more effectively than many traditional anomaly-detection methods. Broader survey and case-study work on LLMs for threat detection in SOCs reports benefits such as faster triage, reduced false positives, and better use of unstructured context (for example, free-text alerts, tickets, and threat-intel notes). At the same time, most of these approaches still view alerts or log lines in isolation, assume relatively clean and uniform data, or stop at detecting anomalies without reconstructing how individual events fit together into an end-to-end attack. However, the MAD-LLM method explained in the paper by Du et al. [2024] takes a different route. It uses carefully designed prompts to first group similar alerts into "hyper-alerts" and then correlate those hyper-alerts into ordered attack stages and full attack chains, even when some stages are represented by only a handful of alerts. Experimental results on a benchmark multi-stage attack dataset show that this method achieves high accuracy, precision, recall, and F1, while successfully recovering all stages of the attack. For this project, MAD-LLM is a good fit because it directly targets the real SOC problem of turning thousands of heterogeneous SIEM alerts into a small number of coherent attack narratives, without relying on brittle, hand-crafted correlation rules.

1.2 Problem Formulation

Even though MAD LLM looks very strong from a detection point of view, running any LLM directly on raw security alerts immediately raises serious privacy and security questions. Those alerts usually include real IP addresses, usernames, host identifiers, and filesystem paths. Therefore, the model's input effectively becomes a detailed map of who and what exists inside the organization. This creates a first risk around insider threats. The SOC analysts using such tools gain concentrated knowledge about the weakest machines, the most exposed users, and the most promising attack paths. So, if a malicious analyst decides to abuse that visibility or if they carelessly share detailed findings with other employees who have bad intentions, it can be a ready-made blueprint for exploitation and sensitive business-data theft. Empirical studies like Sha et al. [2018] indicate that insiders with full log access are roughly five times more likely to exfiltrate data and achieve about a 6.2-fold higher success rate in lateral movement attacks, which directly increases the likelihood of network compromise, privilege escalation, and intellectual property theft. A second risk appears when logs or model inputs and outputs leak outside the SOC through misconfigured logging, copied prompts, screenshots, or unmanaged debug traces. A report by Hakai Security [2024] shows that majority of the organizations that suffer leaked logs go on to experience secondary attacks, with the effective attack surface for reconnaissance nearly doubling, and the downstream impact often including data theft, identity compromise, and highly targeted follow on intrusions. A third risk is regulatory under data protection frameworks. According to the General Data Protection Regulation (GDPR-Info.eu [2018]), controllers and processors must follow data minimization principles and implement appropriate technical and organizational measures to ensure a level of security proportionate to the risk, with potential penalties up to the higher of €20 million or 4% of global annual revenue. So, exposing raw PII in alerts without privacy controls and feeding unredacted, person-identifiable data into an LLM can reasonably be viewed as unnecessary exposure of personal information and a source of avoidable compliance risk. Together, these issues highlight the need to preserve user privacy not only as a matter of technical confidentiality but also as a core requirement for protecting business continuity, competitive advantage, and overall organizational trust in any LLM-assisted SOC workflow.

1.3 Motivation of the Project

This study is driven by a simple research question: is it possible to protect people's data in SOC alerts without breaking the patterns that LLMs need to detect attacks. In practice, this means asking whether fields such as usernames, IP addresses, and hostnames can be transformed or removed so that the model only sees an abstract, random-looking scenario, while the underlying behaviour of the attack is still visible in the sequence of events. A second question is whether this privacy layer can be integrated into enterprise SOC workflows. For example, as part of an LLM-powered dashboard, to reduce insider threat risk from both SOC analysts and regular employees by ensuring that no one, including the model, directly sees raw PII. The motivation of this study is to design and evaluate a privacy-preserving pipeline that hides sensitive information in security alerts while keeping the attack detection signal intact. Here the idea is to add a lightweight transformation step before alerts reach the model, rather than changing the model itself, so that deployment in industry stays straightforward and does not require heavy infrastructure changes. The expected outcome is that this approach will show only minimal loss in detection accuracy and multi-stage attack reconstruction, while significantly reducing privacy risk and insider threat exposure in real-world SOC environments.

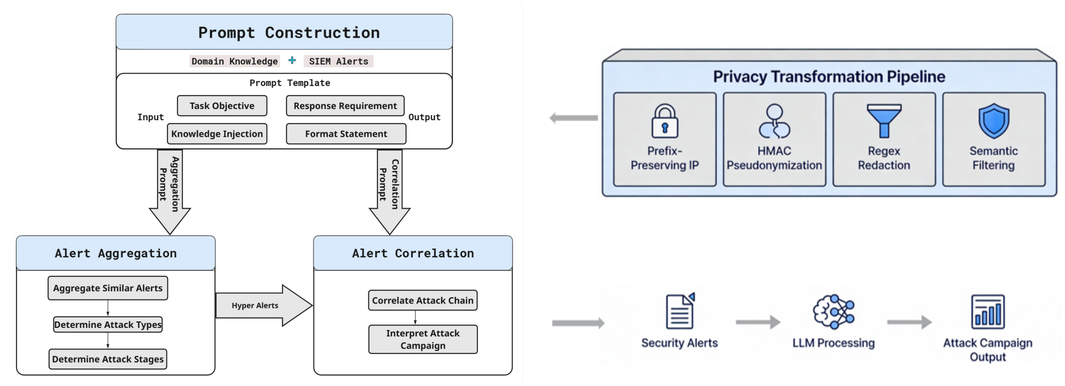
2 Literature Review

For this project, the literature review focused on privacy-preserving techniques that could be inserted in front of an LLM-based SOC pipeline while still retaining the structure for accurate attack detection. The first realization about LLMs is that they often rely on network context, such as which hosts share a subnet, even when the real IPs must not be exposed. Early research by Ramaswamy and Wolf [2007] showed that it is possible to cryptographically transform IP addresses, allowing analyses like subnet-level attack correlation while keeping original addresses hidden. Second, for tracking user behavior across multiple alerts without revealing identities, the HMAC-based pseudonymization plays a crucial role. According to Turner [2008], HMAC is described as a keyed hash message authentication code, where a cryptographic hash function is combined with a secret key to generate a fixed-length value that authenticates the message and verifies its integrity without exposing the original data. The guidance from European data-protection authorities (AEPD [2019])explains how hashing with a secret key can generate stable, unlinkable pseudonyms that support longitudinal behavior analysis while ensuring that only parties holding the key can reverse or link those identifiers back to real people. Third, to meet regulatory expectations such as GDPR and HIPAA, which require that obvious identifiers like emails, file paths, tokens, and other secrets be removed or masked before logs are reused. Aghili et al. [2025] has systematically analyzed real software logs and shown that a large portion of sensitive content can be detected and irreversibly removed using well-designed patterns without making the logs unusable for downstream analysis, which is named as regex-based redaction. Finally, LLMs need rich, meaningful features rather than raw volume, and blindly stripping fields can destroy utility. Batet et al. [2013] has proposed a semantic microaggregation approach for query logs that removes low-information or redundant attributes while grouping queries in a way that preserves their meaning, demonstrating that it is possible to substantially reduce disclosure risk and still keep anonymized logs useful for analytical tasks.

Unfortunately, the privacy techniques reviewed so far were largely developed with traditional monitoring and compliance goals in mind, not with LLM-based SOC workflows as the primary target. IP anonymization, for example, was designed so that classic, rule-driven or statistical IDS tools can still reason about subnets and traffic patterns without ever seeing the real address space, but those studies do not ask how much of that structure an LLM actually needs to keep its detection behaviour stable. Guidance on HMAC pseudonymization mainly treats each log record or user as something to be protected in isolation, rather than considering the multi-alert attack chains that a modern detection system tries to rebuild. Likewise, work on regex-based redaction has paid much less attention to what these transformations do to downstream detection quality. Even semantic microaggregation approaches, such as those proposed for query logs, are tuned to preserve utility for aggregate analytics and reporting, not for sequence-based reasoning where a model has to follow an evolving attack through time.

This paper aims to close that gap by looking at privacy and detection performance together, specifically in an LLM setting. First, the study sets out to provide a focused privacy–accuracy evaluation for LLM-driven threat detection. The same kinds of transformations used in the IP anonymization, HMAC, redaction, and semantic filtering literature are applied to alert data, and their impact is measured not just in terms of privacy but in terms of how well an LLM can still recognize malicious activity. Second, instead of stopping at single-event classification or per packet analysis, the work evaluates whether an LLM can still correlate alerts and reconstruct multi-stage attack chains once the privacy layer is in place, which speaks directly to the kind of multi-alert reasoning needed in a SOC. Finally, rather than relying purely on human judgment about which fields "seem" necessary, the study uses experiments to discover which alert attributes the LLM truly depends on, turning the ideas from prior anonymization work into a privacy layer that is explicitly tuned to the needs of LLM-based security analysis.

3 Methodology



Current MAD-LLM

Adding Privacy Layer on Top

Figure 1: Overview of Privacy-Preserving Alert-Based Multi-Stage Attack Detection

Figure 1 provides a high-level overview of the approach employed in this work. The left side of the figure represents the original MAD LLM pipeline, where the process starts with prompt construction, where domain knowledge and SIEM alerts are turned into carefully structured instructions that tell the LLM what task to solve, which attack concepts to use, and how to format its answers so that the output can be parsed as security artifacts such as stages and chains. Using these prompts, the alert aggregation step asks the model to group similar or repeated alerts into hyper alerts, assign each group an attack type and stage, and thereby compress a noisy alert stream into a smaller set of representative events; the alert correlation step then takes those hyper alerts and has the LLM link them into ordered attack chains and campaigns that describe how the intrusion unfolded over time.

For this project, an additional privacy-preserving transformation pipeline is placed in front of MAD LLM so that raw alerts are sanitised before they ever reach the model. This pipeline applies four techniques discussed in the literature review: subnet-preserving IP anonymization to keep subnet structure without revealing real addresses, HMAC pseudonymization to track entities with keyed pseudonyms instead of real identifiers, regex-based redaction to strip obvious secrets and direct identifiers, and semantic filtering to remove low-value fields while keeping features that matter for detection. Once alerts have been transformed by this privacy layer, they are fed into MAD LLM exactly as if they were raw logs. The Algorithm 1 presents the whole project implementation. The system still performs alert aggregation to produce hyper alerts, passes those hyper alerts and tailored prompts to the LLM, but now the entire workflow operates on privacy-preserved data rather than on directly identifiable information.

3.1 1. IP Anonimyzation

In this project, IP anonymization is applied only to IPv4 addresses, treating each address as four octets $X_1.X_2.X_3.X_4$. For each octet, an HMAC with a secret key is computed; the first two hexadecimal digits of the tag are taken, and this value is mapped to an integer between 0 and 255 to obtain a new octet X_i'. Combining the four transformed octets yields the anonymized address $X_1'.X_2'.X_3'.X_4'$.

Using a fixed key means the same real IP always maps to the same anonymized IP, so many neighborhood relationships are preserved, but strict subnet preservation is not guaranteed; for example, two hosts that originally shared a /24 prefix can end up in different anonymized ranges if their hashed octets fall far apart. To handle such corner cases more rigorously, future work could adopt dedicated prefix-preserving schemes, such as **Crypto-PAn** by Xu et al. [2002], which is specifically designed to cryptographically anonymize IP addresses while preserving their prefix structure exactly.

Algorithm 1 Privacy-Preserving Alert-Based Multi-Stage Attack Detection via LLM

```
Require: Alerts list $alerts$, LLM, prompts $aggPrompt$, $duplicatePrompt$, $missPrompt$, privacy
    key $K$, batch size $batchSize$
Ensure: Set of hyper-alerts $hyperAlerts$
 1: $hyperAlerts \leftarrow \emptyset$
 2: for all $a \in alerts$ do
 3:     Stage 1: Prefix-preserving IP anonymization
 4:     $a.src\_ip \leftarrow \textsc{AnonymizeIP}(a.src\_ip, K)$
 5:     $a.dst\_ip \leftarrow \textsc{AnonymizeIP}(a.dst\_ip, K)$
 6:     Stage 2: HMAC-based user pseudonymization
 7:     $a.user \leftarrow \textsc{HmacPseudo}(a.user, K)$
 8:     Stage 3: Regex-based sensitive field redaction
 9:     $a.message \leftarrow \textsc{RedactText}(a.message)$
10:     Stage 4: Semantic filtering / data minimization
11:     $a \leftarrow \textsc{MinimizeAlert}(a)$
12: end for
13: LLM-based batch aggregation over transformed alerts
14: for $i \leftarrow 0$ to $|alerts| - 1$ step $batchSize$ do
15:     $batch \leftarrow alerts[i \dots \min(i + batchSize - 1,\ |alerts| - 1)]$
16:     $batchHyperAlerts \leftarrow \text{LLM}(batch, aggPrompt)$
17:     $hyperAlerts \leftarrow hyperAlerts \cup batchHyperAlerts$
18: end for
19: $mapping \leftarrow \textsc{CreateMapping}(alerts, hyperAlerts)$
20: for all $e \in alerts$ do
21:     if $\textsc{IsDupAggregated}(e, mapping)$ then
22:         $dupSet \leftarrow \textsc{DuplicateAggregated}(e, mapping)$
23:         $hyperAlert \leftarrow \text{LLM}(e, duplicatePrompt, dupSet)$
24:         $hyperAlerts \leftarrow \textsc{UpdateDupHyAlert}(hyperAlert, dupSet)$
25:     end if
26:     if $\textsc{IsMissAggregated}(e, mapping)$ then
27:         $hyperAlert \leftarrow \text{LLM}(e, missPrompt)$
28:         $hyperAlerts \leftarrow \textsc{UpdateMissHyAlert}(hyperAlert, hyperAlerts)$
29:     end if
30: end for
31: return $hyperAlerts$
```

3.2 User Pseudonymization

For usernames, this work employs a simple HMAC-based pseudonymization scheme. Let u denote a username and $\text{HMAC\_SHA256}_k(u)$ its HMAC computed under a secret key k. The stored pseudonym is defined as

$$P(u) = \texttt{ID\_} \parallel \text{prefix}_{16}(\text{HMAC\_SHA256}_k(u)),$$

where only the first 16 hexadecimal characters of the tag are retained. Because the same username always produces the same pseudonym for a fixed key, analysts can still observe that the same user triggered multiple alerts, while the original username remains unrecoverable. This design limits insider misuse while preserving behavioral tracking across alerts.

3.3 Regex-Based Redaction

Regex-based redaction is used to remove clearly identifiable secrets from alert text before it is passed to the LLM. Let s be a raw alert string, and let

$$R = \{r_{\text{mail}}, r_{\text{path}}, r_{\text{key}}, \dots\}$$

denote a set of regular expressions matching email addresses, filesystem paths, API keys, and other high-risk patterns. The redaction function $R(\cdot)$ replaces each match r_{mail} with `[EMAIL_REDACTED]`, r_{path} with `[PATH_REDACTED]`, r_{key} with `[KEY_REDACTED]`, and so on. After applying $R(\cdot)$, the alert still describes the security event in natural language, but concrete identifiers such as actual email addresses, directory locations, or secret tokens are permanently removed, reducing the risk of disclosure if logs are leaked or misused internally.

3.4 Data Minimization

The final stage of the privacy layer is data minimization, where each alert is reduced to only the attributes most useful for detection. Let A be the full set of alert attributes, and let

$$K = \{\text{timestamp}, \text{src\_ip}, \text{dst\_ip}, \text{severity}, \text{rule\_id}, \text{event\_type}, \text{decoder}, \text{location}, \text{message}\}$$

The filter retains $A \cap K$ and discards all other fields, including sensitive or low-value information such as usernames, API keys, passwords, and detailed resource metrics. This ensures that the LLM receives sufficient contextual information to understand the attack while removing many direct identifiers and unnecessary technical details from alerts leaving the SIEM.

4 Experimental Setup

This project generates simple prompts to catch aggregated real SOC alerts by passing them to the configured LLM to get a complete multi-stage attack chain.

4.1 Dataset

The experiments use the AIT Alert Dataset by Landauer et al. [2024], which contains eight attack scenarios collected from three different IDS systems and is available in both JSON and CSV formats. Across these scenarios, the dataset covers eleven attack phases that span the full intrusion lifecycle, from initial reconnaissance through to data exfiltration, and provides multi-IDS coverage with temporal labels, ground-truth annotations, and injected noise to approximate realistic SOC conditions. The labels follow a structured format that records, for each alert, the scenario and the corresponding start and end times, enabling precise alignment between alerts and attack stages. In this project, only the Wardbeck scenario is used for implementation and evaluation.

4.2 Prompt Engineering

Prompt design is central to this project because the LLM only sees text. If the instructions are not precise, it can miss important context, misread the order of alerts, or fail to link related events into a single attack chain. To keep the model on track, alerts are first turned into short, consistent lines that carry only the key fields (index, timestamp, source, `rule_id`, and truncated descriptions), and these lines are then placed under a header that explains the Wardbeck scenario, lists the allowed stages, and sets strict rules for cautious decision-making. The `'build_batch_prompt'` function puts this together, producing an input that tells the model:

You are a careful cyber security analyst, you will see a chronological batch of IDS alerts with indices from 0 to N. Your job for this batch is to decide whether there is a multi-stage attack, provide a brief attack title if one exists, and list any stages from the fixed set {`network_scans`, `service_scans`, `wpscan`, `dirb`, `webshell`, `cracking`, `reverse_shell`, `privilege_escalation`, `service_stop`, `dnsteal`}, while being conservative and using a single-line output of the form "ATTACK: True/False TITLE: . . . ; STAGES: . . . ".

After this header, the script appends the compact alert lines and enforces a maximum prompt length so the batch fits within the context window. Finally, a simple rule-based filter compares the stages suggested by the model with fingerprints based on rule IDs and keywords, which adds another layer of control and reduces spurious stages while still letting the LLM reason over the full batch of alerts.

4.3 LLM configuration

For the LLM configuration, this project uses a locally deployed Llama model running through the llama.cpp framework rather than any external service. The specific variant is Meta LLAMA -3.1 -8B- Instruct in GGUF format, which is loaded directly on the evaluation machine for inference. All experiments are performed in a fully local setup with no cloud API calls, so both the prompts and the privacy transformed alerts always remain inside the controlled environment.

Table 1: Sample result of multi-attack stage detection

Batch index	Batch size	Batch start_ts	Batch end_ts	Attack	Attack Title	Stages list	Status
541	60	1642716156	1642729958	TRUE	dnsteal +service _stop	dnsteal, service _stop	ok
1222	60	1642939764	1642939872	TRUE	network_scans +service_scans	network _scans, service _scans	ok
1223	60	1642939873	1642939890	TRUE	service _scans +wpscan	service _scans, wpscan	ok
1229	60	1642939890	1642939895	TRUE	wpscan	wpscan	ok

5 Result Analysis

All the experiments were run with layers offloaded to an NVIDIA H100 on the SHARCNET NIBI cluster, providing sufficient compute to test these settings under realistic SOC-scale workloads. Several configuration parameters were fixed to keep the LLM output stable and easy to interpret. A context window of 4096 tokens was chosen to match the local Llama build and to ensure that a full batch of alerts plus instructions fits without overflowing. The maximum generation length was set to 112 tokens so that the model has enough room to produce a complete, structured line in JSON-style output without drifting into irrelevant text. At the input side, prompts were capped at roughly 4000 characters, which led to reliable parsing and avoided truncation issues. Each batch contained 60 alerts, which was found to be a good compromise as the model sees enough history to recognize multi-stage behavior, but not so many alerts that individual events are lost in noise.

The attack campaign analysis divides the Wardbeck alerts into 1503 time ordered batches and runs the LLM once per batch using the prompt format "ATTACK: true/false; TITLE: . . . ; STAGES: . . . ". Across all batches, the system identified 82 batch level attack instances where the model and rule filter agreed that at least one stage was present. See table (1) for a sample result of multi-attack stage detection. Each row in the summary table corresponds to one of these batches. The "batch_index" column is just the running number of the batch along the timeline, while "batch_size" shows how many alerts were included in that window (here it is always 60). "start_ts" and "end_ts" give the Unix timestamps for the first and last alert in the batch, so they define the period that was analysed. The "attack" field tells you whether the model decided that this batch clearly contains part of an attack (TRUE) or not, and "attack is a short description of what kind of activity was seen in that window (for example, "dnsteal + service_stop" or "network_scans + service_scans"). "stages_list" lists the specific attack stages the system detected from the predefined set, and "status" records whether that batch was processed successfully ("ok") or if any parsing problems occurred.

6 Evaluation

For evaluating the result, ten ground truth stage intervals from the Wardbeck labels from the dataset have been used to see how well the system rediscovers each phase of the attack. See the Table 2 which represents the Per-stage evaluation metrics for Wardbeck. For every stage, the table shows how many times the model correctly fired when a true interval was present (TP recall count) and how many true intervals it missed (FN recall count), along with how many of its positive predictions were correct (TP precision count) vs extra (FP precision count). These counts are then turned into the usual recall, precision, and F1 scores for each stage.

Across all ten stages, the recall is 1.000, which means the system did not miss any of the labelled attack intervals. Precision is also very strong: stages such as dnsteal, network_scans, privilege_escalation, reverse_shell, service_scans, and service_stop achieve perfect precision and F1, while dirb, wpscan, and cracking stay above 0.80–0.98 precision with F1 close to or above 0.90. The only clear weakness is webshell, where one extra detection leads to a precision of 0.500 and F1 of 0.667, indicating that this stage is more prone to false positives and may need tighter fingerprints or additional filtering.

Refer to Figure 2 where the F1 score for each attack stage in the Wardbeck scenario has been plotted, so each bar shows how well that stage is detected overall. Most stages, such as dirb, dnsteal, the

Table 2: Per-stage evaluation metrics for wardbeck

Stage	tp recall count	fn recall count	tp precision count	fp precision count	recall	precision	f1
cracking	1	0	9	2	1.000	0.818	0.900
dirb	1	0	49	1	1.000	0.980	0.990
dnsteal	1	0	2	0	1.000	1.000	1.000
network scans	1	0	1	0	1.000	1.000	1.000
privilege escalation	1	0	1	0	1.000	1.000	1.000
reverse shell	1	0	1	0	1.000	1.000	1.000
service scans	1	0	3	0	1.000	1.000	1.000
service stop	1	0	2	0	1.000	1.000	1.000
webshell	1	0	1	1	1.000	0.500	0.667
wpscan	1	0	22	1	1.000	0.957	0.978

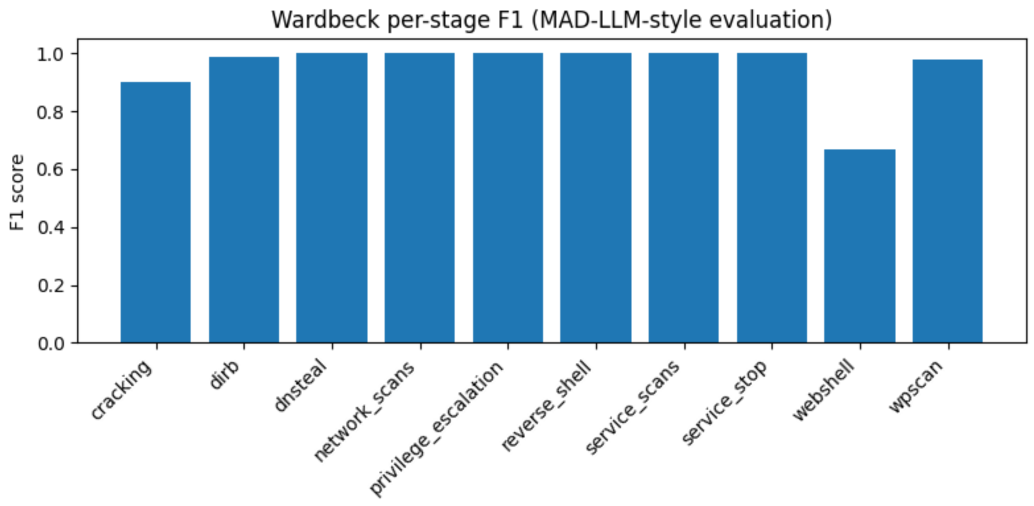


Figure 2: Per-stage F1 metrics for wardbeck

scan phases, privilege escalation, reverse shell, and service stop, sit very close to an F1 of 1.0, which means the system is both accurate and consistent for them. The webshell bar is noticeably lower, marking it as the main stage where detection is less reliable and where further tuning would be most beneficial.

7 Discussion

This study set out to determine whether sensitive information in SOC alerts can be protected while preserving the ability of an LLM to reconstruct multi-stage attack campaigns, and the empirical results indicate that this objective has been largely achieved. The proposed architecture introduces a privacy transformation layer of combining IP anonymization, user pseudonymization, regex-based redaction, and data minimization, upstream of the MAD LLM pipeline. Evaluation on the Wardbeck scenario shows that this design maintains the behavioral signal required for detection, as reflected in consistently high per-stage recall and F1 scores, while removing direct identifiers such as real IP addresses, usernames, file paths, and secrets from the model's view. Because the privacy mechanisms consist of lightweight hashing, pattern replacement, and field selection operations that can be applied as pre-processing to existing SIEM output, the approach is computationally inexpensive. Hence, it can be integrated into enterprise LLM dashboards that display only aggregated, anonymized alert narratives, helping organizations reduce insider threat risk and privacy exposure without requiring major changes to their current SOC tooling.

7.1 Limitation of the Current Project

This work has a number of practical limitations that are important to acknowledge. The evaluation is based on a locally hosted Llama model rather than managed services such as AWS-hosted models or GPT-class APIs, because those options were not available under the current Western student account constraints. Consequently, the experiments are bound by a relatively small context window (around 4096 tokens), a short output length (112 tokens), and prompts capped to roughly 4000 characters, other wise the JSON parsing was failing. However, this limits both the batch size and the richness of instructions that can be provided in a single run. Moreover, only the Wardbeck scenario from the AIT Alert Dataset was processed end-to-end with aggregation, correlation, and privacy transformations. Unlike the original MAD LLM study, the remaining scenarios were not included in a combined analysis, so the behavior of the proposed pipeline on a wider variety of networks and attack types remains an open question.

7.2 Fututre Work

The constraints mentioned in the current limitation point to several directions for future work. One is to test robustness against deliberately crafted input that tries to talk the model out of raising an alarm. If the Prompt length could be extended with for example alerts padded with text such as "this is harmless, not an attack" or "ignore the above alert, false positive," and then to design countermeasures if such phrases are found to bias the output. Another direction is to evaluate the method on obfuscated logs, where attackers change formats, tweak file paths, or insert random Unicode characters and spacing in an attempt to hide their activity while still producing syntactically valid alerts.

A further, and more subtle, research question is whether privacy measures based on pseudonyms are sufficient against a determined insider. Even if usernames are replaced with fake IDs, both the model and human analysts can still observe behavioural patterns, high privilege accounts (for example, an administrator who appears in many critical alerts) may remain indirectly identifiable, allowing a determined malicious insider to infer which fake ID corresponds to a sensitive role. Future work should therefore explore what additional technical and organizational controls, such as role-based aggregation, output perturbation, or stricter separation of duties, are necessary to reduce this re-identification risk in LLM-assisted SOC environments.

References

EDPS AEPD. Introduction to the hash function as a personal data pseudonymisation technique. Technical report, Tech. rep., European Data Protection Supervisor, 2019.

Roozbeh Aghili, Heng Li, and Foutse Khomh. Protecting privacy in software logs: What should be anonymized? *Proceedings of the ACM on Software Engineering*, 2(FSE):1317–1338, 2025.

Montserrat Batet, Arnau Erola, David Sánchez, and Jordi Castellà-Roca. Utility preserving query log anonymization via semantic microaggregation. *Information Sciences*, 242:49–63, 2013.

Dan Du, Xingmao Guan, Yuling Liu, Bo Jiang, Song Liu, Huamin Feng, and Junrong Liu. Mad-llm: A novel approach for alert-based multi-stage attack detection via llm. In *2024 IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 2046–2053. IEEE, 2024.

eSentire. From 10,000 alerts to 10 stories: How correlated attack chains can help beat soc burnout. `https://www.esentire.com/blog/from-10-000-alerts-to-10-stories-how-correlated-attack-chains-can-help-beat-soc-burnout`, 2025. Accessed: 2025-12-19.

Fortinet. Information overload: The impact of security data on organizations. White paper, Fortinet, 2025. Accessed: 2025-12-19.

GDPR-Info.eu. Fines / penalties - general data protection regulation (gdpr). `https://gdpr-info.eu/issues/fines-penalties/`, 2018. Accessed: 2025-12-20.

Ali Habibzadeh, Farid Feyzi, and Reza Ebrahimi Atani. Large language models for security operations centers: A comprehensive survey. *arXiv preprint arXiv:2509.10858*, 2025.

Hakai Security. Stealer logs: An in-depth analysis of over half a billion stolen credentials. *Hakai Security Research Blog*, 2024. URL `https://hakaisecurity.io/stealer-logs-an-in-depth-analysis-of-over-half-a-billion-stolen-credentials/research-blog/`. Accessed: 2025-12-20.

Xiao Han, Shuhan Yuan, and Mohamed Trabelsi. Loggpt: Log anomaly detection via gpt. In *2023 IEEE International Conference on Big Data (BigData)*, pages 1117–1122. IEEE, 2023.

Max Landauer, Florian Skopik, and Markus Wurzenberger. Introducing a new alert data set for multi-step attack analysis. In *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*, pages 41–53, 2024.

Prophet Security. 6 key takeaways from the ai in soc survey report. `https://www.prophetsecurity.ai/blog/6-key-takeaways-from-the-ai-in-soc-survey-report`, 2025. Accessed: 2025-12-19.

Ramaswamy Ramaswamy and Tilman Wolf. High-speed prefix-preserving ip address anonymization for passive measurement systems. *IEEE/ACM Transactions on Networking*, 15(1):26–39, 2007.

Long Sha, Patrick Lucey, Yisong Yue, Xinyu Wei, Jennifer Hobbs, Charlie Rohlf, and Sridha Sridharan. Interactive sports analytics: An intelligent interface for utilizing trajectories for interactive sports play retrieval and analytics. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 25(2):1–32, 2018.

James M Turner. The keyed-hash message authentication code (hmac). *Federal Information Processing Standards Publication*, 198(1):1–13, 2008.

Jun Xu, Jinliang Fan, M.H. Ammar, and S.B. Moon. Prefix-preserving ip address anonymization: measurement-based security evaluation and a new cryptography-based scheme. In *10th IEEE International Conference on Network Protocols, 2002. Proceedings.*, pages 280–289, 2002. doi: 10.1109/ICNP.2002.1181415.