Ca' Foscari
University
of Venice

Master Degree
in Computer Science and Information Technology

Final Thesis

WARDEN: Design and Evaluation of an Agentic AI System for Autonomous SOC Alert Triage

Supervisor
Ch. Prof. Stefano Calzavara

Graduand
Filippo Zane
Matriculation Number 880119

Academic Year
2025 / 2026

To my dad Nico,
who would have been proud
of this moment.

Acknowledgements

I would like to begin this thesis by expressing my heartfelt thanks to my advisor, Professor Stefano Calzavara, for his constant help, guidance, and support during the writing of this thesis and throughout my academic years.

I would like to thank my family — my mother, my brother, my aunts, my cousins, and my grandmothers — for always being by my side and supporting me every step of this journey.

A special thank you goes to Rachele, my source of strength and my safe haven, for supporting me with patience, love, and trust even during the hardest times. I love you.

I would also like to thank my friends — Lorenzo, Davide, Stefano, Isabella, Mattia M., Nicolò, Matteo, Mattia F., Ardi, Giovanni, Aurelo, Federico and Marin — for the friendship you have always shown me.

Finally, I would like to thank all my colleagues — too many to name one by one — for their support, collaboration, and the shared experiences that have accompanied me over the years. Special thanks go to Davide, Alan, Adriano and Nicoletta for the trust and encouragement they have shown me, and for believing in me as I took on new responsibilities.

My greatest thanks go to my dad. I owe him the strength with which I began this journey and the determination with which I was able to complete it. This achievement is his as well.

Abstract

Modern Security Operations Centers face growing volumes of SIEM alerts, increasing the risk of analyst burnout, lower triage quality, and slower incident response. This thesis presents WARDEN, an autonomous AI agent for first-level investigation of alerts generated by Wazuh, an open-source SIEM based on OpenSearch.

WARDEN follows a multi-phase architecture: a deterministic pre-filter based on known false positives, an LLM validator for borderline cases, and a lead investigator agent that performs evidence-driven analysis through playbooks. For high-severity alerts or low-confidence outcomes, a challenger reviews the verdict before it is stored.

The system was evaluated on 60 real alerts collected in production. The deterministic phase resolved 15% of cases in 0.1 seconds on average without token consumption. All 60 investigations required 233 seconds on average and reached a 90% analyst satisfaction rate based on dashboard feedback. The verdict distribution reflects a conservative design: since false negatives can have serious consequences, escalation to human review is treated as a valid outcome.

The results show that agent-based AI can support SOC automation while preserving analyst oversight. The evaluation also highlights limitations: reliance on external LLM providers, limited context awareness in the false-positive knowledge base, and occasional investigation scope expansion. These aspects guide future work, starting with on-premises language model deployment.

Keywords Security Operations, Agentic AI, Large Language Models, Alert Triage.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Over the past decade, organizations have moved critical infrastructure to the cloud, adopted IoT devices at scale, and digitized business processes that used to run on paper. This expansion has dramatically increased the attack surface that defenders must monitor [13, 54].

At the same time, cybercrime has become more scalable and accessible. In particular, ransomware has evolved from relatively simple malware campaigns into a mature criminal ecosystem characterized by *Ransomware-as-a-Service (RaaS)*, double extortion, targeted intrusion paths and supply-chain compromise [16, 15]. The RaaS model has lowered the technical expertise required to conduct ransomware attacks, as affiliates can rent or purchase ready-to-use ransomware products while relying on more experienced operators for development and infrastructure [15]. According to ENISA, ransomware remains one of the most prominent threats to both public and private organizations, with attacks continuing to grow in frequency and impact [16].

Generative AI has also shifted the offensive landscape. Attackers can now use LLMs to write more convincing phishing emails, generate polymorphic malware variants, and search for vulnerabilities [6]. What used to require advanced technical skills can now be scripted in minutes. The *IBM X-Force Threat Intelligence Index 2026* reports that cybercriminals are increasingly exploiting generative AI to scale phishing operations, accelerate malicious code development, and craft more convincing social engineering lures through higher-quality, contextually aware language [21]. Researchers have also documented that large language models (LLMs) can be used to generate targeted spear-phishing emails [20]. As a result, cyberattacks that once required advanced technical skills can now be carried out more easily, lowering the barrier to entry for cybercrime.

Delays in detection and response also have a clear economic impact. According to IBM's 2025 *Cost of a Data Breach Report*, the average cost of a breach is USD 4.4 million [22]. The same report notes that organizations with less mature security programs may take more than eight months, on average, to fully contain an active breach [22].

In this context, this thesis focuses on the challenge of how to support and automate part of the work performed by analysts within *Security Operations Centers (SOCs)* using *agent-based* AI architectures. More specifically, this thesis proposes and evaluates **WARDEN**, a multi-agent system designed to assist with alert triage and incident investigation. The underlying idea is to delegate certain stages of structured analysis to specialized agents, with the aim of reducing the cognitive load on analysts, speeding up response times, and making the alert evaluation process more consistent.

WARDEN should be understood as a conditional multi-agent system rather than as a fully autonomous SOC platform. A primary investigation agent conducts the core analysis, while additional agents are activated only when the alert requires false-positive validation or adversarial review.

Chapter 2 of this thesis provides the necessary context to understand the motivations behind this work. It presents the typical structure of a security operations center (SOC) and describes the time-consuming alarm triage activity, which is often the subject of first-level analysis. Operators are often overwhelmed with numerous alerts and, in order to meet company-defined MTTR (mean time to respond) requirements, they end up underestimating the impact of an alarm or considering it a false positive without first effectively verifying every possible case. This, combined with the fact that many alarms often turn out to be repeated false positives, leads analysts to develop so-called "alert fatigue": a state of cognitive and sensory desensitization that causes operators to ignore or deactivate potentially significant alerts. The methodologies used to date to attempt to automate the first response process are then highlighted, along with their limitations and shortcomings: systems based solely on detection rules are difficult to manage in the long term, while traditional machine-learning systems (supervised and unsupervised learning) often help classify an alarm but do not support the analyst in its comprehensive analysis. To address this very need, thanks also to the rapid evolution and adoption of LLM systems, a growing number of agentive AI systems have been developed and scientifically tested. Chapter 2 concludes by defining the objectives of WARDEN, which led to its development.

Chapter 3 provides a general overview of all the components of the WARDEN architecture. The principles underlying its design are highlighted: evidence-based reasoning, operational efficiency, and traceability. WARDEN is coordinated by a Python dispatcher, which acts as the orchestration layer of the system. After receiving a security alert from the SIEM system in use, it classifies, analyzes, contextualizes it, produces a risk classification (false positive, true positive, or requires human review), and provides a confidence score for its reasoning. In cases where the analysis's confidence level does not allow for direct closure of the case, a second agent, called the *challenger*, is invoked. This agent refutes the first agent's conclusions and attempts to increase or decrease the confidence and classification. To improve overall efficiency and limit the use of tokens, the system also includes a false positive classification system based on deterministic rules defined by analysts. If an exact match is found, the alert is closed immediately. In case of ambiguity in the correspondence, a third AI agent is called to try to understand the actual validity of the exclusion rule and possibly close the alarm.

Chapter 4 describes the concrete implementation of WARDEN and its integration into the corporate monitoring infrastructure. Since the project was developed in a real-world operational context, based on the use of the Wazuh SIEM, the chapter initially introduces the elements necessary to understand its integration: Wazuh's main components, the roles of agents, managers, and indexers, the structure of the indexes used for alerts and inventory, and the system for classifying alarms using rules, severity levels, and associated metadata. It then presents the technologies used in the development of WARDEN and the system's implementation pipeline, consisting of a dispatcher, Python tools for querying OpenSearch and enrichment sources, a knowledge base for managing false positives, a local database for verdict persistence, and a web dashboard for consulting investigations. The chapter also clarifies how WARDEN uses Claude Code as an agentic reasoning component. This choice reflects the constraints present at the time of development, in particular the lack of a sufficiently integrated local enterprise LLM capable of supporting automated investigations.

Chapter 5 describes the experimental evaluation of WARDEN and discusses the results ob-

tained during alert analysis. The goal of the chapter is to verify whether the system is effectively capable of supporting first-level triage efforts, producing coherent and reasoned investigations based on evidence retrieved from the monitored environment. To this end, the scenarios used for the evaluation, the metrics considered, and the system's behavior at different stages of the pipeline are presented: recognition of known false positives, investigative workflow selection, context gathering through queries to Wazuh and enrichment sources, verdict generation, and, in the most critical or uncertain cases, review by a verification agent. The results are analyzed not only in terms of verdict accuracy, but also with respect to operational elements important to a SOC, such as the time required to complete an investigation, the amount of evidence collected, the cost of using the language model, and the possibility of reducing the number of alerts requiring manual intervention. The chapter then illustrates the strengths and limitations that emerged from the experiment, clarifying under which conditions WARDEN can offer concrete support to the analyst and in which cases a human evaluation remains necessary.

Chapter 6 discusses the main limitations of the current prototype and outlines possible future improvements, including the adoption of local LLMs, stronger customer-specific context, and further integration with SOC knowledge bases. Finally, Chapter 7 summarizes the results of the thesis and reflects on the practical role of WARDEN as a support tool for SOC analysts.

In accordance with the Thesis Guidelines of Ca' Foscari University of Venice[1], I declare that I used generative artificial intelligence tools, specifically ChatGPT with GPT-5.5 and Claude Sonnet 4.6, as support during specific phases of this work. Specifically, AI was used to help me translate and revise certain passages in English and to develop the WARDEN web interface. This interface was developed only to make the tool easier for SOC analysts to use, without requiring them to interact with it through terminal commands.

[1]Available at `https://www.unive.it/web/fileadmin/user_upload/ateneo/norme-regolamenti/studenti/Linee_guida_Tesi_def_09-03-2026.pdf`

Chapter 2

Background and related work

This chapter describes the organizational and operational context of Security Operations Centers (Section 2.1), with particular attention to the problem of alert fatigue (Section 2.1.2) and the limitations of existing automation approaches (Section 2.2). Section 2.3 examines the application of LLMs and agent-based AI systems to cybersecurity tasks, placing this work within the broader context of AI-enhanced SOC automation. Section 2.4 explores related systems that have explored similar approaches and Section 2.5 summarizes the existing research contributions and gaps that form the basis of this thesis.

2.1 The Role of the Security Operations Center

Security Operations Centers (SOCs) are organizational units responsible for the continuous monitoring, detection, analysis, and coordination of responses to cybersecurity events. Rather than being defined by a single technology, a SOC should be viewed as a technical and organizational tool that brings together people, processes, and tools to ensure visibility into an organization's digital environment and to support incident management activities. This view is consistent with both academic work on SOC organization and operational guidance on incident response [42, 12, 60].

The exact structure of a SOC varies considerably across organizations. Large enterprises may operate an internal SOC, while smaller organizations often rely on *Managed Security Service Providers (MSSPs)* or hybrid models in which internal teams retain responsibility for risk-related decisions, while external providers offer support and continuous monitoring of the infrastructure. While the structure of a SOC often varies depending on the context in which it operates, its mission remains constant: to transform raw security telemetry data into actionable insights that help identify potential incidents and respond as quickly as possible [42, 60].

From a technical perspective, the central platform in many SOCs is a *Security Information and Event Management (SIEM)* system, which collects, normalizes, correlates, and stores logs and security events from heterogeneous sources such as endpoints, firewalls, servers and cloud services in order to detect and alert on potentially malicious activity. Modern SOC architectures increasingly complement SIEMs with additional tools such as *Security Orchestration, Automation and Response* (SOAR) platforms, *Endpoint Detection and Response* (EDR) solutions, *Network Detection and Response* (NDR) systems, threat intelligence platforms and case-management systems. Together, these systems provide a layered monitoring capability that spans endpoints, network traffic, cloud workloads, and application logs.

2.1.1 Alert triage

Operationally, most SOCs organize around a three-tier model [42, 29]. When a security alert is generated, it is assigned a severity level and placed in a queue, or distributed among SOC analysts according to a predefined dispatching strategy such as round-robin (alerts are distributed cyclically among all analysts) or shortest-queue-first (the alert is assigned to the analyst with the fewest alerts currently being handled).

SOC teams are generally structured into three or more tiers, depending on analysts' experience, background, and seniority:

- *Tier 1.* Serves as the first point of contact and determines whether an alert is a *false positive* or a *true positive*, usually by following playbooks and standard operating procedures. If the alert cannot be resolved at this stage, it is escalated to L2 analysts;
- *Tier 2.* Handles alerts escalated by Tier 1 analysts, performs forensic analysis and carries out the initial incident response activities;
- *Tier 3.* Manages more complex incidents, conducts threat hunting and continuously improves security systems to prevent similar alerts in the future, especially in cases where the alert is flagged as false positive.

Figure 2.1 summarizes this typical SOC structure. Additional roles are included for completeness, although the thesis focuses primarily on the analyst workflow.

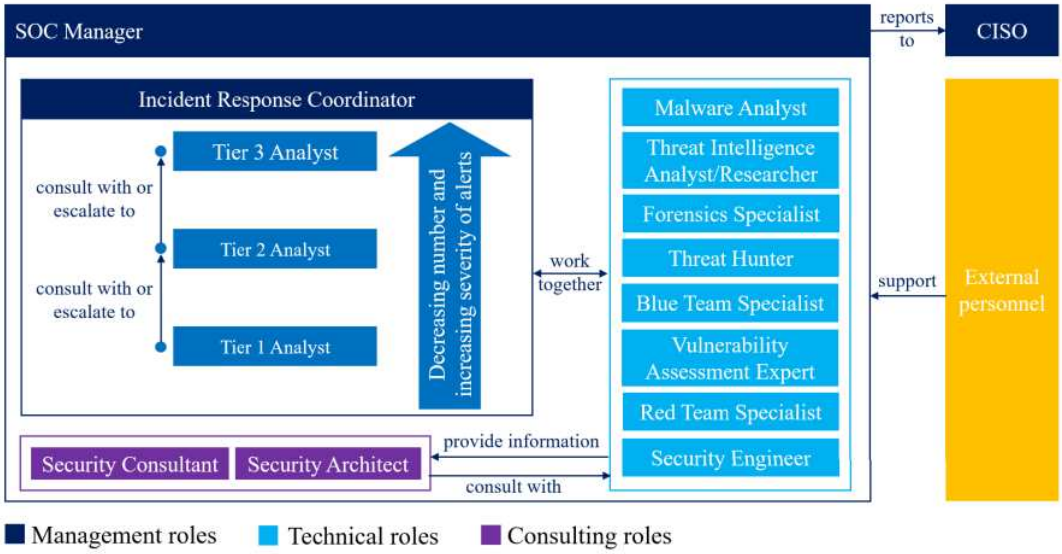


Figure 2.1: Usual SOC organization scheme [42]

This tiered structure helps distribute responsibilities across analysts with different levels of expertise, but it also introduces several limitations.

1. First of all, the initial triage decision is often made under time pressure and with incomplete context. Analysts must combine technical alert data with background knowledge about the user, asset criticality, historical behavior and the customer environment [4].

   As noted by Zhong *et al.* [58], first-line analysts often work under significant time pressure and with incomplete contextual information.

2. Second, the quality of an investigation may depend heavily on the experience and judgment of the analyst handling the case. This can introduce inconsistencies, especially when procedures are incomplete or playbooks do not cover the specific situation. The 2013 Target breach is a clear example of missed alerts: attackers stole about 40 million payment card numbers and 70 million personal records, even though Target's security tools had generated warnings that were not investigated in time [35, 11].

   Playbooks help analysts approach recurring cases in a consistent way. Their usefulness, however, depends on how closely they reflect the environment in which they are applied. An alert that falls outside the documented procedure, or a playbook that has not been updated after an infrastructure change, still leaves the analyst to rely on personal experience and judgment.

3. Third, SOC workflows often require analysts to manually switch between multiple tools, such as SIEM platforms, threat intelligence services, online sources and endpoint systems. This constant context switching increases cognitive load, slows down investigations, and can reduce the quality of decisions.

All of these factors lead to a well-known and well-documented problem among analysts: *alerts fatigue*, a state of mental exhaustion and desensitization caused by an excessive number of alerts, many of which are low-priority or false positives.

2.1.2 Alert fatigue

A Trend Micro survey [41] found that 54% of SOC teams feel overwhelmed by alerts, and security experts spend 27% of their time handling false positives. Alahamdi *et al.* [4] conducted a qualitative study of SOC analysts' perspectives on security alerts across multiple organizations and they found that the vast majority of alerts, approximately 99%, are perceived by analysts as false positives. This finding is consistent with earlier quantitative studies and industry reports that documented false positive rates ranging from 40% to over 90% depending on the detection technology and the overall organizational maturity [39].

The nature of these false positives varies considerably: some stem from legitimate operational activities that mistakenly match threat signatures, while others result from incorrectly configured detection rules or insufficient contextual understanding. Chhetri *et al.* [10] have documented several representative scenarios observed in production SOC environments, as illustrated in Table 2.1.

Tariq *et al.* [39] break down alert fatigue into four root causes, which directly inform WARDEN's design:

- *Staff and skill shortages.* SOC teams often lack sufficient staff with the technical skills needed to perform accurate triage. This problem is aggravated by high turnover and the difficulty of training new analysts on the industry-specific knowledge required for effective incident triage [38].

  The shortage of cybersecurity personnel is a widely recognized challenge. According to the *Cybersecurity Workforce Study 2023* conducted by ISC^2, the global shortage of cybersecurity personnel stands at approximately 4 million professionals, with organizations reporting difficulties in recruiting and retaining qualified SOC analysts [23].

Scenario	Example (Severity)
Misconfigured Security Rule	A firewall rule intended to block malicious traffic from a specific region accidentally blocks all traffic from that region, impacting legitimate business partners (High).
IDS False Positive	An IDS flags a large file transfer between two trusted servers as a potential data exfiltration attempt due to the size of the data, even though it is a scheduled software update (Medium).
Malware Detection False Positive	A malware detection tool identifies a legitimate system utility as malware due to a shared code snippet with a known malware sample (Low).
Phishing E-mail False Positive	An e-mail security tool identifies an internal training e-mail containing a simulated phishing link as a real phishing attempt (Low).
Denial-of-Service (DoS) Attack False Alarm	A network monitoring tool misinterprets a sudden surge in legitimate traffic, e.g., a flash sale on a company website, as a DoS attack, triggering an unnecessary investigation (Medium).
Anomaly Detection False Positive	An anomaly detection system flags a user who is downloading a large dataset for a data analysis project as suspicious due to their deviation from usual activity (Low).

Table 2.1: Common false positive scenarios in SOC operations [10]

- *High false-positive rates.* Detection systems are often configured to prioritize detection rates over accuracy, in order to reduce the number of actual attacks that slip through the system. The obvious trade-off is that a greater number of false positives reach the SOC, which may consequently have difficulty handling them.

- *Disconnected tools and overloaded dashboards.* Even a simple, routine alert often requires the analyst to consult several systems, including the SIEM, an endpoint security platform, threat intelligence services, and the asset inventory. Each tool provides a different view of the same event, while direct integration between them is often limited.

  These platforms also run on separate servers and use different URLs, credentials, and interfaces. The analyst must therefore move between several browser windows and manually combine the retrieved information.

  As a result, the analyst reconstructs the case by linking users, hosts, indicators, and timestamps across multiple sources. This repeated context switching increases investigation time and makes the process more prone to errors.

- *Rigid operating procedures.* Many SOCs rely on predefined playbooks that do not always adapt well to the variability of real-world alerts [60]. In some cases, analysts still have to follow several manual steps even when the outcome is already clear. This rigidity can waste time and, in less straightforward cases, may also cause relevant signals to be missed.

Alert fatigue is not limited to a general feeling of overload. It affects how much attention analysts can devote to individual cases and makes it harder to apply the same level of precision throughout a shift. The result may be an inconsistent triage process in which similar alerts receive different treatment or relevant evidence are noticed only after an escalation.

The 2023 Tines survey reported that 63% of security professionals experienced some degree of burnout and more than 80% had seen their workload increase during the previous year [40]. For a SOC analyst, this often means examining a growing queue of alerts while trying to determine which to prioritize and to which a deeper investigation is needed. Under these conditions, a genuine incident can remain hidden among a much larger number of routine or low-value events.

Efforts to address alert fatigue have taken several forms:

- *Alert tuning.* The manual refinement of detection rules in order to reduce false positives. It's a common practice but requires significant analyst time, domain expertise, and continuous maintenance as the threat landscape evolves.

- *Alert prioritization.* Systems that leverage machine learning to score and rank alerts by severity and likelihood of being a true positive have shown promise in reducing cognitive load on Tier 1 analysts [59].

However, these approaches typically operate as filtering mechanisms rather than full analytical agents, and they do not address the need for contextual investigation and reasoning that characterizes expert-level alert triage.

2.2 Limitations of Current Automation Approaches

The automation tools already present in the SOC ecosystem (including SOAR playbooks, rule-based filtering and machine learning-based anomaly detection) have significantly improved efficiency in certain well-defined scenarios. SOAR platforms, for instance, excel at executing predefined, deterministic workflows: enriching alerts with threat intelligence lookups, blocking known-malicious IP addresses or creating tickets in case-management systems [39]. However, these systems operates within rigid boundaries and struggle when confronted with novel attack patterns, ambiguous evidence, or scenarios that require multi-step reasoning and contextual judgment.

2.2.1 The brittleness of Rule-Based systems

The dominant paradigm for SOC automation remains rule-based detection and response. Detection rules are deterministic checks that define when an event should become an alert.

For example, a rule may trigger when the same IP address causes five failed login attempts in one minute. Figure 2.2 illustrates this detection logic: individual failed login events are collected on a timeline, evaluated by the detection engine[1] against predefined rules, and elevated to a brute-force alert when thresholds are met.

Response playbooks then describe what should happen after a specific type of alert appears. For a brute-force alert, for example, the playbook may include steps such as: block the offending IP address, check if any login attempts succeeded, review account activity for signs of compromise and notify the security team.

[1] A detection engine is a key element of cybersecurity and system security that collects raw data, evaluates them based on predefined rules or behavioral patterns, and triggers alerts or automated responses when it detects anomalies, known malware signatures, or suspicious activity.

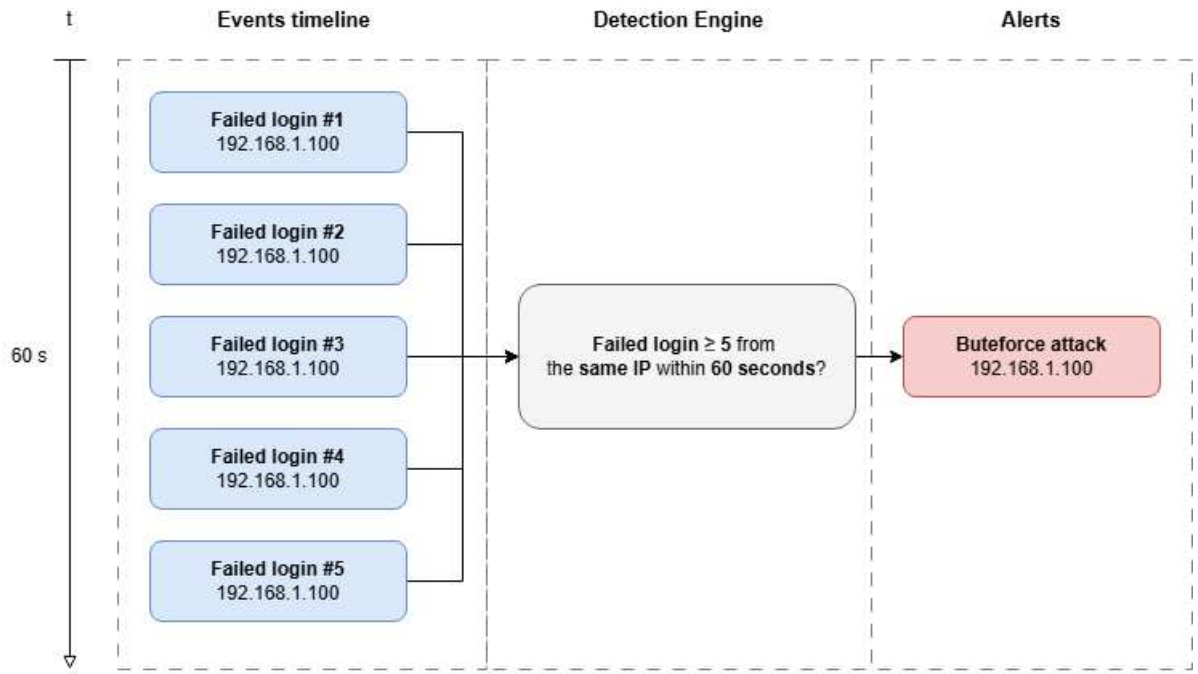


Figure 2.2: Flowchart of a brute-force attack

This approach is effective for cases that can be expressed through stable and well-understood conditions. Its limitations become more visible when those conditions change or when the same observable can have different meanings depending on the surrounding activities.

Generally speaking, the most important limitations of rule-based systems are:

- *Maintenance over time.* Detection logic cannot be always treated statically. A rule that was accurate when introduced may become noisy after a software deployment, a network configuration or a change in normal user activity.

  Furthermore, in the context of a production SOC with hundreds of clients being monitored simultaneously, the behavior of a specific network component may be detected as malicious in one context, while in another it may not be.

  Keeping the rules useful therefore requires continuous work from detection engineers and experienced analysts. They must inspect recurring false positives, understand whether the underlying behavior has changed, and decide whether the rule should be narrowed, extended, or replaced.

- *Expert knowledge is usually hard to turn into rules.* Experienced analysts do not only follow fixed conditions. They build judgment over time and, when they notice unusual combinations of events, understand the meaning of small contextual details.

  Much of this knowledge is usually hard to write down a simple *"if this, then that"* logic. A rule-based system can capture some of this reasoning, but usually only the clearest parts of an attack. This leaves a gap between what a rule can detect and what an expert analyst might recognize.

- *Rules may become hard to manage over time.* As SOC teams try to reduce false positives, rules often gain exceptions and special cases. A rule may start simple but after many changes the detection logic may become difficult to understand and debug.

- *Rule-based systems react slowly to new threats.* Attackers constantly change their techniques to avoid detection and the system can miss the malicious activity until a new rule or signature is created, tested and deployed.

These limitations have made it necessary to develop autonomous systems that can help operators respond effectively and maintain the monitoring environment. Given recent advancements in artificial intelligence, machine learning systems and large language models (LLMs) have been implemented.

2.2.2 Machine Learning-Based approaches

Machine learning has been widely explored as a complement to rule-based detection in SOC environments. Instead of requiring analysts to define every condition that should trigger an alert, supervised models learn from previously labelled examples to distinguish benign from malicious activity. Anomaly detection systems follow a different approach and identify behaviors that deviate from an expected baseline [25].

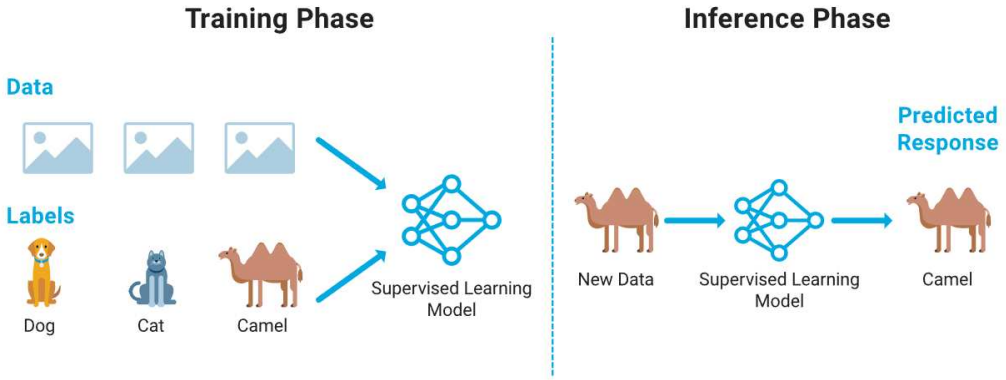


Figure 2.3: The training and inference phases in supervised learning [26].

The availability of historical alerts makes machine learning a natural option for alert filtering and prioritization [59]. A model learns which combinations of features have previously been associated with malicious activity and uses this information to rank new alerts. This supports the initial triage process, but covers only part of the analyst's work. A score does not indicate which log source should be queried next, whether the observed activity is expected for a specific user, or whether a suspicious event led to a successful compromise.

In practice, machine learning does not remove the main difficulties of alert triage:

- Most ML-based systems operate as classifiers or ranking mechanisms. They assign a score, label, or priority to an alert, but rarely identify which evidence the analyst should inspect next. They also provide limited information about the assumptions behind the decision or the alert's relevance within the specific organizational environment.

  This limits their use in SOC operations, where analysts need enough evidence to justify closing, escalating, or responding to an alert.

- Supervised learning depends heavily on labelled data. In a SOC, producing reliable labels requires analyst time and domain expertise. Similar alerts may also receive different

labels depending on the available context, the analyst's experience, or the operational conditions at the time[2].

Building a suitable training dataset is therefore slow and expensive. Models trained on past alerts may also lose accuracy when users, systems, policies, or attacker behavior change.

- Unsupervised anomaly detection reduces the dependence on labelled data, but introduces a different problem. An unusual event is not necessarily malicious. Al Jallad *et al.* note that anomaly-based intrusion detection systems often produce high false-positive rates because previously unseen patterns may represent legitimate activity absent from the training data [3].

  Without organizational context, anomaly detection still produces alerts that require manual review. It helps identify deviations, but leaves the analyst responsible for deciding whether those deviations represent a threat.

ML-based approaches therefore provide useful support for SOC triage. They help rank alerts and detect unusual patterns, but do not reproduce the contextual and evidence-based reasoning required during a full investigation.

2.2.3 From Classification to Contextual Reasoning

The limitations discussed in Section 2.2.1 and Section 2.2.2 show that SOC triage should not be treated only as a classification problem. A rule-based system determines whether a predefined condition is met and, if that's the case, an alert with a predefined severity is triggered. A machine learning model, instead, estimates whether an event resembles previously observed malicious or benign examples. Both approaches support detection and prioritization, but they represent only part of a real investigation activity.

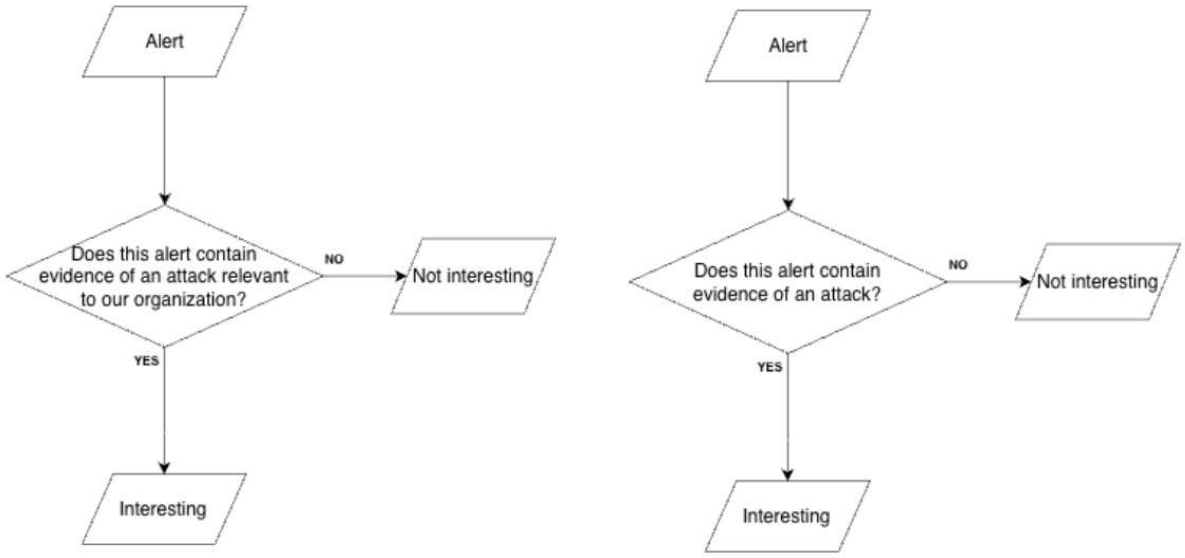
When analysts evaluate an alert, they do more than simply assigning it a label or a score: they determine whether the observed activity is relevant within the specific organizational environment [33]. For example, a failed login becomes concerning when it originates from an unusual country or IP address, targets a privileged account, or occurs alongside repeated authentication failures. Similarly, a newly observed process or a modification to the Domain Admins group[3] requires information about the affected host, user, timing, and expected administrative activity. Tariq *et al.* note that analysts rely on both system knowledge and experience to decide whether an alert is legitimate or a false positive. [39].

Figure 2.4 illustrates this difference. Figure 2.4b shows a workflow based only on the technical information contained in the alert. Figure 2.4a also considers organizational knowledge, including user roles, expected behavior, schedules, policies, and system characteristics. This additional information helps determine whether the same technical event is expected or suspicious in its specific setting.

Recent work on context-aware alert classification supports this view. Oniagbi shows that LLM agents can support Tier 1 triage by classifying alerts and producing natural-language explanations, although their results remain dependent on prompt design, available context, and human

[2]This limitation is not specific to SOCs. Data collection and labeling are widely recognized as major bottlenecks in machine learning, especially when reliable labels require expert knowledge [32].

[3]The Domain Admin group is the highest-privilege standard group in a Windows Active Directory (AD) environment. Members of this group have almost near-unrestricted control over the entire domain.



(a) With context

(b) Without context

Figure 2.4: Workflow for classifying alerts with and without context [33]

supervision [28]. Bono *et al.* report that the adoption of generative AI in live SOC operations is associated with a reduction in mean time to resolution, indicating improved analyst productivity when these systems form part of existing operational workflows [8]. More recent experiments further show that adding structured organizational context to LLM-based alert classification reduces false positives and improves consistency across repeated executions [33].

Contextual reasoning therefore represents an important limitation of current SOC automation. Effective triage requires more than predicting a class. It requires gathering evidence, connecting related events, interpreting them within the organizational environment, and producing a reviewable explanation. Agentic AI systems address this need by combining tool interaction, evidence retrieval, and structured reasoning within the investigation process [34].

2.3 LLMs and Agentic AI in Cybersecurity

Large language models (LLMs) are part of the broader family of generative AI systems. Simply put, generative AI refers to models capable of producing new content (such as text, images, audio, or video) based on patterns learned from training data. LLMs apply this concept to language: they generate text by predicting and composing sequences of words that are consistent with the input they receive.

Many modern LLMs are also described as *foundation models*. A foundation model is a large-scale model trained on extensive and diverse data, so that it can be adapted to many different tasks rather than being built for a single specific purpose [55]. For example, the same model can summarize a report, answer questions, write code, explain a warning, or help structure an investigation. This flexibility is one of the reasons why LLMs have attracted attention in the field of cybersecurity.

In the field of security, models such as GPT-4 and Claude have demonstrated the ability to interpret unstructured text, generate context-aware explanations, and support multi-step reasoning. These capabilities make them useful for tasks that are difficult to express as hard-coded rules,

such as summarizing threat intelligence, assisting analysts during alert evaluation, generating queries, or producing incident reports.

One of the main limitations of these models is *hallucinations*. In the context of generative AI, a hallucination refers to the generation of content that appears plausible but is not based on factual or accurate information. This can include fabricated details, misleading statements, or explanations that seem coherent but cannot be verified in light of the available evidence [31]. In the field of cybersecurity, this problem is particularly relevant because an incorrect but convincing explanation can lead an analyst down the wrong investigative path.

2.3.1 The Evolution of Agentic AI

Vinay [43] proposes a five-generation taxonomy that traces the architectural evolution of agent-based AI systems in the field of cybersecurity, from early single-model LLM-based reasoners to fully autonomous pipelines.

- *Generation 1: single LLM reasoners.* The first group includes systems in which the model receives a prompt and returns a textual answer without interacting with the monitored environment. Such a model can summarize an alert, explain a technical report, or reorganize information already included in the prompt. Any fact that is not provided in the input, however, remains outside the investigation. The model cannot check the SIEM for related events, inspect the current state of an endpoint, or verify an indicator against an external service.

  In SOC context, Generation 1 systems can help analysts understand complex threat reports or draft incident summaries, but they cannot investigate alerts or interact with security infrastructure.

- *Generation 2: tool-augmented agents.* Generation 2 systems extend LLMs with the ability to interact with external tools through structured APIs. These tools support tasks such as querying security platforms, consulting threat intelligence feeds, invoking analysis services, and running sandbox-based checks.

  Architectures such as *ReAct* enable the model to reason about which tool to call, observe the result and decide on the next step to perform in an iterative loop [57].

  This capability can significantly expands the practical utility of LLMs since they can now be used to create independent agents that can autonomously decide how to investigate a case, validate hypotheses and automate multi-step investigative workflows that were previously manual.

- *Generation 3: multi-agent systems.* In these architectures, the investigation is divided between agents with different responsibilities. One agent may decide which workflow applies, while others retrieve evidence or review the resulting conclusion. The distinction is not simply that several model calls are made: each agent receives a narrower task and passes a structured result to the next component.

  For example, one agent might focus on behavioral analysis (to identify suspicious patterns in logs) while another agent gathers evidence querying external systems and a third one synthesizes findings into a final decision.

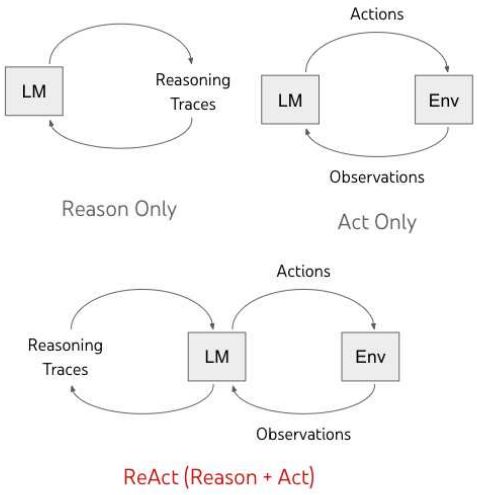


Figure 2.5: ReAct reasoning loop [56]

This decision somehow mirrors the way human SOC teams operate, with different analysts bringing different expertise to bear on a problem.

Multi-agent systems have been shown to improve both accuracy and transparency compared to single-agent baselines, as each agent's reasoning can be inspected independently [53, 14].

- *Generation 4: MCP-based and standardized tool ecosystem.* Generation 4 systems introduce standardized interfaces for communication between agents and external tools. Technologies such as the Model Context Protocol (MCP) and structured function calling define how agents access services, exchange data, and invoke operations. This improves interoperability, access control, reproducibility, and auditability.

- *Generation 5: autonomous pipelines.* These systems extend the agent beyond a single analytical task. The same pipeline may receive a detection, collect supporting data, determine a likely cause and prepare or execute a response. The main architectural change is therefore the amount of the operational workflow placed under AI control.

Despite their potential, autonomous workflows raise significant concerns regarding reliability, security, and accountability. In critical environments, such as SOCs, incorrect or inadequately validated actions can compromise production systems, unnecessarily exacerbate incidents, or make it more difficult to verify the results of investigations.

2.3.2 LLM applications in cybersecurity

In recent years, large language models have revolutionized numerous industries. From ChatGPT to Claude, from Gemini to LLaMa, these tools are now ubiquitous in everyday life and are radically changing the way people interact with technology. It was, of course, inevitable that these technologies would also make their way into the world of cybersecurity.

Today, LLMs are being deployed across the entire spectrum of security operations. Major technology companies have integrated them into their core security platforms. Google's Security

Operations platform (formerly known as Chronicle) uses Gemini to enable natural-language searches across all collected security telemetry data, allowing analysts to query the data as usual and receive AI-generated detection rules mapped to the MITRE ATT&CK framework [19, 18]. Microsoft's Security Copilot, powered by GPT-4, integrates with Sentinel to provide incident summaries, guided response actions, and automated KQL query generation, thereby making threat hunting accessible even to analysts who may not be proficient in complex query languages [24, 9]. In addition, Microsoft's Sentinel MCP server can gather threat intelligence from external sources (such as technical reports or blog posts), identify attack patterns, and automatically scan internal data for similar activity [2].

In addition to more traditional commercial platforms, the cybersecurity research community has developed large language models (LLMs) that are specialized and tailored to specific security domains or tasks. Models such as CyBERT, MalBERT, and SecBERT are trained on industry-specific datasets for malware classification, phishing detection, and vulnerability analysis [30]. The CTIBench and CyberSecEval 2 benchmarks, both released in 2024, provide standardized evaluation suites to assess the performance of LLMs in tasks such as phishing detection, malware analysis, intrusion detection, and incident response [5, 7].

While security teams use large language models (LLMs) to automate defensive security operations, hackers are exploiting these same capabilities to make their activities even harder to detect. For example, there have been reports of cases where LLMs have been used to generate variants of polymorphic malware capable of evading signature-based detection systems. Industry reports highlight a concerning trend: malware developed using large language models (LLMs) has risen from 2% of all detected threats in 2021 to an estimated 50% by 2025 [1]. Furthermore, the widespread adoption of artificial intelligence tools has lowered the barrier to entry for sophisticated attacks, enabling even non-experts to create convincing spear-phishing emails and generate obfuscated malicious code on a large scale[20].

2.3.3 Multi-Agent Debate

Another promising line of research examines multi-agent debate as a mechanism for improving decision quality. Du *et al.* show that having multiple large language models generate candidate solutions and then critique each other's reasoning leads to more accurate and consistent answers on complex reasoning tasks [14].

Multi-agent debate is based on the observation that the first plausible explanation is not always the correct one. During a long investigation, a model may continue to build on an early assumption even when later evidence weakens it. A second agent can be used to examine that conclusion from a different position and identify evidence that was ignored or interpreted too confidently.

2.4 Related work

Recent work has started to explore how LLM-based agents can support SOC analysts not only by classifying alerts, but also by collecting evidence, coordinating investigation steps, and producing explanations that can be reviewed by humans.

The following sections discuss three representative systems, CORTEX [53], AgentSOC [34] and AutoSOC [27], which are particularly relevant to WARDEN because they address the same shift from static alert classification to evidence-based and agentic investigation.

2.4.1 CORTEX: Collaborative LLM Agents for Alert Triage

CORTEX [53] is one of the closest works to this thesis, since it explores the use of collaborative LLM agents for SOC alert triage. Its main assumption is that a single LLM should not manage the whole investigation alone. Instead, the process is divided among specialized agents, following a *divide-and-conquer* design.

System architecture

As shown in Figure 2.6, CORTEX organizes the triage workflow into four stages. An *Orchestrator Agent* manages the pipeline, a *Behavior Analysis Agent* routes the alert to the most suitable investigation workflow, workflow-specific *Evidence Acquisition Agents* query enterprise tools such as SIEM, identity, and asset systems, and a final *Reasoning and Coordination Agent* combines the collected evidence into an auditable decision report.

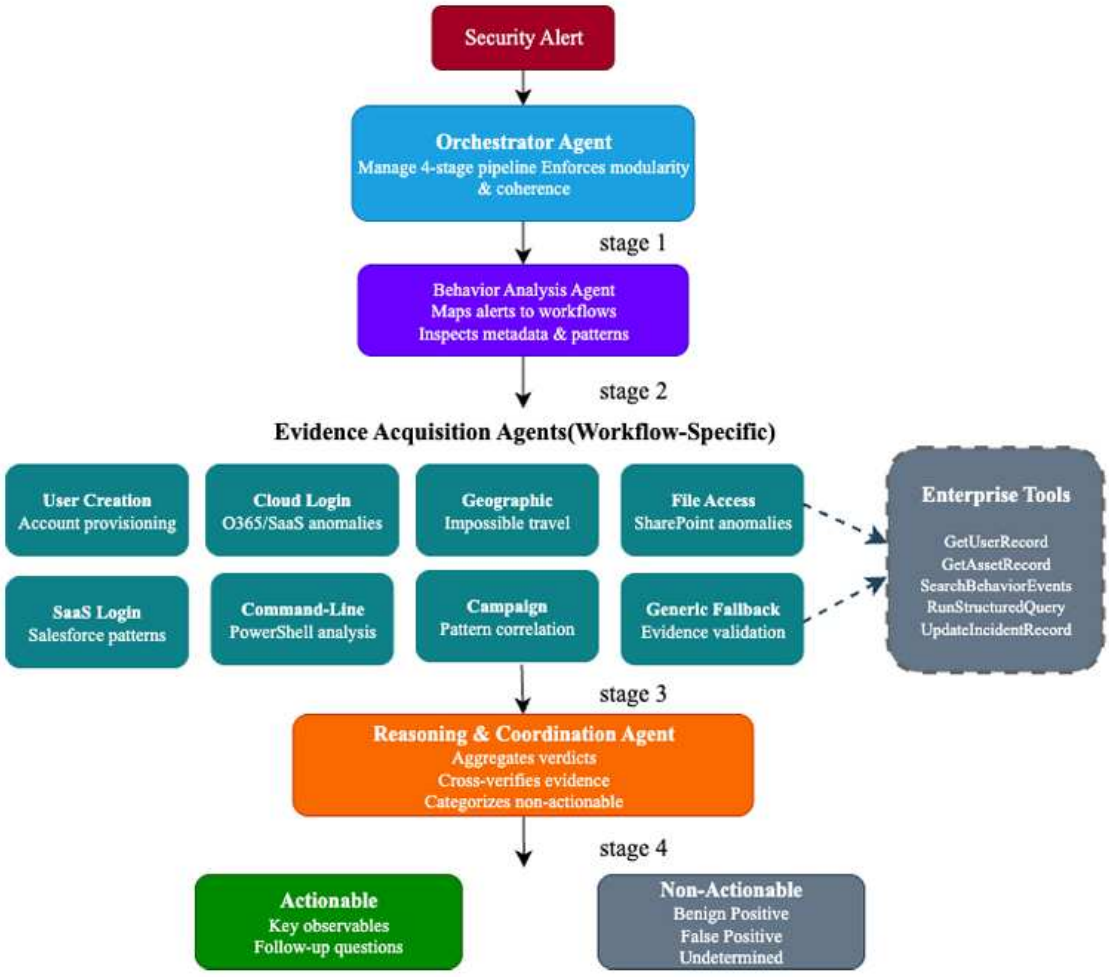


Figure 2.6: CORTEX architecture [53]

What makes CORTEX relevant is not only its multi-agent structure, but also the type of investigation enabled by this architecture. The agents do not produce a verdict from the model's internal knowledge alone. They retrieve external evidence through specialized tools and ground the final decision in observable data, such as SIEM events, identity records, resource metadata and structured queries. This model is therefore guided toward an evidence-based investigation, rather than a purely text-based classification.

CORTEX defines alert triage as a binary classification between *actionable* and *non-actionable* alerts. An alert is considered *actionable* when it requires escalation or further response. In contrast, a *non-actionable* alert does not require immediate escalation and may correspond to a benign event, a false positive or a case where the available evidence is insufficient. The authors further divide non-actionable alerts into more specific categories, including false positives, logic-based false positives, data-based false positives and indeterminate cases [53].

The evaluation does not rely only on aggregate accuracy. The authors report F1 scores separately for actionable and non-actionable alerts, because a single accuracy value may hide poor performance on the class that matters most operationally. The distinction also reflects the different cost of each error. Missing an actionable alert may leave a real attack without investigation. Marking a benign alert as actionable creates the opposite problem: it consumes analyst time and increases noise in the queue. Reporting the two classes separately makes this trade-off explicit.

Performance evaluation

CORTEX results are broken down according to two dimensions: the quality of the decision and the cost incurred in generating that decision. For triage quality, the authors compare CORTEX with two simpler baselines: a single LLM that classifies alerts without tools, and a single tool-using agent that gathers evidence before making a decision. The evaluation reports F1 scores for actionable and non-actionable alerts, together with the false-positive rate, as shown in Table2.2.

Model	**Act. F1**	**Non-act. F1**	**Subclass F1**	**FPR (%)**
Single-agent (prompt)	0.61	0.73	0.49	29.8
Single-agent (tool-use)	0.66	0.77	0.54	24.9
CORTEX	0.78	0.86	0.69	14.2

Table 2.2: Decision performance reported by CORTEX [53]

These results should be interpreted as evidence of a general trend, rather than as a benchmark directly comparable to WARDEN. The systems use different datasets, and no shared benchmark applies the same metrics to all the architectures discussed in this chapter. The transferable result is qualitative: role-specialized agents grounded in retrieved evidence improve triage quality and reduce unnecessary escalations compared with a single-agent design.

Model	**Tokens**	**Tool calls**	**Latency (s)**
Single-agent (prompt)	2,100	0.0	28.0
Single-agent (tool-use)	4,152	1.3	44.6
CORTEX	23,600	3.1	152.4

Table 2.3: Efficiency comparison reported by CORTEX [53]

The point most relevant to WARDEN concerns cost. As shown in Table 2.3, CORTEX achieves better triage quality at a higher operational cost. It performs more tool calls, consumes more tokens, and requires more time per alert than the single-agent baselines [53]. This efficiency gap, more than the exact accuracy values, directly informs WARDEN's design.

Scope and applicability

The main limitation of CORTEX is its operational cost. Dividing triage among several specialized agents improves the quality of investigation, but also increases the number of exchanged messages, tool calls and token required for each alert. In a production environment, a large part of the daily alert queue usually consists of recurring false positives and log-risk events. Running a complete multi-agent pipeline on all of them would be difficult to sustain in terms of both latency and cost.

In any case, CORTEX clearly demonstrates the value of multi-agent reasoning in an analytical context. This is precisely the point that most influenced WARDEN: the final verdict should base its conclusions primarily on the evidence gathered, rather than solely on the model's internal knowledge. This also helps prevent hallucinations, which, as previously mentioned, is a major problem with LLMs.

WARDEN maintains these principles, but eliminates the assumption that the same level of reasoning must be applied to every alert. Instead of employing the entire multi-agent system, WARDEN handles ordinary cases and those recognized as false positives, while a second agent is activated only when the alert is ambiguous, of high severity, or sufficiently risky to require more aggressive intervention.

2.4.2 AgentSOC: Multi-Layer Agentic Framework

AgentSOC [34] proposes a broader agent-based architecture for SOC automation. While CORTEX focuses primarily on alert classification and investigation, AgentSOC aims to cover a broader portion of the SOC workflow: from alert ingestion to response planning. Its goal is not only to determine whether an alert should be escalated to a higher level, but also to assess what the attacker's next moves might be and what defensive actions would be appropriate.

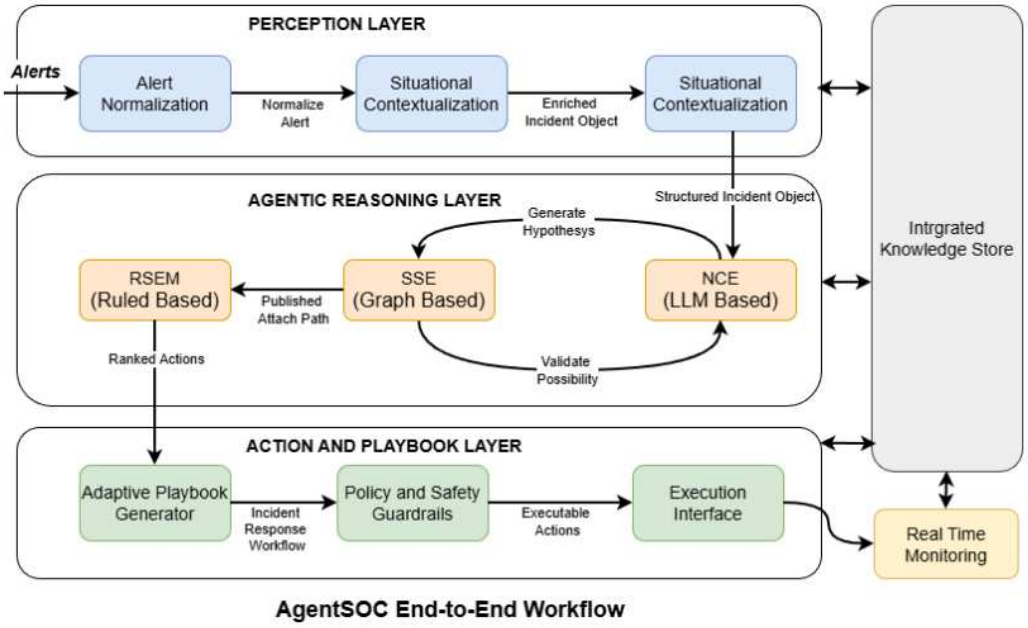


Figure 2.7: AgentSOC End-to-End Workflow [34]

System architecture

As shown in Figure 2.7, AgentSOC operates through a continuous *Sense–Reason–Act* cycle of operations. The first phase is handled by the *Perception layer*, which receives alerts from security tools and transforms them into structured incident objects. This involves normalizing the alert, adding contextual information, and reducing noise. In practical terms, the system does not merely process the raw alert but enriches it with information about users, resources, privileges, network topology, and security policies.

The central part of the framework is the *Agentic Reasoning Layer*. This layer is responsible for moving from a single alert to a possible attack narrative.

1. First, the *Narrative Counterfactual Engine* uses LLM-based reasoning to generate plausible hypotheses about the incident.

   These hypotheses are not accepted immediately but they are passed to the *Structural Simulation Engine*.

2. The *Structural Simulation Engine* checks whether the proposed attack path is actually feasible in the enterprise environment.

This is an important design choice: an LLM may generate an explanation that sounds reasonable, but that explanation is not useful if the required network path, privilege relationship, or asset dependency does not exist[4].

3. After the validation step, the *Risk Scoring and Evaluation Module* ranks possible defensive actions. The objective is not simply to choose the strongest containment action, but to balance security effectiveness with business impact.

   For instance, isolating a host may be effective from a defensive point of view, but it may also interrupt a critical service.

   AgentSOC applies this reasoning to both sides of the decision. It estimates how an attack might progress, but it also checks whether the proposed countermeasure could create an unacceptable operational impact.

The selected response is then handled by the *Action and Playbook Layer*. This layer translates the results of the reasoning into actions that comply with company policies, such as isolating a host, restricting privileges, enforcing multi-factor authentication (MFA), or continuously monitoring the situation. Before execution, the action is validated against security criteria and organizational constraints. High-impact actions can be escalated to human analysts, while low-risk actions can be executed automatically or tested in simulation mode.

Performance evaluation

The paper evaluates AgentSOC through a proof-of-concept based on a 5,000 events sample from the LANL authentication dataset[5]. The authors simulate an authentication anomaly involving possible credential misuse and lateral movement. In the reported example, the system enriches the alert, generates multiple attack hypotheses, checks whether they are feasible in the enterprise context, and ranks possible response actions.

The overall behavior of the prototype is summarized in Table 2.4. The LLM-based reasoning module generates three possible explanations for the same alert. The most likely is credential misuse, followed by lateral movement, while the benign misconfiguration hypothesis is assigned a lower confidence score. It is important to note that the hypotheses are passed to the structural validation phase, which verifies them in light of the organizational topology and privilege relationships.

In this example, the hypothesis of an incorrect but harmless configuration is discarded because it is not supported by the modeled environment.

After the hypothesis validation phase, AgentSOC ranks potential response actions using a risk-based score. In the proof-of-concept, host isolation receives the highest score because it ensures effective containment with a relatively limited operational impact. User deactivation is also considered, but it entails a greater operational impact, while simple monitoring is deemed insufficient for the threat level described in the scenario [34]. The resulting mitigation ranking is reported in Table 2.5.

[4]This behavior is consistent with the known tendency of LLMs to produce plausible but unsupported outputs when they are not grounded in external evidence or constrained by verifiable context [31]. In SOC environments, this is especially problematic because a convincing explanation may still be operationally wrong.

[5]https://csr.lanl.gov/data/auth/

ID	Generated hypothesis	Confidence	Structural validation
H1	*Credential misuse followed by lateral movement*	0.74	Feasible: the network path exists and the user's group membership allows an SMB pivot.
H2	*Kerberos ticket abuse followed by privilege escalation*	0.52	Conditionally feasible: the path is possible only if Tier-1 credentials are present on the target.
H3	*Benign misconfiguration*	0.21	Not feasible: no service or scheduled task associated with the user supports this explanation.

Table 2.4: AgentSOC hypothesis generation and structural validation [34]

The authors report the processing time for each phase. The complete reasoning cycle takes approximately 506 ms. Most of this time is spent on the hypothesis generation phase based on the large language model (LLM), while graph validation and risk score calculation are relatively lightweight operations.

Rank	Action	Containment	Impact	Score
1	*A1 - Isolate ws-fin-27*	0.92	0.15	0.599
2	*A2 - Disable user123*	0.84	0.30	0.498
3	*A3 - Monitor events*	0.15	0.00	0.105

Table 2.5: AgentSOC mitigation ranking [34]

Scope and applicability

AgentSOC is presented by its authors as a proof of concept and should be interpreted accordingly. Its evaluation is based on a sample of the LANL authentication dataset, the enterprise topology used for validation is synthetic and limited to fifty nodes, and response actions are performed in dry-run mode rather than in a real environment. These aspects do not detract from the work's interest, but they place AgentSOC in a different, broader context than WARDEN. AgentSOC addresses a closed-loop workflow that extends to countermeasure selection and execution, while this thesis focuses on level-1 triage of a real, heterogeneous alert queue. The two systems therefore address different parts of the SOC workflow.

What WARDEN inherits from AgentSOC is its caution and scrutiny of plausible explanations. AgentSOC doesn't act on a hypothesis just because the model deems it convincing, but first checks whether it holds true in the context of the monitored organization. WARDEN draws on the same principle to implement its verdict system, but uses a different method. It requires evidence from the monitored environment and allows the challenger agent to reconsider a verdict, especially when certain claims are true.

The part that WARDEN omits is response execution. AgentSOC can isolate a host, disable a user, or change privileges, while this approach keeps these operations under human control. In a multi-tenant SOC, an incorrect containment action would not only waste analysts' time but could also disrupt a customer's infrastructure. Furthermore, some customers explicitly require that no containment actions be performed automatically, but that a human operator always be the author of certain actions.

For this reason, WARDEN stops at the triage decision and proposes response steps rather than executing them. Integrating WARDEN with an EDR to prepare such actions and execute them after human confirmation remains a natural direction for future work.

Furthermore, considering the hundreds of customers continuously monitored (and thousands of devices), creating a specific knowledge base for each of them is too costly. This task would require ongoing maintenance every time an operation is performed in the environment. Since the system administrators of the monitored environments are not the company I work for, requiring continuous feedback for every maintenance operation is impractical and, more importantly, unsustainable.

2.4.3 AutoSOC Cyber Analyst: Tier 1 and Tier 2 Automation

AutoSOC Cyber Analyst (ASOC-CA) [27] is an AI-assisted framework designed to help SOC analysts classify alerts, map events to the MITRE ATT&CK framework, and report incidents. Unlike CORTEX and AgentSOC, AutoSOC does not rely primarily on LLM-based agents to conduct investigations. Instead, it combines a machine learning classifier, a deterministic mapping layer, and an LLM-based reporting component.

AutoSOC addresses a narrower and clearly defined part of the SOC workflow. It classifies SIEM alerts and uses the resulting prediction to prepare an incident report for the teams that continue the investigation. The system is therefore intended to accelerate selected Tier 1 and Tier 2 activities rather than reproduce the full work of a SOC analyst.

System architecture

Figure 2.8 shows the relatively linear structure of AutoSOC. The monitored hosts forward their logs to Splunk Enterprise, where correlation searches produce the alerts used by the system. ASOC-CA receives each alert through a Splunk HTTP Event Collector webhook and converts it into the feature representation required by the classification model.

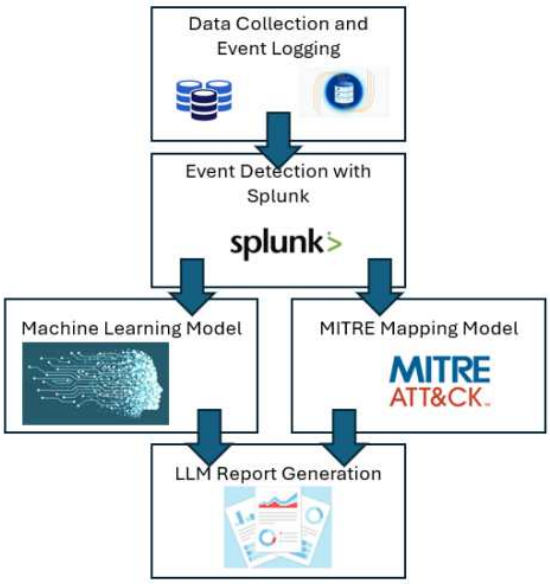


Figure 2.8: AutoSOC high-level architecture [27]

Each alert is normalized and transformed into a structured feature vector. This feature vector contains information such as source and destination IP addresses, host identifiers, authentication results, temporal characteristics, and protocol metadata. The classification phase is then performed by a machine learning model. In the implementation described by the authors, the primary use case is the detection of brute-force attacks on the Remote Desktop Protocol (RDP).

Following machine learning-based classification, AutoSOC applies a deterministic layer of MITRE ATT&CK mapping. For example, RDP brute-force activity is mapped to technique T1110[6]. This deterministic layer is an important design choice because it avoids leaving the

[6]https://attack.mitre.org/techniques/T1110/

ATT&CK mapping entirely to the LLM. The LLM is used only after classification and mapping, with the goal of generating a human-readable incident report. The report summarizes the stage of the suspected attack, relevant indicators, affected hosts, and recommended response actions.

In this sense, AutoSOC primarily uses the language model (LLM) as a reporting and explanation component. The fundamental decision regarding whether an activity is malicious is made by the machine learning classifier, while the deterministic layer provides the ATT&CK mapping and remediation references. This makes AutoSOC more controlled than a system based exclusively on an LLM, but also more dependent on the quality of the trained classifier and the specific scenario for which it was designed.

Performance evaluation

The experiment compares a supervised Random Forest with an unsupervised K-Means model. Both are tested on data produced in the authors' virtualised network, where repeated RDP brute-force attempts are executed while Splunk records the resulting activity. The evaluation therefore concerns one well-defined distinction: separating the simulated attacks from legitimate authentication traffic.

Table 2.6 summarizes the results obtained. The Random Forest classifier offers the best performance, with an accuracy of 98.1%, a precision of 98.9%, and a false positive rate of 1.06%. K-Means also achieves high accuracy, but performs worse on all reported metrics, especially in terms of false positives and false negatives. In this scenario, the supervised model is therefore better suited for the classification task.

Model	**ACC (%)**	**TPR (%)**	**TNR (%)**	**FPR (%)**	**FNR (%)**	**Precision (%)**
Random Forest	98.1	98.9	92.9	1.06	7.14	98.9
K-Means	95.3	96.8	86.7	3.26	12.5	97.8

Table 2.6: Comparison between Random Forest and K-Means [27]

The results obtained are particularly promising, especially for the Random Forest model. However, they must be interpreted within the context of the experimental setup. The evaluation focuses on a specific attack model—namely, the RDP brute-force attack—and the features were designed for that scenario. This makes the results useful as proof of concept, but does not demonstrate whether the same pipeline can be generalized to a broader set of SOC alerts, such as privilege escalation, execution of suspicious processes, data exfiltration, cloud identity abuse, or file integrity violations.

The document also evaluates the quality of reports generated by large language models (LLMs) using text generation metrics such as ROUGE[7] and BLANC. This is useful because it demonstrates that AutoSOC is not limited to classification but also aims to produce results intended for analysts. It is important to note that these metrics do not fully measure whether the investigation process is correct or provides useful information, whether all relevant evidence has been collected, or whether the final decision is valid under ambiguous conditions. They only report on the quality of the generated text.

[7]https://en.wikipedia.org/wiki/ROUGE_(metric)

Scope and applicability

AutoSOC is evaluated based on a single, well-defined application domain: RDP brute-force attack attempts versus legitimate authentication traffic. The report quality metrics, ROUGE and BLANC, evaluate the readability and similarity of the generated text but do not show whether the underlying investigation was correct or complete. For a stable and repetitive detection task, a compact supervised pipeline is a reasonable and efficient choice. However, this also means that the approach does not easily transfer to the problem addressed in this thesis, where the alert queue is heterogeneous and a labeled dataset is not available for every type of alert that might appear.

This is where WARDEN differs from AutoSOC. AutoSOC bases its decision on a classifier trained on labeled examples of a specific attack, while WARDEN cannot assume that such a dataset exists for every alert it receives. Alerts that do not match a known false positive pattern are analyzed by directly querying available data sources. This way, the same analysis mechanism can also be applied to alert types that have never been included in a training set.

WARDEN, however, adopts an important idea from AutoSOC. AutoSOC excludes the language model from stages that don't require it, using a deterministic layer to map techniques to MITRE ATT&CK rather than delegating that task to the language model. WARDEN follows the same principle in its first stage, where a deterministic false positive filter excludes known benign patterns before any model is invoked. The more complex reasoning is then reserved for cases that actually require it.

2.5 Research Gaps

The literature reviewed in this chapter demonstrates a clear shift in the world of defensive cybersecurity: from static alert automation to agent-based, context-aware SOC systems. Rule-based detection and machine learning models remain useful for identifying, filtering, and prioritizing alerts, but they only address part of the much more complex problem of triage.

In the day-to-day work of a SOC, the difficult task is rarely simply deciding whether an alert indicates an attack. The real challenge is understanding whether that alert is relevant to the specific customer environment, gathering the appropriate evidence for or against the resulting hypothesis, and deciding whether the case can be closed as a false positive or escalated.

The three systems discussed previously in Section 2.4 all move in this direction. The previous sections examined which aspects of their design can be reused and where they become less applicable to the context of this thesis. AutoSOC is efficient, but constrained to a single, well-defined scenario. AgentSOC extends beyond first-level triage to include response execution, which in this work is intentionally kept under human control. Additionally, it requires the effort of creating and maintaining a knowledge base for each individual organization, which, as described, is simply too costly to achieve. CORTEX bases its actions on the evidence retrieved, but applies a comprehensive multi-agent pipeline to each alert.

What remains unresolved is how much agent logic is actually required for a given alert and how to evaluate its performance in a real-world environment with thousands of devices and hundreds of continuously monitored environments.

A SOC doesn't require the same depth of analysis for every alert: some alerts can be closed quickly because they match a known benign pattern in a known context, while others require evidence gathering and contextual interpretation. Treating all alerts equally wastes resources on simple cases and may still fail to examine complex ones with due care. The open question, therefore, is not whether LLM agents are capable of prioritizing alerts, but how they should be orchestrated so that investigation quality, latency, token usage, and human oversight remain acceptable in practice. This issue becomes even more relevant if the system is also used to examine lower-severity events for anomalies that would not normally trigger an alert. In such a case, running an unconditional LLM pipeline without considering false positives or organizational context would quickly become untenable.

Many of the proposals examined in this chapter are tested on a single attack scenario, a synthetic topology, or a small proof of concept. These experiments are useful for demonstrating that an architecture can work, but they say little about its behavior in a real SOC. An implementation-oriented evaluation must look beyond the final label. The same alert can produce different responses in repeated executions, a correct decision may still be impractical if too slow or too expensive, and the explanation should be verified to ensure it is actually based on evidence retrieved from the environment rather than hypotheses formulated by the model or hallucinations.

In this thesis, WARDEN is used to study these questions in a Tier 1 context. It connects to the tools already available in the SOC and analyzes alerts using the data retrieved from them. At the same time, it avoids running the entire agent workflow when a previously validated false positive rule is sufficient to close the case.

2.6 Conclusions

This chapter has described the operational context in which this thesis is positioned. Security Operations Centers process large volumes of alerts every day, many of which are false positives or low-priority events. This creates alert fatigue, increases the cognitive load on analysts, and makes triage decisions harder to keep consistent over time.

The techniques reviewed in this chapter automate different parts of alert handling. Rules remain effective for stable and recognizable patterns, although they require tuning as the environment changes. Machine-learning models can rank alerts or identify deviations from a learned baseline, but their output is generally a prediction rather than an investigation.

Tier 1 triage also involves checking what happened around the alert, identifying the affected asset, and determining whether the observed activity is expected in that customer's environment. These steps explain why classification accuracy alone does not capture the complete task.

Tool-using LLM agents offer a way to combine some of these steps. They can formulate queries, read the returned events and organize the findings into a report that an analyst can inspect. The systems discussed in Section 2.4 implement this idea at different levels.

CORTEX divides triage among specialized agents and grounds their conclusions in enterprise data, although this results in additional model calls and latency. AgentSOC extends the workflow to hypothesis validation and response selection, including operations that WARDEN intentionally leaves outside its scope. AutoSOC uses a lighter pipeline and obtains strong results for RDP brute-force detection, but the experiment does not cover a heterogeneous alert queue.

These differences lead to the design question addressed in the following chapter: how much agentic reasoning is actually necessary for each alert? Running a complete multi-agent workflow may be justified for a critical or ambiguous case, but it is wasteful for an alert that matches a false-positive pattern already reviewed by an analyst. WARDEN is built around this distinction. Its architecture combines inexpensive preliminary checks with a full investigation only when the available evidence requires one.

This motivates the design of WARDEN. Rather than applying the same reasoning pipeline to every alert, WARDEN follows a conditional multi-agent approach. Known false positives can be handled through a fast knowledge base, ordinary alerts can be investigated by a primary agent using retrieved evidence, and ambiguous or high-risk cases can be passed to additional verification agents. The next chapter presents this architecture in detail and explains how it is designed to balance investigation quality, cost, latency, and human oversight in a realistic SOC environment.

Chapter 3

Architectural proposal

Given the problems identified in Chapter 2, namely alert fatigue, triage inconsistencies, and the limited flexibility of rule-based automation, this chapter introduces the architecture of WARDEN, a conditional agentic system designed to support SOC alert triage.

Rather than treating triage as a pure classification task, WARDEN uses large language models as reasoning components that can select investigative steps, invoke tools, interpret retrieved evidence, and produce a structured verdict for analyst review.

This chapter will introduce the proposed architecture of WARDEN, the system implemented throughout the course of this thesis for supporting alert triage and investigation in a SOC environment. While Chapter 4 will describe implementation details, this chapter focuses on explaining the reasoning behind the design and the logical flow of the investigation pipeline.

WARDEN is a conditional multi-agent architecture designed for analyzing SOC alerts. A single primary agent is tasked with performing reasoning operations and secondary agents only get involved when they have operational utility, for instance verifying suspicious results or confirming the validity of potentially false alerts. The use of secondary agents is motivated by the nature of the SOC workflow, where deploying more than one agent per alert can increase reasoning quality, as suggested in CORTEX [53], at the cost of higher latency, greater token consumption and increased costs. For that reason, WARDEN avoids utilizing multi-agent processing in all cases by introducing a conditional stage-based investigation pipeline which allows more computationally intensive steps of the analysis pipeline to be applied conditionally.

The system is based on the idea that not all SOC alerts should be treated equally. While some false positive alerts could easily be categorized and dismissed based on certain criteria, other cases were too vague to allow automated reasoning. Therefore, WARDEN distinguishes between deterministic automation, adversarial verification and reasoning based on LLMs.

3.1 Design Principles

The findings obtained by Wei *et al.* on the CORTEX [53] system indicate that multi-agent architectures can achieve important improvements in alert triage accuracy when agents are constrained to ground their reasoning in external evidence rather than relying on parametric knowledge alone.

WARDEN follows four major principles, namely evidence-based reasoning, conditional utilization of agents, efficiency, and analyst reviewability.

3.1.1 Evidence-Based Reasoning

The first design requirement is that every decision must be grounded in concrete and verifiable evidence retrieved from the monitored environment. In a Security Operations Center (SOC), a verdict is useful only if an analyst can understand which data was considered, how it was interpreted, and why a specific conclusion was reached [17]. A generic response produced by a language model, even when plausible, is insufficient for security operations because it may omit relevant context or provide unsupported explanations.

Therefore, WARDEN considers the LLM as a reasoning module instead of an isolated classifier. This means that the investigative agent needs to gather evidence from operational sources, such as SIEM alerts, endpoint inventory, known indicators of compromise, background information, and related activities among monitored agents. The final output must include not only a verdict, but also the evidence used to support it, relevant indicators, the confidence level, and suggested next steps.

This approach is consistent with that observed in modern agent-based SOC systems [53, 34, 37], where the central challenge is not simply to assign a label to an alert, but to coordinate the use of tools, the collection of evidence, and contextual interpretation. Compared to traditional classifiers, which are often fragile and difficult to interpret in operational contexts [36], this approach makes the analysis more transparent and easier to verify.

3.1.2 Conditional Multi-Agent Design

The second implementation principle is the conditional activation of agents. By default, WARDEN does not use multiple agents for each alert; instead, the architecture activates specialized agents only when certain conditions are met.

This choice was motivated by the need to find the right balance between investigation quality and operational costs. Multi-agent systems such as CORTEX [53] demonstrate that agents specialized in specific roles can improve triage quality, but they also introduce greater latency and increased token usage [43]. In a production SOC environment, this trade-off is critical because a system that is accurate but too slow or too expensive may be difficult to deploy at scale.

WARDEN does not invoke all agents for every alert. The main investigator handles the cases that require a complete analysis. Before reaching that stage, the system checks whether the alert resembles a false positive that has already been reviewed. When the match is plausible but not strong enough for deterministic closure, a smaller validation agent examines only that question. The challenger is reserved for cases in which the severity or uncertainty makes an independent review worthwhile.

This arrangement was chosen to avoid paying the coordination cost of a multi-agent workflow when a simpler decision is sufficient.

3.1.3 Operational Efficiency

The third principle involves efficiency. SOCs handle large volumes of alerts, many of which are repetitive or of minor significance [4, 39]. Performing a full LLM-based investigation for every alert would be inefficient and could render the system impractical.

The pipeline begins with operations that are cheaper and easier to reproduce than an LLM investigation. WARDEN first queries its false-positive knowledge base using fields extracted directly from the alert. This phase is deterministic and introduces no model cost.

A match is sufficient for automatic closure only when it satisfies the required confidence and safety conditions. Otherwise, the alert continues through the pipeline. The system therefore saves model calls on known cases without treating every partial match as proof that the alert is benign.

This multi-stage design helps reduce unnecessary LLM calls while maintaining the ability to perform more in-depth analysis when needed. In other words, the pipeline is designed to deploy computational resources where they are most useful.

3.1.4 Traceability

The fourth principle is traceability. WARDEN is not intended to replace the analyst in the SOC workflow [10]. Its goal is to reduce repetitive work, accelerate evidence collection, and provide a structured preliminary assessment that can be reviewed by a human operator.

For this reason, the final verdict is not a single label; rather, the analysis output contains a structured decision, a confidence score, a severity assessment, the reasoning behind the conclusion, the evidence gathered during the investigation, MITRE ATT&CK mappings where applicable, and recommended actions. This makes the output suitable for review by the analyst and for subsequent evaluation of the system.

3.2 High-Level Architecture

Generally speaking, WARDEN takes a Wazuh [1] alert (the SIEM system used) and feeds it into the investigation workflow pipeline that contains multiple steps. The dispatcher serves as the core orchestrator, which means it extracts the alert, performs initial checks to exclude false positives, checks whether there should be an investigation at all, runs corresponding agents and tools, and logs the decision outcome.

The architecture of the entire project can be broken down in into three logical macro-areas:

- *Inputs and information sources*, containing Wazuh alerts, OpenSearch indices, endpoints' list, threat intelligence platforms, and behavioral baselines;
- *WARDEN core engine*, containing the dispatcher, deterministic false positive filter, AI-based false positive validator, primary investigator agent, challenger agent, and the tool interface;

[1] https://wazuh.com/

- *Output and Review layer*, encompassing the verdict storage, investigation report generation, logs, and the user interface for the analysts.

The main flow of the investigation is summarized in the Figure 3.1. The dashed-line steps are performed only when the alert requires it.

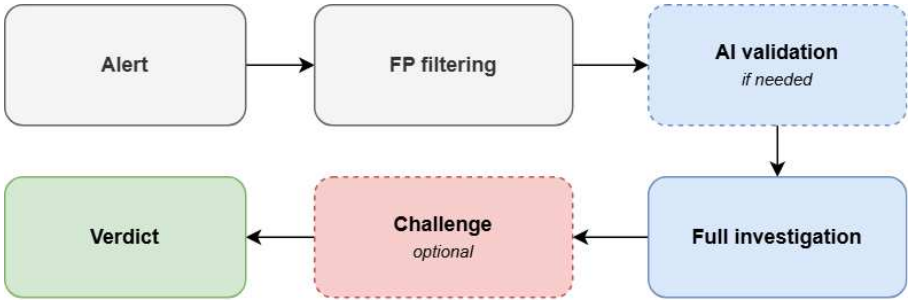


Figure 3.1: WARDEN investigation loop

The use of a dispatcher is essential because it separates coordination tasks from the reasoning process. The dispatcher does not evaluate whether an alert is unreliable. Its role is to control the execution flow, enforce safety rules, prepare the context for agents, record metrics, and ensure that the final output is saved.

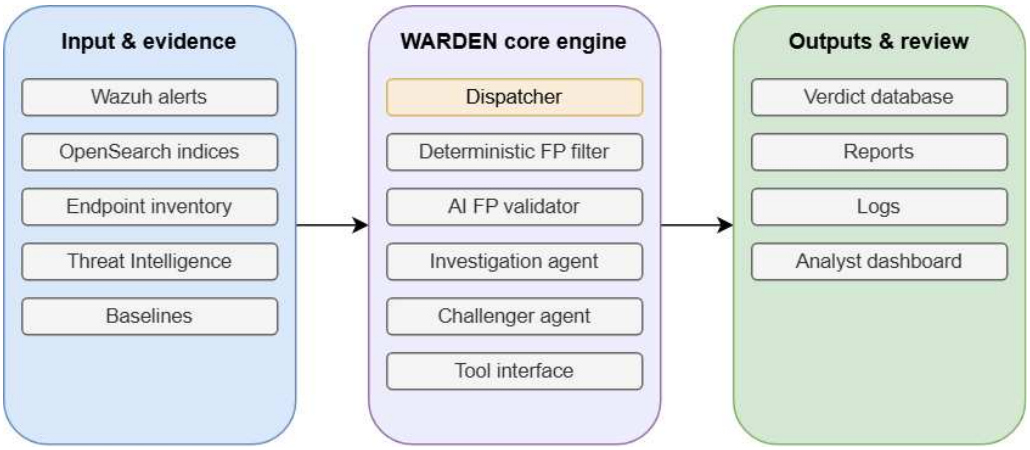


Figure 3.2: High-level architecture of WARDEN.

3.3 Multi-Phase Investigation Pipeline

WARDEN adopts a multi-phase investigation pipeline. The system applies increasingly detailed analysis as an alert moves through the pipeline. Each phase has a clear role: it either closes the investigation early or forwards the alert to the next phase for deeper analysis. Figure 3.3 shows the complete investigation flow.

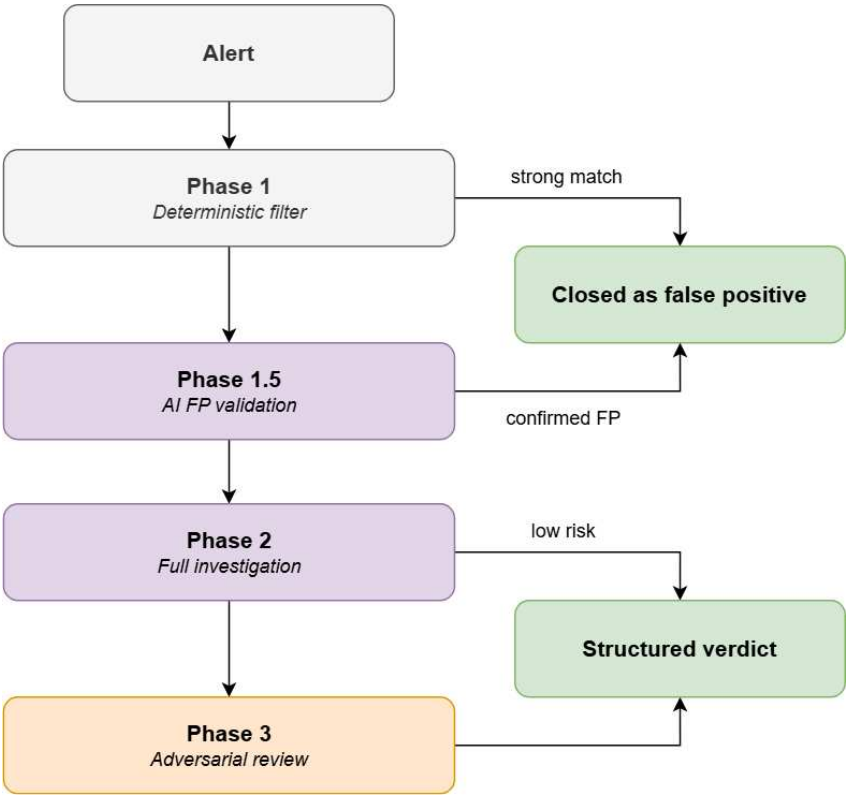


Figure 3.3: Multi-Phase investigation pipeline

3.3.1 Phase 1: Deterministic False Positive Filtering

Before starting a full LLM-based investigation, WARDEN applies a preliminary false-positive filtering stage. This stage is designed to reduce the cost and latency of repeated investigations on alerts that analysts have already identified as benign.

In a real SOC, many alerts are repetitive. For example, the same monitoring script, backup process, vulnerability scanner, or administrative activity may trigger the same alert every day on the same host. Therefore, after an analyst's initial review of the case and subsequent confirmation that the alert in question is a false positive, it would be appropriate to update the monitoring systems to prevent the same false detection from being generated in the future. To enable this, WARDEN allows you to save and manage false positives through its own knowledge base, which can be expanded as desired by the analyst over time. Following an incorrect detection or false positive, an operator can add this information to the system and thus prevent similar future alerts from requiring a full investigation.

The false-positive knowledge base acts as a memory of already reviewed benign cases. When a new alert arrives, WARDEN checks whether it contains elements that are clearly related to a known false positive. These elements may include, for example, a recurring file path, process

name, service name, source address, or other stable information that helps explain why the previous alert was considered harmless.

3.3.2 Phase 1.5: AI-Assisted False Positive Validation

The second phase, also called Phase 1.5, is used only in the case of false positive candidates. This phase is invoked whenever Phase 1 finds a possible match but the detection confidence is not high enough to safely close it. Phase 1.5 uses an AI validation agent that receives the alert and the candidate false-positive pattern. The agent then decides how close the alert is to the scenario described by the exclusion rule.

The goal of this phase is not to replace the main reasoning agent, but rather to understand whether a potentially false positive alert can actually be closed or whether the analysis should continue.

This phase is essential to avoid closing alerts as false positives because a rule was mistakenly triggered or captured a case where it actually shouldn't have been triggered (for example, due to an analyst's configuration that was too imprecise and stringent).

3.3.3 Phase 2: Multi-Step Autonomous Investigation

All alerts that survive the first verification phase enter the actual investigation phase of WARDEN, also called Phase 2. Figure 3.4 illustrates the entire analysis process in detail. The agent first classifies the alert into one of the predefined investigation workflows. Each workflow defines the evidence to collect, the checks to perform, and the type of reasoning required for the specific alert category. WARDEN currently supports the following workflows:

- *SSH brute force*, for alerts related to repeated failed SSH authentication attempts;
- *File integrity*, for alerts generated by the *File Integrity Monitoring* capability of Wazuh agents [2];
- *Web attack*, for alerts related to web exploitation attempts, such as SQL injection, cross-site scripting, Shellshock, or similar payload-based attacks;
- *Anomalous login*, for alerts involving suspicious successful authentications, such as a login after repeated failures or activity outside expected working hours;
- *Ransomware*, for alerts involving encryption patterns, suspicious file extensions, ransom notes, or other behaviors associated with ransomware activity;
- *Firewall drop*, for alerts involving repeated firewall deny events, blocked connections, or suspicious traffic patterns;
- *PowerShell execution*, for alerts involving suspicious PowerShell activity, encoded commands, unusual parent processes, or execution patterns often linked to living-off-the-land techniques;
- *Windows Defender alert*, for alerts generated by Microsoft Defender, with a focus on threat name, affected file, remediation status, and containment result;

[2] https://documentation.wazuh.com/current/user-manual/capabilities/file-integrity/index.html

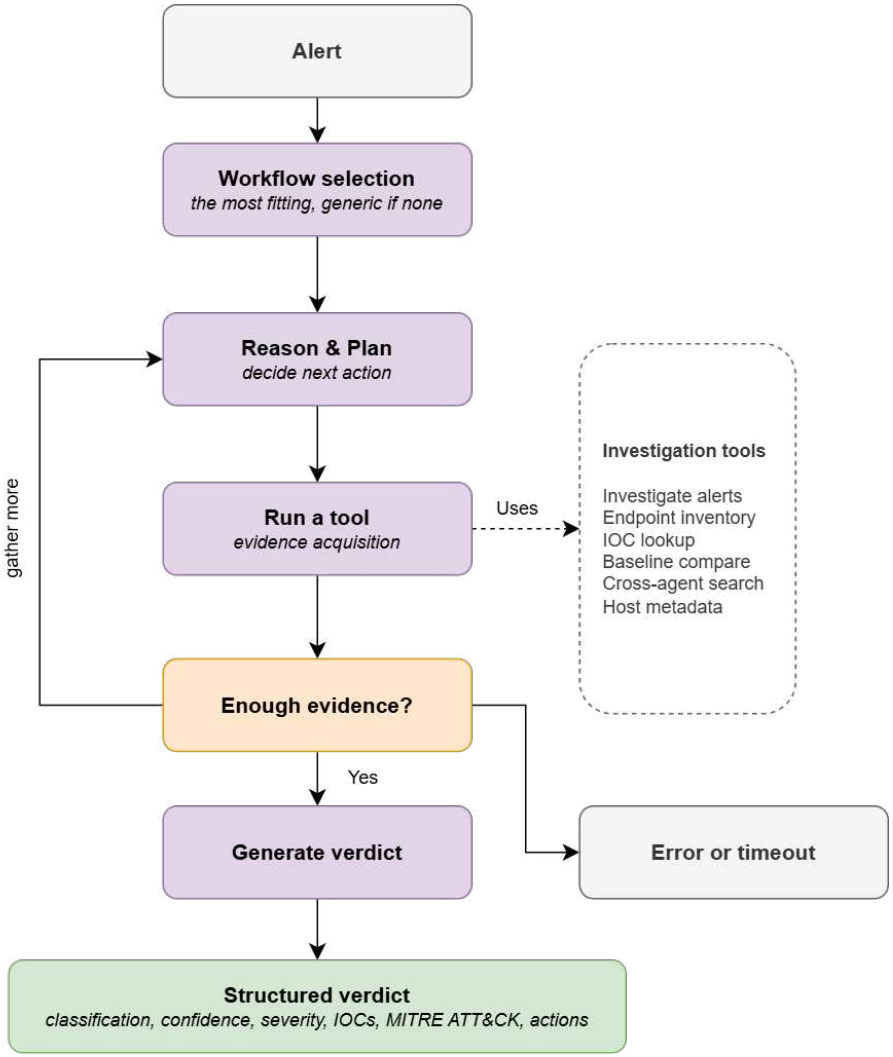


Figure 3.4: WARDEN Phase 2 investigation loop

- *Generic*, for alerts that do not match any specific workflow but still require a complete investigation.

Each workflow described above contains the set of operations (events) that the investigation agent should search for to correlate the event. This list of operations does not, however, represent a list of fixed steps that the agent performs and on the basis of which it produces its outcome.

For example, in an SSH brute force alert, the agent may try to collect the number of previously failed login attempts, identify the source IP address and check it against threat intelligence platforms to discover its reputation, search for the same identifier in other agents belonging to the same customer (to detect if the attack is limited to one agent or not) and investigate the host to find whether the host is compromised.

During the investigation, the agent has access to a set of tools developed as local Python scripts and therefore freely callable via CLI, that expose controlled operations on available security data. These tools allow the agent to query Wazuh alerts, inspect endpoint inventory, perform IOC searches, compare current status to baselines, search for correlations between agents, and retrieve metadata about the monitored host.

Listing 3.1 shows the contents of the `SKILL.md` file related to the *SSH Brute Force* workflow. The analysis process includes exactly what has been identified as essential for investigating this type of alert, in the form of CLI command syntax that must be executed by the agent.

````
#### SSH Brute Force

- **Alert context**
```bash
python tools/query_alerts.py --agent-id <ID> --around "<timestamp>" \
  --minutes 15 --rule-id 5712
```

- **Source IP reputation**
```bash
python3 tools/lookup_ioc.py --type ip-src --value <source_ip>
```

- **Cross-agent spread** (scoped to customer)
```bash
python3 tools/search_cross_agent.py --indicator <source_ip> --hours 24 \
  --agent-group "<GROUP>" --agent-prefix "<PREFIX>--"
```

- **Successful login check**
```bash
python tools/query_alerts.py --agent-id <ID> --search "authentication_success" \
  --source-ip <IP> --minutes 30
```

- **Host inspection** (if login succeeded)
```bash
python tools/query_inventory.py --agent-id <ID> --type processes
python tools/query_inventory.py --agent-id <ID> --type ports
```
````

Listing 3.1: SSH Brute Force workflow from SKILL.md file.

The investigation can finish in three ways. In the normal case, the agent reaches a conclusion that is supported by the collected evidence. The execution may instead stop because a required tool fails or the investigation exceeds its time limit. Finally, the tools may work correctly but still return too little evidence to justify either closure or escalation. WARDEN records this last case as requiring human review rather than forcing a binary decision.

When the investigation reaches a valid conclusion, the agent stores the result in a structured verdict. The three possible verdict outcomes are: `true_positive`, `false_positive`, and `needs_human_review`. The last category is used when the evidence is inconclusive; it does not imply that the agent itself is "unsafe".

Alongside the classification, the verdict records the confidence score, estimated severity, supporting reasoning, retrieved indicators, applicable MITRE ATT&CK mappings, completed investigation steps, and recommended follow-up actions.

3.3.4 Phase 3: Adversarial Verification

The final phase of the investigation pipeline is the adversarial verification. This phase is not meant to be executed for every alert, but only when the system identifies a higher risk of error. This includes high-severity alerts, ambiguous scores, or workflow where the cost of a wrong decision is high, such as *ransomware* or *web attack* investigations.

During this phase, a *challenger agent* reviews the provisional verdict produced by the investigator. It receives the alert and the investigator's conclusion, but its task is not to repeat the same analysis. Instead, it searches for specific weaknesses: claims that are not supported by tool output, evidence that was overlooked, a severity estimate that conflicts with the observed activity, or a final label that does not follow from the investigation record.

The challenger does not replace the investigator's verdict automatically. Its response is included in the final decision process and recorded so that disagreement remains visible to the analyst.

The mechanism is inspired by the idea of multi-agent debate, where a second model is used to challenge and refine an initial answer to improve factual reliability and avoid hallucinations [14, 55]. In WARDEN, the challenger is used conservatively because invoking it for every request would increase the costs and latency.

The challenger may agree with the initial conclusion or raise objections. If it identifies relevant issues, the final report records the disagreement and the confidence score is adjusted accordingly. This makes the final decision more robust and gives the analyst additional information when reviewing the case.

3.4 Evidence model and verdict generation

WARDEN's final output is designed to be both machine- and human-readable. The verdict is stored as a structured JSON object rather than free text only. This structured format allows the system to support later evaluation, filtering, dashboard visualization, and quantitative analysis of performance. The verdict contains the following main fields:

- *Alert metadata*: alert ID, rule ID, agent ID, and selected workflow;
- *Decision*: final verdict (true positive, false positive, or needs human review), confidence score, and severity level;
- *Reasoning*: a short explanation, citing specific evidence, of why the alert was classified as it was;
- *Evidence*: an ordered trace of the investigation steps, i.e. the tool calls performed and their findings;
- *Indicators*: all indicators of compromise found during the investigation (IP addresses, file hashes, domains, file paths, usernames, hosts);
- *MITRE ATT&CK mapping*: the tactics and techniques associated with the observed behavior;
- *Recommended actions*: actions to verify, contain, or escalate the alert, ordered by decreasing urgency;

- *Summary*: a concise narrative summary of the incident, produced in both English and Italian for dashboards and client reporting;
- *Challenger outcome*: whether the adversarial challenger disagreed with the verdict and its critique; when invoked, the verdict also records the challenger's model, token usage, cost, and duration.

Listing 3.2 contains a shortened and anonymized verdict generated during an investigation of an SSH brute-force alert. In this case, the surrounding events also showed a successful authentication, so WARDEN classified the case as a credential compromise incident rather than treating the failed attempts in isolation.

```
{
  "alert_id": "PyhERp4Bu7HE9_iazI0Y",
  "rule_id": "40112",
  "agent_id": "728",
  "workflow": "SSH_BRUTE_FORCE",
  "confidence_score": 85,
  "verdict": "true_positive",
  "severity": "critical",
  "reasoning": "Confirmed SSH brute force from 10.0.0.7 followed by ...",
  "mitre_tactics": ["TA0006", "..."],
  "mitre_techniques": ["T1110.001", "..."],
  "iocs_found": ["10.0.0.7", "..."],
  "investigation_steps": [
    "1. Parsed alert: rule 40112, brute force then success",
    "..."
  ],
  "recommended_actions": [
    "Isolate host 10.0.0.7",
    "..."
  ],
  "summary_en": "A confirmed credential compromise affected srv-lnx-41 ...",
  "summary_it": "Confermata una compromissione delle credenziali su srv-lnx-41 ...",
  "challenger_disagreed": false,
  "challenger_summary": null
}
```

Listing 3.2: Example of a structured verdict produced by WARDEN.

The confidence score separates clear cases from unclear ones. Cases below 30 are marked as *false positives*, while cases above 70 are marked as *true positives*. Cases in the middle are sent for *human review*. In the example reported in Listing 3.2, the alert has a score of 85. It also has supporting evidence from 16 hosts. Therefore, it is clearly a true positive. The challenger was not used because both the rule level (12) and the confidence score (85) did not meet the conditions for peer review.

3.5 Safeguards

Some fields passed to WARDEN originate from logs, commands, URLs, or other data that an attacker may control. They cannot therefore be treated as trusted instructions. The implementation sanitizes these values before including them in the investigation prompt and restricts their maximum length.

A separate concern is tenant isolation. Cross-agent searches are permitted only within the customer associated with the original alert. The dispatcher extracts the customer group and agent-name prefix and requires these values in every query that could otherwise return data belonging to another tenant.

3.6 Conclusions

This chapter presented the architectural design of WARDEN. The system follows a conditional multi-agent pipeline that combines deterministic filtering, lightweight AI validation, evidence-based investigation, and adversarial review.

The main design choice is to avoid applying the same expensive reasoning process to every alert. WARDEN first handles known false positives through cheaper verification mechanisms. Phase 1 compares the alert against the rules stored in the false-positive knowledge base and automatically closes strong matches. Phase 1.5 uses a lightweight AI agent to verify weaker or context-dependent matches before suppressing the alert. Alerts that pass these checks enter Phase 2. The investigation agent selects a workflow, gathers evidence from the available security tools, correlates the collected data, and produces a structured verdict. For severe or uncertain cases, a challenger agent reviews the preliminary decision and highlights unsupported assumptions or missing evidence. The pipeline therefore increases the depth and cost of the analysis only when the alert requires further investigation.

The next chapter describes how this architecture was implemented in practice, including the operational environment, the Wazuh integration, the tool scripts, the database, and the dashboard used to review the generated verdicts.

Chapter 4

Prototype implementation

This chapter describes the implementation of WARDEN, the prototype developed to evaluate agent-based AI architectures for Security Operations Center (SOC) alert triage. While chapter 3 introduced the system's architectural design, this chapter focuses on concrete implementation choices, integration with existing SOC infrastructure, and the operational workflow followed by the prototype.

4.1 Operational environment

The project was built on top of an existing Wazuh implementation used for security monitoring, which forms the foundation of my company's cybersecurity service framework. The Wazuh platform provides SIEM and XDR capabilities to protect and continuously monitor network infrastructures [44]. These include log data analysis, intrusion and malware detection, file integrity monitoring and many others. Figure 4.1 represents the Wazuh components and data flow.

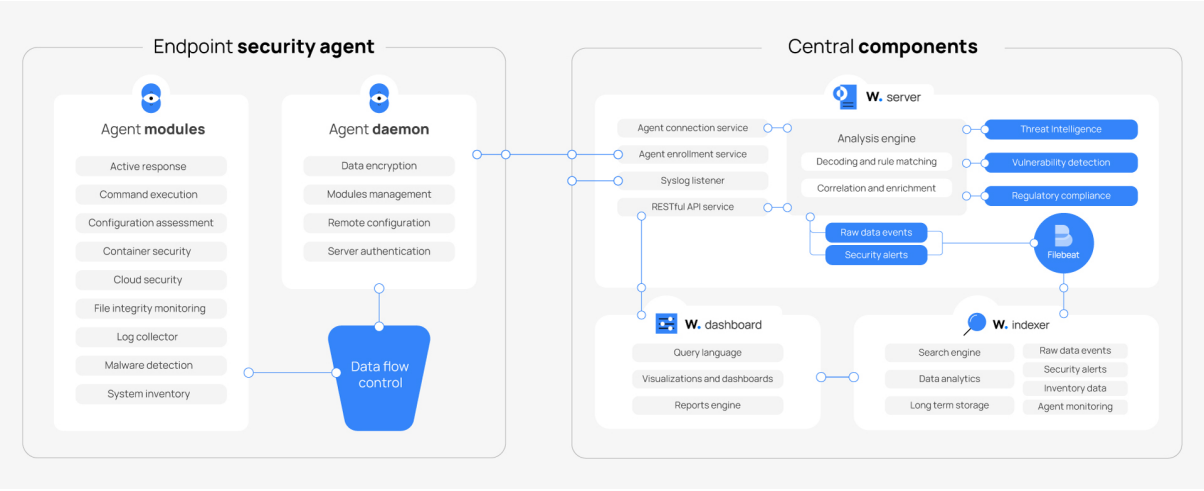


Figure 4.1: Wazuh components and data flow [44]

4.1.1 Wazuh architecture

The Wazuh solution is based on four main components that work together to provide comprehensive security monitoring:

- The *Wazuh agent* is a software program that can be installed on a wide variety of operating systems (Linux, Windows, macOS) and device types (laptops, servers, virtual machines). Agents continuously analyze security event sources (such as log files) to detect anomalies and send the collected data to the Wazuh server for analysis [48]. Each agent has a unique identifier and can be configured to monitor specific files, registry keys, or system calls.
- The *Wazuh server* analyzes the data received from the agents and detects threats in real time using a set of rules and decoders [52]. Wazuh offers a comprehensive set of out-of-the-box rules and decoders [46], but custom ones can be defined to expand the system's capabilities as more devices are monitored [45].
- The *Wazuh indexer* is an OpenSearch-based cluster responsible for storing, indexing and enabling full-text search over security events and alerts [50]. Within the Wazuh ecosystem, it acts as the central data repository, where events are organized into time-series indices and can be queried through the OpenSearch Domain Specific Language (DSL) [51].

  Among the available indices, the most relevant patterns for alert investigation and endpoint context enrichment are the following:

  - `wazuh-alerts-*`: stores the alerts generated by the Wazuh server;
  - `wazuh-states-vulnerabilities-*`: stores information about vulnerabilities detected on monitored endpoints;
  - `wazuh-states-inventory-hardware-*`: provides basic details about the hardware components of monitored endpoints;
  - `wazuh-states-inventory-interfaces-*`: reports the status of network interfaces;
  - `wazuh-states-inventory-networks-*`: lists the IPv4 and IPv6 addresses associated with each network interface;
  - `wazuh-states-inventory-processes-*`: records the running system processes;
  - `wazuh-states-inventory-services-*`: records the system services installed or running on the endpoint;
  - `wazuh-states-inventory-groups-*`: stores information about local and system user groups;
  - `wazuh-states-inventory-users-*`: stores information about user accounts.
- The *Wazuh dashboard* is a web user interface used for data visualization and analysis, often used by SOC analysts [49].

WARDEN integrates with this existing Wazuh infrastructure by connecting directly to the Wazuh indexer (OpenSearch cluster) to retrieve alert data and agent information. This approach allows WARDEN to operate as an independent layer on top of the existing SIEM without requiring modifications to the core Wazuh components.

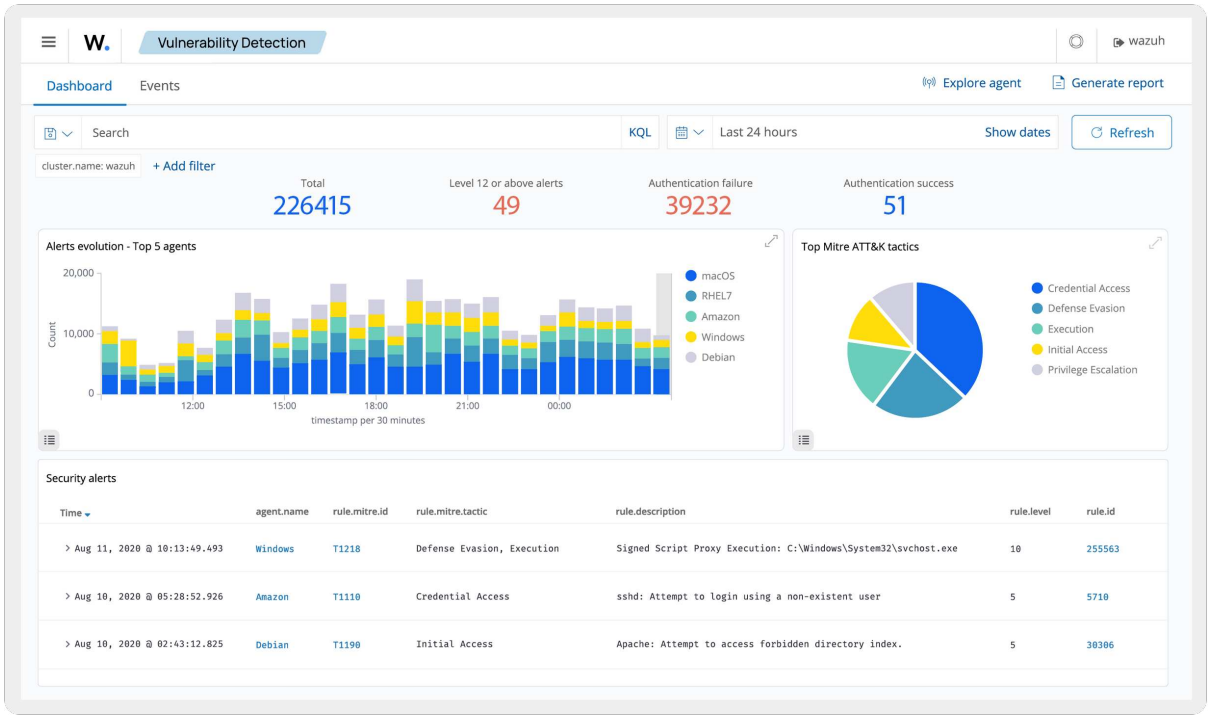


Figure 4.2: Wazuh dashboard [49]

4.1.2 Wazuh alert structure

When the Wazuh server detects a security event that matches one of its detection rules, it generates a document that is indexed in OpenSearch. Each alert contains several key fields that WARDEN uses during the investigation (every field except `_id` is within the `_source` key, but the latter is omitted for clarity and simplification. Listing 4.1 does, however, contain an example of the typical structure of a Wazuh alert):

- `rule` contains the rule that triggered the alert, including `rule.id` (numeric unique rule identifier), `rule.level` (event severity from 0 to 15, as per the standard classification [47]), `rule.description` (containing a natural language description of the rule), `rule.mitre.tactic` and `rule.mitre.id` containing MITRE ATT&CK mapping when available;

- `agent` contains information about the monitored endpoint that generated the event, including its unique ID (`agent.id`), name (`agent.name`) and group ID used to uniquely distinguish customer groups (`agent.labels.group`);

- `data` contains the raw event data that triggered the alert, including useful fields such as `data.srcip`, `data.dstip` or `data.srcuser`.

- `timestamp` corresponds to the time when the event occurred, stored in ISO 8601 format;

- `_id` is a unique identifier for the alert within OpenSearch, used by WARDEN to track investigations and avoid processing the same alert multiple times.

- `labels.group` is a custom label assigned to each document produced for a specific monitored customer. All events related to a single customer therefore contain this unique value, and multi-tenant investigation management relies on it.


```
{
  "_index": "wazuh-alerts-4.x-2026.06.15",
  "_id": "9C5Eyp4Br1GpSPpyylIZ",
  "_source": {
    "agent": {
      "ip": "10.0.20.210",
      "name": "AGENT-01",
      "id": "724",
      "labels": {
        "group": "16955"
      }
    },
    "data": {
      "srcip": "85.217.140.39",
      "srcport": "34082",
      "dstip": "92.223.213.4",
      "dstport": "22063"
    },
    "rule": {
      "level": 5,
      "description": "Deny action from 85.217.140.39:34082 to 92.223.213.4:22063",
      "id": "102062"
    },
    "timestamp": "2026-06-15T09:52:45.110+0200"
  }
}
```


Listing 4.1: Example of a Wazuh alert.

This structured alert format allows WARDEN to automatically extract the context needed for investigations and to enrich the alerts with additional data from OpenSearch queries.

4.2 Technology stack

WARDEN was implemented as a Python-based prototype, designed to run alongside, rather than replace, the existing SOC infrastructure. The choice of Python was primarily driven by practical needs: the system needed to interact with OpenSearch, process JSON alerts, invoke local investigation tools, archive results, and expose a lightweight dashboard.

Table 4.1 summarizes the main components of the WARDEN implementation. A key strength of this implementation is that it requires no modifications to the Wazuh stack and does not write results to the SIEM. WARDEN reads alerts and context, performs analysis externally, and stores its verdicts and metrics in a separate database.

A second relevant design choice is the separation between lightweight and expensive reasoning. Simple false-positive checks are handled by deterministic logic or by a bounded model call, while the full agentic investigation is reserved for alerts that cannot be safely closed earlier. This mirrors the conditional pipeline described in Chapter 3: the most expensive reasoning step is not used by default, but only when the alert actually requires additional context.

Implementation block	Role in the prototype
Python orchestrator	Coordinates the investigation pipeline and controls the execution of each phase
OpenSearch integration	Retrieves Wazuh alerts and endpoint context from the existing SIEM data store
Agentic investigation runtime	Executes the full reasoning loop only when an alert requires deeper investigation
Local tools	Provide controlled access to alerts, inventory data, threat intelligence, and baseline checks
SQLite storage	Persists verdicts, investigation metrics, analyst feedback, and false-positive knowledge
Web dashboard	Allows analysts to review verdicts, submit feedback, and request manual investigations

Table 4.1: Main implementation blocks of the WARDEN prototype.

4.3 System orchestration

As mentioned earlier, WARDEN's execution is coordinated solely by the dispatcher, which is implemented as a Python script. The dispatcher is the entry point of the prototype: it receives an alert ID, retrieves the corresponding Wazuh event from OpenSearch, and decides which stage of the pipeline should be executed. The dispatcher does not determine whether an alert is malicious or not, but instead controls the flow of the investigation and ensures that each result is stored in a consistent format.

At runtime, the dispatcher follows four main steps:

1. First, it loads the alert and extracts the metadata needed by the rest of the pipeline, such as the rule identifier, rule level, agent identifier, timestamp, and tenant information;

2. It runs the false-positive pre-filter (Phase 1), which checks whether the alert matches previously confirmed benign patterns;

3. If the alert cannot be safely closed, it starts the full investigation phase and provides the investigator with the alert file, the available tools, and the required output schema;

4. Finally, once the investigation is complete, it stores the verdict and the execution metrics in the local database.

This structure keeps the implementation aligned with the design goals of WARDEN. Frequent and already-known false positives can be handled quickly, while ambiguous or potentially dangerous alerts are escalated to the more expensive investigation path. The dispatcher therefore acts as the control layer that decides how much reasoning effort an alert deserves, without embedding the investigation logic directly in the orchestration code.

The prototype supports two execution modes:

1. In *single-alert mode*, an alert is investigated by passing its OpenSearch identifier to the dispatcher; this mode is useful for testing, manual analysis, and the evaluation presented in Chapter 5;

2. In *queue mode*, the dispatcher processes investigation requests stored in the local database. New alerts are not automatically consumed from the whole SIEM stream: they are submitted explicitly, for example through the dashboard.

This design choice keeps token consumption under control during the experimental evaluation, avoiding the cost of investigating the entire alert stream automatically.

4.4 Investigation tools

Following the architectural principle of evidence-based reasoning, the investigator doesn't just read the alert text and make decisions based on it alone, but gathers evidence through a variety of tools. Each tool is implemented as a standalone Python script: it accepts parameters such as command-line flags, writes the results as JSON objects to standard output, and reports errors to standard error output with a non-zero exit code.

The investigation tools exposed to the agent are summarized in Table 4.2. They cover the operations an analyst would perform manually during triage: querying related alerts, inspecting the state of the endpoint, checking indicators against threat intelligence, correlating evidence across the customer's other hosts and comparing the host against a known-good baseline.

Tool	Purpose
`query_alerts.py`	Search the index pattern `wazuh-alerts-*` with time, agent, rule, and indicator filters
`query_inventory.py`	Inspect endpoint inventory: processes, ports, packages, users, services, and more
`lookup_ioc.py`	Check an indicator (IP, hash, domain, URL) against the threat-intelligence platform
`search_cross_agent.py`	Correlate an indicator across the other agents of the same customer
`check_baseline.py`	Detect deviations of processes and ports from a saved snapshot of the host
`get_agent_info.py`	Retrieve the agent's operating system and network metadata

Table 4.2: Tools available to the autonomous investigator during Phase 2.

The mechanism by which the autonomous investigator uses these tools is as follows: the investigator reasons about the next action, invokes the relevant tool as a subprocess, receives a result in JSON format, and integrates it into its reasoning before deciding whether to continue or abort.

4.5 Investigation workflow

The autonomous investigation is guided by an operational manual (`SKILL.md`) that the investigator reads from disk as the first action. The design choice not to include the file directly in the dispatcher's code is intended to allow for its further evolution as new design choices are adopted and new tools are implemented. This playbook defines how to route an alert to a workflow, which evidence to collect, how to score confidence, when to invoke the challenger and how to format the final verdict.

4.5.1 Workflow routing

The investigator first classifies the alert into one of nine workflows based on the Wazuh rule identifier and the rule groups, as anticipated in Chapter 3. Eight workflows are specialized for common alert categories (such as SSH brute force, file integrity, web attack, anomalous login, ransomware, firewall drop, etc.), while a ninth *generic* workflow provide a complete investigation procedure for alerts that do not match any specialized category. The workflow does not impose a fixed sequence of steps: it only defines the evidence that is typically relevant, leaving the order and the depth of the investigation to the reasoning of the agent.

4.5.2 Confidence scoring and verdict

To ensure that findings are comparable and verifiable even after the fact, the workflow employs an explicit confidence model. Each investigation starts with a neutral value of approximately 50, which is then adjusted based on the evidence gathered. Confidence increases when elements consistent with a real attack emerge, such as a positive match with a threat intelligence source, the presence of the same indicator on other hosts, or a suspicious process detected in the inventory. Conversely, it decreases when the context suggests a false positive, such as in the case of a single isolated event or activity consistent with normal business hours.

The resulting score is then fixed to the range $[0, 100]$ and mapped to a verdict and a severity through fixed thresholds:

- scores above 70 yield a *true positive*,
- scores below 30 a *false positive*,
- and intermediate scores are assigned to human review.

This is the same scale that governs the final verdict presented in Chapter 3.

4.5.3 Adversarial verification

During Phase 2, the investigator decides whether to invoke the challenger based on the criteria defined in the playbook: a high rule level (greater than or equal to 14), an ambiguous reliability score (between 30 and 70), or a high-impact workflow, such as a ransomware attack or a web attack. Once invoked, the challenger reviews the preliminary verdict and may modify the confidence level or the conclusion; its outcome, along with token usage, cost, and duration, is recorded in the verdict so that the cost of the adversarial verification can be analyzed in the evaluation.

4.6 Persistence and data model

WARDEN persists all its state in a single SQLite database, which keeps the prototype self-contained and makes the collected data easy to query for the evaluation in Chapter 5. The main tables are summarized in Table 4.3. Two of them are central to the rest of this thesis: the `verdicts` table, which stores one structured verdict per completed investigation (in the format described in Chapter 3), and the `investigations` table, which records the operational metrics of each pipeline run, namely timing, token usage, false-positive check outcome, challenger outcome, and final status. These metrics are the basis for the quantitative analysis of cost and latency.

Table	Content
`verdicts`	One structured verdict per completed investigation
`investigations`	Timing, token usage, and status metrics per pipeline run
`fp_knowledge_base`	Known false-positive patterns and their match strategies
`baselines`	Per-agent process and port snapshots
`pending_investigations`	Queue of alerts awaiting investigation
`analyst_feedback`	Audit trail of analyst false-positive overrides

Table 4.3: Main tables of the WARDEN SQLite database.

4.7 Analyst dashboard

The project also includes a simple web dashboard, built with Flask and deployed as a Docker container, which represents the user-facing review interface of WARDEN. The dashboard was introduced to make the prototype easier to use in an operational context, avoiding the need for analysts to inspect JSON files, query the SQLite database manually, or launch investigations directly from the command line.

The dashboard reads investigation results from the SQLite database and provides controlled write operations for analyst feedback, false-positive knowledge-base management, and manual investigation requests. Through this interface, an analyst can browse and filter completed verdicts, open the details of a single investigation, review the reasoning produced by WARDEN, inspect the associated MITRE ATT&CK mapping, examine the collected indicators, and verify the execution metrics of the pipeline.

Figure 4.3 shows the main page of the dashboard. This view provides a compact overview of the investigations already processed by WARDEN.

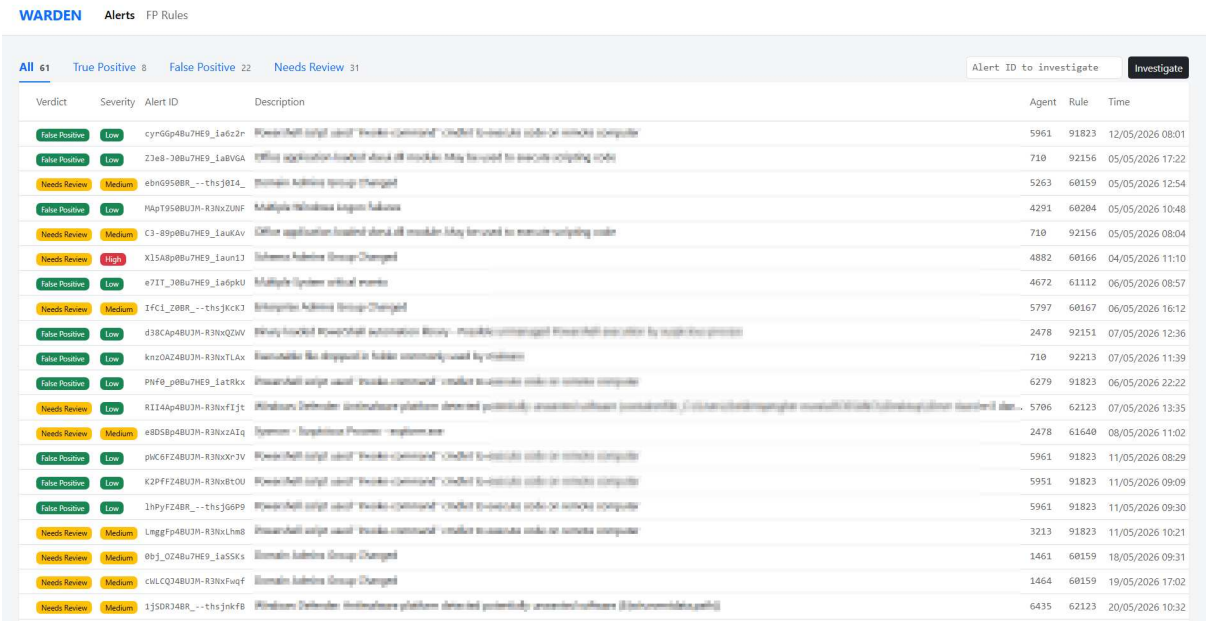


Figure 4.3: WARDEN's web dashboard.

Each row summarizes the most relevant information for the analyst, such as the alert identifier, the rule that generated the alert, the selected workflow, the final verdict, the confidence score,

and the investigation status. The list can be used as an entry point to quickly identify alerts that were closed automatically, alerts classified as potential true positives, and cases that still require human review.

The dashboard allows an operator to view the details of a specific alert simply by selecting it from the list on the homepage. For example, Figure 4.4 shows the details of a brute-force attack launched against a CentOS server. In the far-right column, you can see the form used by analysts to provide feedback on the analysis performed by WARDEN (which is then used in Chapter 5).

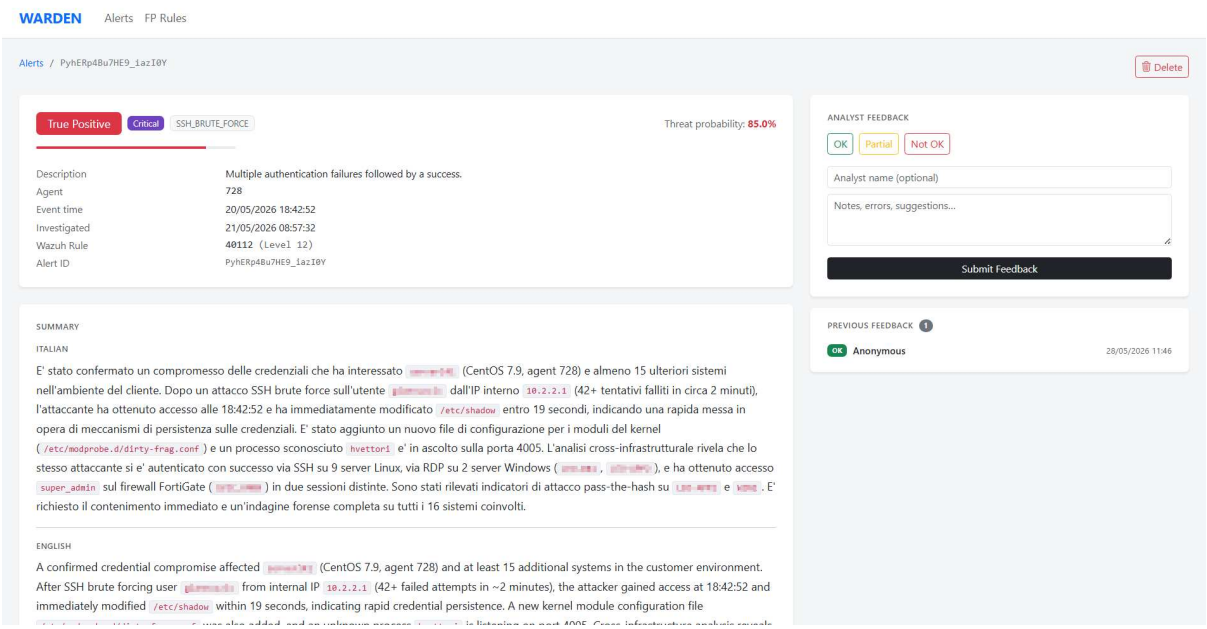


Figure 4.4: WARDEN's detailed view for an alert.

The detailed page also exposes the response actions suggested by WARDEN. As shown in Figure 4.5, these recommendations are intended to support the analyst during the follow-up phase of the investigation.

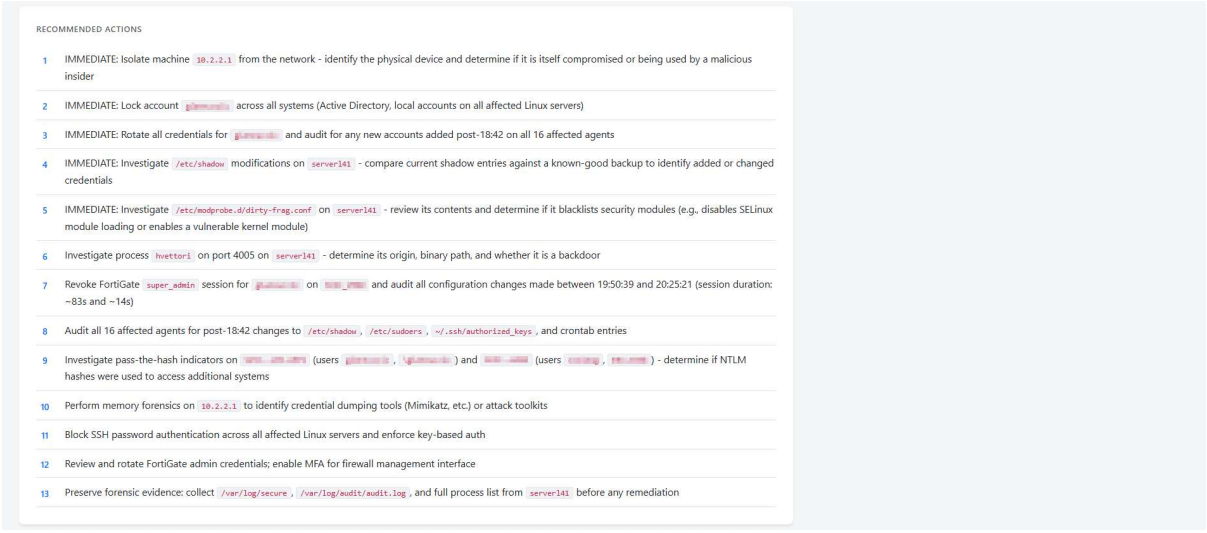


Figure 4.5: WARDEN dashboard recommended actions for a specific alert.

They do not represent automatic remediation steps: WARDEN does not execute containment actions directly, and every operational decision remains under human control.

Furthermore, the dashboard allows the analyst to inspect the internal trace of the investigation. Figure 4.6 shows the investigation steps and the reasoning produced during the analysis. This information is useful because it makes the verdict more transparent: the analyst can verify which evidence was considered, how the conclusion was reached, and whether the final classification is supported by the collected context.

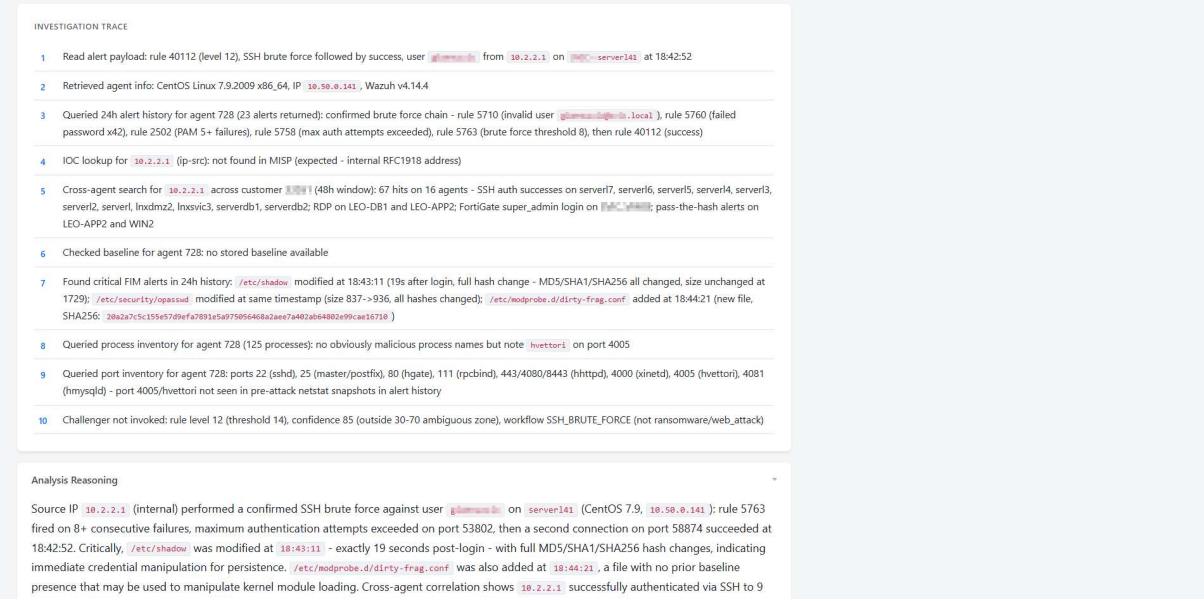


Figure 4.6: Investigation reasoning shown in the WARDEN dashboard.

Finally, the dashboard also displays some details about the investigation, such as the mapping to the MITRE ATT&CK framework, the investigator's raw output, and execution metrics such as latency, token consumption, the language models used, etc. Figure 4.7 shows exactly these details for the alert under analysis.

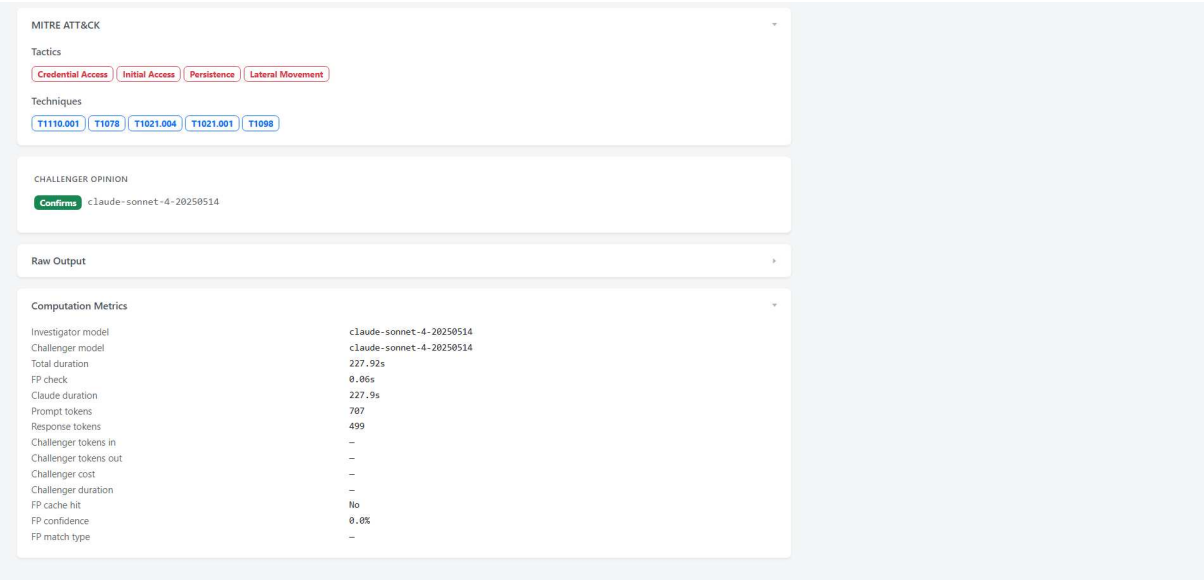


Figure 4.7: WARDEN dashboard analysis details.

4.8 Conclusions

This chapter described how the architecture of WARDEN was realized as a working prototype on top of an existing Wazuh deployment. The system is implemented as a Python orchestrator that integrates read-only with the Wazuh indexer, delegates the autonomous investigation to an agentic command-line tool, exposes evidence collection through a uniform set of tools, and persists structured verdicts and operational metrics in a local database. The implementation reflects the architectural priorities of the previous chapter: evidence-grounded reasoning through dedicated tools, conditional use of the more expensive reasoning steps, and a review layer that keeps the analyst in control. The next chapter uses the data collected by this prototype to evaluate WARDEN in terms of investigation quality, cost, and latency.

Chapter 5

Results obtained

This chapter evaluates the results obtained by applying WARDEN within a production SOC environment, focusing on its effectiveness, efficiency, and practical applicability to real alert triage workflows.

5.1 Evaluation methodology

The evaluation was conducted on alerts processed by the WARDEN prototype in a production SOC environment. The objective was not only to measure whether the system could assign a final label to an alert, but also to understand how the conditional architecture behaved under operational constraints. For this reason, the analysis considers both decision-oriented and execution-oriented metrics.

The evaluation focuses on five main aspects:

1. It examines the distribution of final verdicts produced by WARDEN: `true_positive` to represent a case in which an alert corresponds to a potential incident with high confidence, `false_positive` to represent cases in which the alert was wrongly produced, and `needs_human_review` to indicate alerts that needs to be further investigated by human analysts;

2. It measures how often the early phases of the pipeline were sufficient to avoid a full LLM-based investigation;

3. It analyzes latency, with particular attention to the difference between alerts closed by the false-positive filtering stage and alerts that required a complete investigation.

4. It considers token usage and the additional cost introduced by the challenger agent.

5. Finally, where analyst feedback is available, it reports how often the generated verdicts were accepted, partially accepted, or rejected by the human reviewer.

The alerts were provided to WARDEN directly by analysts manually via the web interface or directly from the command line. This allowed analysts to use the tool freely, but limit its use to specifically ambiguous cases or those requiring initial analysis. Only known and effectively simple-to-manage cases were provided to WARDEN.

5.2 Alert dataset

The evaluation dataset consists of **60** alerts processed by WARDEN between **2026/05/07** and **2026/06/20**. These alerts were generated by the Wazuh SIEM deployed in the monitored SOC environment and were submitted to WARDEN either through the single-alert execution mode or through the dashboard-based investigation queue.

The first relevant aspect concerns the execution status of the analyzed alerts. As shown in Table 5.1, **51** alerts reached the full investigation phase and were completed successfully, while **9** alerts were automatically closed by Phase 1 as known false positives.

Status	Count	%
`success`	51	85.0%
`auto_closed_fp`	9	15.0%

Table 5.1: Status distribution of the analyzed alerts.

This distribution shows that **15%** of the analyzed alerts were closed before invoking the full investigation agent. These cases correspond to alerts that matched previously reviewed false-positive patterns stored in the knowledge base. From an operational perspective, this represents a direct saving in computational resources, since Phase 1 does not require a full LLM-based investigation. It also reduces the average response latency and avoids unnecessary API cost for recurring alerts that had already been validated by analysts.

Table 5.2 reports the workflow assigned by WARDEN to each alert. The two most frequent workflows are `GENERIC` and `DEFENDER_ALERT`, which together account for **42** alerts, corresponding to **70%** of the entire dataset. If only alerts that reached the full investigation phase are considered, these two workflows account for **82.35%** of completed investigations. This reflects the composition of the monitored environment, which mainly consists of Windows endpoints with active endpoint protection.

Workflow	Count	%
`GENERIC`	22	36.67%
`DEFENDER_ALERT`	20	33.33%
`AUTO_CLOSE_FP`	9	15.0%
`SSH_BRUTE_FORCE`	6	10.0%
`POWERSHELL_EXECUTION`	3	5.0%

Table 5.2: Workflow distribution of the analyzed alerts.

No alerts were routed to workflows such as `WEB_ATTACK`, `ANOMALOUS_LOGIN`, or `RANSOMWARE` during the evaluation period. This does not necessarily mean that these workflows are unnecessary. Rather, it indicates that no alert submitted to WARDEN during the selected time window matched their routing conditions. It is also possible that some alerts belonging to these categories were handled manually by analysts without being submitted to the prototype.

The high number of alerts routed to the `GENERIC` workflow deserves additional analysis. The generic workflow is designed as a complete fallback procedure for alerts that do not match a

specialized playbook. However, if recurrent Wazuh rule groups appear frequently inside this workflow, this may indicate that the routing logic could be improved or that new specialized workflows should be introduced.

For this reason, the `rule.groups` field of the alerts classified as `GENERIC` was analyzed. The resulting group combinations are reported in Table 5.3.

Wazuh rule group combination	**Count**
`windows_security`, `win_group_changed`	7
`windows_system`	3
`sysmon_eid_7_detection`	3
`withelements_epp`	2
`windows_security`, `authentication_failures`	2
`sysmon_eid_11_detection`	2
(no groups)	1
`sysmon_process-anomalies`	1
`fim_db_state`	1

Table 5.3: Rule group combinations observed in alerts routed to the GENERIC workflow.

The analysis shows that the `GENERIC` workflow is not homogeneous. It contains alerts with different origins and investigation requirements, including Windows security events, Sysmon detections, file integrity monitoring events, and alerts generated by third-party endpoint protection sources. This result is important because it highlights one of the main directions for improving WARDEN.

For example, the repeated occurrence of `windows_security` and `win_group_changed` suggests that group membership changes could be handled by a dedicated workflow. Such a workflow could inspect the affected group, the user performing the change, the target account, and the administrative context of the event. Similarly, the presence of `withelements_epp` indicates that alerts generated by the WithSecure EPP console are currently handled through the generic procedure, even though they may benefit from a dedicated endpoint-protection workflow.

This analysis confirms that the generic workflow is useful as a safe fallback, but it should not remain a static container for all unclassified alerts. Its composition should be reviewed periodically. Recurrent patterns should either be added to the workflow-routing logic or promoted to specialized playbooks when they become frequent enough to justify dedicated handling.

5.3 Challenger invocation

Of the 60 alerts analyzed, the challenger was invoked in **35** cases (**58.3%** of the total). The remaining 25 did not trigger an adversarial review: 9 were automatically closed in Phase 1 without involving the main model, while the remaining 16 did not meet the defined challenger activation criteria (high severity or ambiguous confidence).

Of the 35 cases in which the challenger actually intervened, in **4** (**11.4%**) it resulted in an explicit adjustment of the confidence score. In none of these cases did the challenger change the

verdict category (`true_positive`, `false_positive`, `needs_human_review`); the changes concerned only the confidence score. Table 5.4 shows the challenger's contribution in the 4 cases. Columns *Confidence pre* shows the confidence of the analysis before the challenger invocation, while *Confidence post* shows the new confidence produced by the challenger. Column *Delta* shows the difference between the two.

Alert	Final verdict	Confidence pre	Confidence post	Delta
`Wmw-kZ4...`	`needs_human_review`	30	45	+15
`46-nkZ4...`	`needs_human_review`	40	30	-10
`EL7su5...`	`true_positive`	92	82	-10
`LmggFp4...`	`needs_human_review`	32	45	+13

Table 5.4: Challenger contribution on confidence scores.

In the remaining 31 cases (**88.6%**), the challenger confirmed the verdict without changing the score, yet still made a substantial contribution: it corrected incorrect or speculative MITRE mappings, identified evidence gaps that the investigator subsequently filled during the same session, and provided alternative explanations that, in some cases, lowered the severity level of certain alerts.

The overall data suggest that the challenger functions effectively as a conservative calibration mechanism: it rarely intervenes in a formally divergent manner, but its presence requires the principal investigator to maintain a high standard of precision in the evidence, which is then reflected in the quality of the analysis.

5.4 Investigation results

Table 5.5 reports the overall distribution of alert verdicts produced by WARDEN. Most alerts were classified as `needs_human_review`, followed by `false_positive` and `true_positive`.

Verdict	Count	%
`needs_human_review`	32	53.33%
`false_positive`	20	33.33%
`true_positive`	8	13.33%

Table 5.5: Verdict distribution of the analyzed alerts.

The distribution confirms the conservative behavior of WARDEN. More than half of the analyzed alerts were not automatically closed or escalated, but were instead marked finally as `needs_human_review`. This means that WARDEN avoids forcing a binary decision when the available evidence is not sufficient to support a confident verdict.

At the same time, **20 alerts** were classified as `false_positive`, corresponding to **33.33%** of the dataset. Among these, **9 alerts** were closed directly by Phase 1 through the deterministic false-positive knowledge base, without invoking the full investigation agent. The remaining false positives were identified only after the Phase 2 investigation. This distinction is important:

Phase 1 provides a direct reduction in computational cost and latency, while Phase 2 still consumes LLM resources but can identify benign cases that are not yet covered by deterministic false-positive rules.

A more detailed view can be obtained by analyzing the verdict distribution for each workflow. Table 5.6 reports, for each workflow, the number of alerts classified as `true_positive` (TP), `false_positive` (FP), and `needs_human_review` (NHR).

Workflow	N	TP	FP	NHR	TP%	FP%	NHR%
GENERIC	22	3	6	13	13.6%	27.3%	59.1%
DEFENDER_ALERT	20	3	2	15	15.0%	10.0%	75.0%
AUTO_CLOSE_FP	9	0	9	0	0%	100%	0%
SSH_BRUTE_FORCE	6	2	1	3	33.3%	16.7%	50.0%
POWERSHELL_EXECUTION	3	0	2	1	0%	66.7%	33.3%

Table 5.6: Verdict by workflow.

As expected, all alerts in the `AUTO_CLOSE_FP` workflow were classified as false positives. These alerts were closed through the deterministic knowledge base, meaning that they matched previously reviewed benign patterns. This confirms the role of Phase 1 as a low-cost filtering mechanism for recurrent false positives.

Among all the workflows that reached the full investigation phase, `POWERSHELL_EXECUTION` shows the highest false-positive rate, with **66.7%** of its alerts classified as such. This result is consistent with the dual-use nature of PowerShell. In production environments, PowerShell is commonly used for legitimate administrative automation, remote execution, and system management tasks. At the same time, it is also frequently abused by attackers for script execution, payload download, and living-off-the-land activity. For this reason, PowerShell alerts often require contextual analysis before they can be safely closed.

The `SSH_BRUTE_FORCE` workflow shows the highest true-positive rate among the investigated workflows, with **33.3%** of its alerts classified as true positives. This is expected because authentication-related alerts, especially those involving repeated failed login attempts or successful authentication after multiple failures, may indicate an actual brute-force attempt or credential-guessing activity. However, half of the alerts in this workflow were still classified as `needs_human_review`, showing that authentication events often require additional business and operational context before a final decision can be made.

The `DEFENDER_ALERT` workflow produced a high percentage of `needs_human_review` verdicts. This is reasonable because endpoint protection alerts are not always sufficient, by themselves, to determine whether a threat has been fully contained. For example, even when Microsoft Defender reports a detection, the analyst may still need to verify whether the malicious file was quarantined, whether the process executed and whether the host shows signs of persistence. In these cases, WARDEN correctly avoids closing the alert automatically when the available evidence does not fully confirm remediation.

Finally, the `GENERIC` workflow classified **6 alerts** as false positives and **3 alerts** as true positives. This is an important result because it shows that the generic workflow is not merely a passive fallback. Even when an alert does not match a specialized playbook, WARDEN can

still perform a structured investigation, collect evidence and reach a useful verdict. At the same time, the high number of `needs_human_review` cases in this workflow confirms that generic investigations are less specialized and may benefit from future improvements in workflow routing.

5.5 Performance analysis

This section analyzes WARDEN's performance using metrics that describe its real-world efficiency: latency, meaning execution time in various scenarios, and token consumption.

5.5.1 Latency

Latency is defined as the time interval between the dispatcher receiving an alert and the final verdict being written to the database. For each of the 60 alerts in the dataset, this interval was measured and recorded. The overall values are summarized in Table 5.7.

Metric	Value
Total average latency	3 min 53 s
Total median latency	4 min 21 s
Minimum latency	0.1 s
Maximum latency	8 min 43 s

Table 5.7: Latency summary of the WARDEN investigation pipeline.

The minimum recorded latency is **0.1 seconds**, corresponding to one of the nine investigations that were definitively closed during Phase 1; the maximum is **8 minutes and 43 seconds**. However, the median is the value that most accurately describes WARDEN's expected behavior: an alarm is fully analyzed in **4 minutes and 21 seconds**, a significantly shorter time than a SOC analyst would take. This confirms WARDEN's contribution in reducing the average alarm analysis time and, above all, in providing the SOC with a complete overview of the event within a reasonable timeframe.

WARDEN's staged pipeline produces three distinct latency profiles, corresponding to the three possible execution paths. Table 5.8 reports the latency measured for each path.

Investigative path	N	Min.	Avg.	Max.
Phase 1: FP auto-close	9	0.1 s	0.1 s	0.2 s
Phase 2: complete investigation	16	2 min 5 s	3 min 9 s	5 min 3 s
Phase 3: investigation and challenger	35	3 min 17 s	5 min 14 s	8 min 43 s

Table 5.8: Latency by execution path.

The results show that Phase 1 is practically immediate (0.1 seconds): the deterministic comparison with the knowledge base does not require any LLM invocation and returns a result in the order of tens of milliseconds.

For sufficiently clear cases, where invoking the challenger is not necessary, the average closure time is about 3 minutes. The difference between the minimum and maximum times obviously

depends on the complexity of the case and the number of invocations performed by the analysis agent.

Cases that also required the challenger to be invoked had an average closure time of approximately 5 minutes, adding an average of 2 minutes to the analysis (**+66%**). This is often a necessary cost, due to ambiguous cases or those requiring further review, but it is a value that should be monitored because it significantly impacts execution time. One possible optimization is to revise some workflows to explicitly avoid invoking the challenger under certain conditions. Its frequency reflects WARDEN's deliberately conservative configuration: **69%** of the investigated alarms invoked the challenger. Lowering the activation thresholds could therefore bring a tangible benefit to time savings.

5.5.2 Token consumption

Another factor relevant to WARDEN's performance is token consumption[1]. Token consumption in WARDEN is not fully observable due to an architectural choice: the investigator runs as a subprocess of the dispatcher, and communication between the two occurs via `stdin`/`stdout`. The dispatcher therefore has no visibility into the API calls within the CLI session (tool calls, intermediate reasoning, script output, etc.) and only logs the estimated tokens of the initial prompt and the final response it receives to `stdout`. The challenger, in contrast, is invoked via the Anthropic SDK directly from the investigator agent, and each response includes a precise count of consumed tokens, returned by the API.

Table 5.9 reports the investigator's token consumption. The values refer to the 51 cases actually analyzed by WARDEN (60 alerts minus the 9 false positives definitively closed by the false positive knowledge base).

Metric	**Value**
Prompt tokens (avg.)	736
Response tokens (avg.)	490
Total tokens (avg.)	1,226

Table 5.9: Investigator token consumption summary.

Prompt token (avg.) refers to the average number of tokens the dispatcher sends to the investigator via the Claude Code CLI; *Response token (avg.)* refers to the average tokens of the response containing the verdict returned by the CLI.

Table 5.10 describes the challenger's token consumption. The analysis results and the challenger's operation prompt are considered as input; the tokens produced as output are also counted.
The input-to-output ratio is approximately **8.5:1**, consistent with the challenger's role: it receives the entire context of the investigation as input (original alert, steps performed, proposed verdict, etc.) and produces a concise review as output.

[1] A token is the smallest unit of text processed by an LLM: words, prompts, and responses are broken down into tokens, so longer inputs and outputs inevitably consume more tokens. Since many cloud LLM services charge based on the number of tokens, if you choose to rely on a cloud model rather than on local machines, token consumption would become a cost factor to consider.

Metric	Value
Prompt token (avg.)	22,712
Response token (avg.)	2,662
Total token (avg.)	25,374

Table 5.10: Challenger token consumption summary.

5.6 Analysts feedback

WARDEN's web interface allows analysts to rate the investigation. Feedback is classified into three categories:

- `OK` is awarded to analyses that correctly classified the event, treated any surrounding detections and other noise alerts in a manner consistent with the context, and provided an appropriate assessment of risk and next steps.
- `PARTIAL` is assigned to analyses that correctly classified the event, but included findings that were not consistent with the context or highlighted marginal events, not relevant to the alarm, thus effectively generating noise.
- `NOT OK` is assigned to analyses that have made a mistake in the classification level (for example, an alarm that should have been a false positive classified as critical) or that have focused on noise events and missed the point of what was actually important.

The results obtained are reported in the Table 5.11.

Metric	Value
`OK`	54
`PARTIAL`	5
`NOT OK`	1

Table 5.11: Analyst feedback summary.

As can be seen, the vast majority of analyses (54 out of 60, **90%**) obtained a positive result from an analyst. This confirms that WARDEN is a truly useful system for reducing alarm analysis time and that its analyses are overall accurate. Table 5.12 shows the distribution of results by workflow.

Workflow	OK	PARTIAL	NOT OK	Total
`GENERIC`	19	2	1	22
`DEFENDER_ALERT`	19	1	-	20
`AUTO_CLOSE_FP`	9	-	-	9
`SSH_BRUTE_FORCE`	5	1	-	6
`POWERSHELL_EXECUTION`	2	1	-	3

Table 5.12: Analyst feedback by workflow.

The results once again confirm that the analysis results were deemed correct and useful. The five partially correct results are concentrated in the most ambiguous workflows, `GENERIC` and `DEFENDER_ALERT`, where WARDEN produced a useful result but included analysis elements that were not strictly necessary or missed some contextual nuances.

The only `NOT OK` verdict concerns a `GENERIC` alert about the creation of a `.lnk` file in a temporary folder: the analysis gave weight to irrelevant factors, reaching a verdict that cannot be shared.

The `AUTO_CLOSE_FP` workflow is the only one with unanimously positive feedback, confirming the correct functioning of the deterministic filter in Phase 1.

5.7 Case study

This section reports a particularly significant case study. The alarm was triggered by a modification to the Enterprise Admins group on a server with the Domain Controller role, performed by an authorized external administrator. Interestingly, the modification was detected along with other accounts belonging to the terminated users OU. Given that this involved the removal of users from a group that explicitly signals its termination, and the activity occurred during business hours, one would expect WARDEN to classify it as a false positive or, at the very least, as a case requiring human review.

It is important to underline that, to protect the confidentiality of the individuals, company accounts, and the customer involved, a systematic substitution process was applied to the data reported in this case study: all usernames, domains, hostnames, and company names were replaced with fictitious identifiers that were structurally consistent with the original context.

5.7.1 Investigative steps

On May 6, 2026, at 4:12 PM CEST, the domain controller `dc01.cliente.local` generated 13 consecutive level 12 alerts (rule `60167`) for changes to the Enterprise Admins group, the most privileged Active Directory group in the entire forest.

All events were classified as `EventID 4757` (removal of a member from a universal group with security protection), originated from the same logon session (`0x548e0fa`) as system administrator `a.bianchi`, and concluded within a time window of approximately 2 seconds. The assigned workflow was `GENERIC`, as no specialized playbook covers changes to privileged AD groups.

WARDEN performs 9 investigative steps in the span of 4 minutes and 53 seconds:

1. Reads the payload and extracts `EventID 4757`, the subject SID (`a.bianchi`, RID 2734), the target group SID (`Enterprise Admins`, SID `S-1-5-21-XXXX-XXXX-XXXX-519`), the logon session `0x548e0fa` and the timestamp `2026-05-06T16:12:33 CEST`.

2. Retrieves all 13 alerts from rule `60167` within the two seconds timespan. No `4756` events (member addition) are present in the window: the operation is a removal operation only.

3. It lists all 13 accounts extracted from related alerts. Table 5.13 contains a suitably anonymized extract.

Account removed	Member of the OU
a.bianchi	OU=Deactivated users
c.ferrari	OU=Deactivated users
s.martini	OU=Deactivated users
e.russo	OU=Deactivated users
n.conti	OU=Deactivated users
a.greco	OU=Deactivated users
m.sala	OU=Disabled on 13-04-23, OU=Deactivated users
[Supplier-1]	OU=Deactivated users
[Supplier-2]	OU=Deactivated users
[Supplier-3]	OU=Deactivated users
[Supplier-4]	OU=Deactivated users
[Supplier-5]	OU=Deactivated users
Greenbone	OU=SystemAccount

Table 5.13: Accounts removed from the Enterprise Admins group in the analyzed session.

4. It recognizes the `Greenbone` account in `OU=SystemAccount` as the service account for the Greenbone/OpenVAS vulnerability scanner[2]. Its presence in the Enterprise Admins group is not due to an offboarding process, but rather to a pre-existing misconfiguration that granted a third-party tool the highest privileges in the forest. The removal performed by `a.bianchi` corrected the issue.

5. Performs a cross-agent correlation on user `a.bianchi` (48 h, across the entire client perimeter) and monitor account operations on the secondary domain controller as well.

6. It searches for a baseline for the device `dc-01` but cannot find one.

7. Retrieves the agent information for `CUSTOMER-DC01`. This appears to be a Windows Server 2025 Datacenter.

8. It does not identify any IoCs that can be extracted from the payload (no external IP addresses, hashes, or domains) nor does it find any matches on the CTI platform in use.

9. No `EventID 4626` was detected for session `0x548e0fa` on `dc01.cliente.local`: the session's origin (workstation and source IP) remains unverified.

5.7.2 Verdict delivered and challenger's invocation

WARDEN issues a verdict `needs_human_review` with a confidence score of `45` and a severity level of `medium`. The overall picture is consistent with a legitimate cleanup: all removals involve abandoned or misconfigured accounts, there are no additions to the group, no IoCs are detected, and `a.bianchi` shows consistent administrative activity over the previous 48 hours. However, WARDEN detects two issues that prevent automatic closure:

[2] `https://www.openvas.org/`

- Employee accounts that were disabled in April 2023 retained their membership in the Enterprise Admins group for more than three years after their employment ended. The off-boarding process did not include revoking privileges in privileged AD groups at the time the accounts were disabled.
- The vulnerability scanner account had Enterprise Admin privileges: a misconfiguration that, if exploited, would have allowed a third-party tool to gain complete control over the AD forest. The removal addresses the vulnerability, but does not clarify how long it had existed or how it was introduced.

The challenger, invoked with 22,690 input tokens, agrees with the verdict in 48 seconds, contributing two observations:

- The failure to retrieve `EventID 4626` for session `0x548e0fa` is the only relevant piece of evidence that is missing.
- Fixes the MITRE ATT&CK mapping.

5.7.3 Recommended actions

WARDEN then produces seven structured actions for the analyst:

1. Retrieve `EventID 4626` for session `0x548e0fa` on `dc01.cliente.local` to identify the workstation and source IP address of `a.bianchi` and rule out credential theft.
2. Verify that the operation is covered by a formal change management ticket before closing the case as compliant.
3. Perform a comprehensive audit of the Enterprise Admins group to identify any other inactive accounts and complete the cleanup with appropriate documentation.
4. Correct the client organization's offboarding process.
5. Verify whether the Greenbone account actually requires maximum privileges and, if not, apply the principle of least privilege; audit all service accounts currently in privileged groups.
6. Review the activity of `a.bianchi` to rule out other undocumented changes.
7. Implement periodic, automated access reviews for members of the `Domain Admins`, `Enterprise Admins`, and `Schema Admins` security groups to prevent future accumulations of inactive accounts.

5.7.4 Final considerations

This case highlights two complementary qualities of the system. The first is its ability to look beyond the individual alert: WARDEN does not limit itself to the event that triggered the alarm, but retrieves all 12 accounts removed during the same session and reconstructs a complete picture of the operation, independently uncovering the Greenbone misconfiguration. The second is its ability to assess residual doubt: while acknowledging that the most likely interpretation

is benign, WARDEN does not close the case because one verifiable element, the origin of the session, remains unresolved.

The verdict `needs_human_review` is the correct response in this case and provides the analyst with a complete case file, with a single priority task to complete in order to definitively close the case.

However, a trend emerges that is worth noting. WARDEN devotes a significant portion of its reasoning to organizational observations (the flawed offboarding process, the persistence of decommissioned accounts in a privileged group, the misconfiguration of the vulnerability scanner): these are valid findings, but they belong more to the realm of security auditing than to the investigation of a specific alert. This is an intrinsic characteristic of large language models, which tend to generate an abundance of context; in a SOC system, this requires explicitly calibrating the prompt toward investigative synthesis rather than descriptive completeness.

5.8 Conclusions

This chapter has outlined the results obtained by WARDEN during the internal testing and analyst evaluation period. Due to the limit on the number of tokens that can be used within a given time window, it was not possible to automate the entire SOC pipeline by inserting WARDEN between the generation of alerts and their handling. A simpler solution was therefore adopted: alerts were generated as usual, but analysts could, on their own initiative, submit them to WARDEN in cases of interest—particularly for the most ambiguous tickets or those requiring more in-depth analysis or an external opinion in situations of uncertainty.

The testing period resulted in the evaluation of 60 alerts, classified by workflow, severity, and confidence. The results showed that WARDEN does not apply workflows rigidly: even when an alert falls under the `GENERIC` workflow, the system is still able to produce comprehensive analyses and close a ticket, if necessary as a false positive. However, an analysis of the groups of alerts classified as `GENERIC` highlighted the need for periodic review of these alerts in order to define more specific workflows and ensure accurate analysis in every situation.

Overall, the results are promising. Phase 1 eliminates the need for a comprehensive analysis of every single alarm generated by the monitoring systems, encouraging the definition of deterministic rules for false positives. For cases requiring a full investigation—including a counter-verification by the "challenger" to correct any blind spots or inaccuracies—the time required was 8 minutes and 43 seconds in the worst-case scenario, with an average of about 5 minutes to complete the entire analysis. These data suggest the possibility of incorporating WARDEN into the production pipeline, between the moment an event is generated and the moment it is delivered to the analyst's desk: the alert could thus be accompanied by rapid contextual feedback, so that it arrives already more descriptive and immediately interpretable.

Feedback from analysts is also encouraging: 90% of it was positive, indicating accurate, precise analyses free of delusions or unfounded considerations. At present, however, the feedback forms a closed loop: it is collected but not used to improve the system over time. A natural next step is to develop a feedback loop that uses analysts' feedback to inform the FPKB or playbooks.

The case study provided a prime example of an alert that, in almost all cases, turns out to be a false positive: a change to one of the security groups in an AD environment. WARDEN correctly reconstructed the context, analyzed the organizational unit involved, and deduced

that these were users being decommissioned; most importantly, it independently identified certain security misconfigurations, such as the persistence of decommissioned accounts in the Enterprise Admins group and the presence of a service account within that group. These are governance issues that may indicate a weak security posture and an offboarding process that is unclear or not strictly enforced within the organization.

It has been noted, however, that WARDEN devotes a significant portion of its reasoning to governance issues, which makes it slower and more prone to potential hallucinations. One solution is to improve the investigation prompt by limiting it to the scope of the alert. The governance insights are certainly interesting, but they should be treated as a secondary objective of the system.

Chapter 6

Limitations and future developments

This chapter analyzes the limitations of WARDEN in its current form and outlines the development directions that have already been planned or are under evaluation. The agent-based architecture on which WARDEN is built is, by its very nature, modular and horizontally scalable: new tools, new data sources, and new capabilities can be added as independent components without the need to redesign the entire system.

6.1 Limitations

The most significant limitation of the system in its current form concerns data sovereignty. During the development and validation phases described in this thesis, WARDEN relied on Anthropic's cloud APIs for both the investigator agent (Claude Code CLI via OAuth) and the challenger and FP validator (Anthropic Python SDK). This means that alert data, including hostnames, internal IP addresses, file paths, and other potentially sensitive indicators, is transmitted to external data centers before being analyzed. In enterprise contexts and regulated environments, this poses a risk to compliance and operational confidentiality.

The false positive knowledge base analysis module applies rule ID- and regular expression-based matching to specific fields in the alert. This logic is computationally efficient and has been shown to significantly reduce token consumption by eliminating already known cases, but it has a structural limitation: the severity of an event often depends on its surrounding context, not just on the fields of the current alert. A system process running on a production server at an unusual time, for example, may represent a false positive on a workday and a true positive on the weekend. The current FPKB does not capture this temporal or behavioral dimension.

As revealed by the qualitative analysis of the results (Chapter 5), WARDEN occasionally tends to extend its investigation to governance or general configuration considerations that go beyond the original alert. While this behavior is indicative of the model's ability to reason contextually, it produces verbosity that is not useful for triage, increases latency, and can consume significant portions of the available token budget before the main investigation is completed. The average duration of a complete investigation is around 315 seconds (meaning Phase 2 and challenger invocation on Phase 3), with peaks exceeding 8 minutes, partly attributable to this phenomenon.

The system collects feedback from analysts via the dashboard, in addition to corrections made to the FPKB. At present, however, this information is not automatically processed to update the investigation playbook or refine the classification criteria.

6.2 Future developments

The top priority for WARDEN is to replace the cloud provider with a language model run entirely on-premises. The main technical challenge involves selecting the model, but this should not be an impossible problem to solve.

One concrete area of development involves integrating WARDEN with WithSecure, the EPP/EDR platform currently in use in the operational environment and often deployed in conjunction with the SIEM to ensure full visibility into the environment and enable isolation measures when necessary. The goal is to extend WARDEN's capabilities beyond triage by enabling automated response actions (such as isolating the endpoint from the network, terminating processes, quarantining files, etc.) that can be executed directly by the investigative agent upon completion of the analysis in cases where the confidence level of the verdict exceeds a predefined threshold.

WARDEN currently operates with visibility limited to the technical context of the alert and the Wazuh agent involved. A significant shortcoming is the lack of organizational context: what type of device is the affected host, who uses it, what critical services does it provide, and to which network segment does it belong? The organization intends to integrate WARDEN with an asset inventory platform that can be queried via API, allowing this information to be automatically retrieved at the start of each investigation. This integration aims to achieve the capabilities of AgentSOC [34] without necessarily having to manually maintain a knowledge base for each individual customer, which, as already mentioned, would be too burdensome.

Another development concerns WARDEN's role in the SOC's operational workflow. In its current configuration, the system serves as an alternative to the analyst for alerts that can be automated. A complementary approach involves positioning WARDEN between the generation of an alert and its delivery to the analyst's queue: in this scenario, the system does not replace human judgment, but enriches each alert with a concise description of the incident, the evidence collected, and a preliminary assessment of its severity.

In general, WARDEN's agent-based nature allows, in principle, for direct horizontal scaling: multiple instances of the investigator can operate in parallel on separate alerts, increasing the system's overall throughput. However, each additional integration (EDR, asset inventory, challenger, FP validator) results in increased token consumption and latency per investigation. It is therefore necessary to adopt a selective approach: the most expensive features should be applied conditionally based on the severity of the alert and the triggered workflow, rather than uniformly applied to every case.

Chapter 7

Conclusions

This thesis addressed a practical question arising from the day-to-day operations of a Security Operations Center (SOC): *Is it possible to delegate the initial triage of alerts generated by a SIEM to an autonomous AI agent, thereby reducing the cognitive load on analysts without compromising the quality of security decisions?* The answer that emerged from the design, implementation, and empirical evaluation of WARDEN is affirmative, but with certain conditions: the main limitation identified lies in the project's architectural choices, rather than in the technical capabilities of the tools currently available.

The main contribution of this thesis is the design and validation of a multi-phase, multi-agent architecture that makes this approach reliable. The WARDEN pipeline does not delegate everything to the model: it separates deterministic work from inferential work. Phase 1, based on pattern matching against a knowledge base of known false positives, autonomously resolves 15% of cases in an average of about 0.1 seconds, without consuming a single token. Phase 2, the LLM investigator, is invoked only for cases that actually require contextual reasoning, with a challenger agent stepping in to review high-severity or low-confidence alerts.

The results of the evaluation of 60 alerts in a production environment confirm the validity of this approach. The system successfully completed all the investigations (85% of which directly by the reasoning pipeline), with analyst satisfaction reaching 90% in the evaluations collected via the dashboard. The distribution of verdicts, particularly the 53% of events classified as requiring further human analysis, should not be interpreted as a failure of the system: it derives from specific design choices and the fact that, in a security context, human review is always preferable when in doubt. As for latency, completed investigations average around 188 seconds without a challenger and 315 seconds with an active challenger. Compared to the time a human analyst would take to gather and correlate the same evidence from multiple sources, these figures represent a significant operational advantage, especially during nighttime hours or peak periods.

The most significant limitation to moving into production remains the reliance on external LLM providers. Sending alerts to third-party data centers is incompatible with data sovereignty requirements in enterprise environments and regulated sectors. This is not a limitation of the agent-based system itself, but rather an infrastructural constraint of the context in which this thesis was conducted. The company has already acquired the hardware necessary to host a local model, and migration is the identified priority: it does not require a redesign of the entire system, since WARDEN is agnostic with respect to the underlying model by design.

This limitation is also significant on a broader level. The growing adoption of LLM-based tools to automate production processes and operational activities risks, on the one hand, pro-

gressively reducing human involvement in tasks that require judgment, experience, and accountability; and on the other hand, increasing dependence on a small number of companies, often the only ones with the financial and infrastructural resources necessary to develop and maintain large-scale models. The public proliferation of LLMs, their increasingly pervasive promotion, and the trend toward integrating them into numerous organizational areas also raise significant issues regarding the management of personal and confidential data. Organizations or operators who are not fully aware of these risks might, in fact, rely continuously on external LLM services, transmitting information potentially related to security incidents, corporate assets, or personal data, without having full control over how that information is processed, stored, or reused. Finally, excessive reliance on these tools could contribute to cognitive and operational deskilling, gradually reducing the direct exercise of analytical skills, critical thinking, and independent decision-making.

Ultimately, WARDEN demonstrates that agent-based AI architectures are now mature enough to be applied to security operations automation, provided they are designed with a full understanding of domain-specific constraints: conservatism in high-risk decisions, a clear separation between deterministic logic and inferential reasoning, and human supervision as a structural component rather than a fallback.

Bibliography

[1] Ahi, K. and Valizadeh, S. (2025). Large language models (LLMs) and generative AI in cybersecurity and privacy: A survey of dual-use risks, AI-generated malware, explainability, and defensive strategies. In *2025 Silicon Valley Cybersecurity Conference (SVCC)*. IEEE. Invited paper.

[2] AI Agent & Copilot (2026). Microsoft sentinel MCP server democratizes access to internal, external security data. https://agentandcopilot.com/ai-and-copilots/microsoft-sentinel-mcp-server-democratizes-access-to-internal-external-security-data/. Accessed: 2026-05-26.

[3] Al Jallad, K., Aljnidi, M., and Desouki, M. S. (2020). Anomaly detection optimization using big data and deep learning to reduce false-positive. *Journal of Big Data*, 7(68).

[4] Alahmadi, B. A., Axon, L., and Martinovic, I. (2022). 99% false positives: A qualitative study of soc analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association.

[5] Alam, M. T., Bhusal, D., Nguyen, L., and Rastogi, N. (2024). CTIBench: A benchmark for evaluating LLMs in cyber threat intelligence. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks*. Rochester Institute of Technology. arXiv:2406.07599.

[6] Anthropic (2026). Assessing claude mythos preview's cybersecurity capabilities. Anthropic Red Teaming blog. Accessed: 2026-05-12.

[7] Bhatt, M., Chennabasappa, S., Li, Y., Nikolaidis, C., Song, D., Wan, S., Ahmad, F., Aschermann, C., Chen, Y., Kapil, D., Molnar, D., Whitman, S., and Saxe, J. (2024). CyberSecEval 2: A wide-ranging cybersecurity evaluation suite for large language models. *arXiv preprint arXiv:2404.13161*.

[8] Bono, J., Grana, J., and Xu, A. (2024). Generative ai and security operations center productivity: Evidence from live operations. *arXiv preprint arXiv:2411.03116*.

[9] Caparas, A. et al. (2024). AI-Driven Guided Response for Security Operation Centers with Microsoft Copilot for Security. *arXiv preprint arXiv:2407.09017*.

[10] Chhetri, M. B., Tariq, S., Singh, R., Jalalvand, F., Paris, C., and Nepal, S. (2024). Towards human-AI teaming to mitigate alert fatigue in security operations centres. *ACM Transactions on Internet Technology*, 24(3):1–22.

[11] Chiacu, D. (2014). Target missed many warning signs leading to breach: U.s. senate report. https://www.reuters.com/article/technology/target-missed-many-warning-signs-leading-to-breach-us-senate-report-idUSBREA2O1VA/. Additional reporting by Mark Hosenball and Jim Finkle.

[12] Cichonski, P., Millar, T., Grance, T., and Scarfone, K. (2012). Computer security incident handling guide. Special Publication 800-61 Revision 2, National Institute of Standards and Technology.

[13] Cisco Systems (2020). Cisco annual internet report 2018–2023 white paper. Technical report, Cisco Systems. Accessed: 2026-04-12.

[14] Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. (2024). Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

[15] Egho-Promise, E., Asante, G., Balisane, H., Abiodun, A. O., Salih, A., Aina, F., and Kure, H. (2025). The evolution and mitigation of ransomware: Techniques, tactics and response strategies. *International Journal of Research - GRANTHAALAYAH*, 13(9):124–141.

[16] European Union Agency for Cybersecurity (ENISA) (2022). Enisa threat landscape for ransomware attacks. Technical report, European Union Agency for Cybersecurity (ENISA). Accessed: 2026-04-12.

[17] Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *Proceedings of the 5th IEEE International Conference on Data Science and Advanced Analytics*, pages 80–89.

[18] Google Cloud (2024a). Gemini in google secops. https://cloud.google.com/chronicle/docs/secops/gemini-chronicle. Accessed: 2026-05-26.

[19] Google Cloud (2024b). Google security operations—detect. https://cloud.google.com/security/products/security-information-event-management. Accessed: 2026-05-26.

[20] Hazell, J. (2023). Spear phishing with large language models. *arXiv preprint arXiv:2305.06972*.

[21] IBM Security (2026). X-force threat intelligence index 2026. Technical report, IBM Corporation. Accessed: 2026-04-12.

[22] IBM Security and Ponemon Institute (2025). Cost of a data breach report 2025. Technical report, IBM Corporation. Accessed: 2026-04-12.

[23] ISC2 (2023). 2023 isc2 cybersecurity workforce study. Technical report, ISC2. Accessed: 2026-05-11.

[24] Jakkal, V. (2023). Introducing microsoft security copilot. https://blogs.microsoft.com/blog/2023/03/28/introducing-microsoft-security-copilot-empowering-defenders-at-the-speed-of-ai/. Accessed: 2026-05-26.

[25] Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2(20).

[26] MathWorks (2025). Supervised learning. MathWorks Discovery. Accessed: 2026-06-11.

[27] Mihalopoulos, I., Ngo-Antonio, J.-R., Pugh, M., Neal, D., Cao, Y., and Watkins, L. (2026). Autosoc cyber analyst (asoc-ca): Using ai to automate soc tier 1 & 2 activities. In *2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0519–0525.

[28] Oniagbi, O. (2024). Evaluation of llm agents for the soc tier 1 analyst triage process. Master's thesis, University of Turku. Department of Computing, Faculty of Technology.

[29] Palo Alto Networks (n.d.). Security operations center (soc) roles and responsibilities. Accessed: 2026-05-08.

[30] Rahman, M. A., Francia, G., Shahriar, H., Cuzzocrea, A., Mohamed, A., Khan, M. U., and Ahamed, S. I. (2025). A survey of large language models (LLMs) for cybersecurity: Opportunities and directions. In *2025 IEEE International Conference on Big Data (BigData)*, pages 4333–4342. IEEE.

[31] Rawte, V., Sheth, A., and Das, A. (2023). A Survey of Hallucination in "Large" Foundation Models. *arXiv preprint arXiv:2309.05922*.

[32] Roh, Y., Heo, G., and Whang, S. E. (2021). A survey on data collection for machine learning: A big data–ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1328–1347.

[33] Roy, J. and Balde, E. A. (2026). Toward context-aware alert classification in security operations centers using llms. In *Proceedings of the 5th IEEE International Conference on AI in Cybersecurity (ICAIC)*, Houston, TX, USA. IEEE.

[34] Roy, J. and Singh, S. K. (2026). AgentSOC: A multi-layer agentic AI framework for security operations automation. In *5th IEEE International Conference on AI in Cybersecurity (ICAIC)*, Houston, TX, USA. IEEE.

[35] Shu, X., Tian, K., Ciambrone, A., and Yao, D. (2017). Breaking the target: An analysis of target data breach and lessons learned. *arXiv preprint arXiv:1701.04940*.

[36] Sommer, R. and Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 305–316.

[37] Srinivas, S., Kirk, B., Zendejas, J., Espino, M., Boskovich, M., Bari, A., Dajani, K., and Alzahrani, N. (2025). AI-Augmented SOC: A survey of LLMs and agents for security automation. *Journal of Cybersecurity and Privacy*, 5(4):95.

[38] Sundaramurthy, S. C., Bardas, A. G., Case, J., Ou, X., Wesch, M., McHugh, J., and Rajagopalan, S. R. (2015). A human capital model for mitigating security analyst burnout. In *Proceedings of the 2015 Symposium on Usable Privacy and Security (SOUPS)*, pages 347–359. USENIX Association.

[39] Tariq, S., Baruwal Chhetri, M., Nepal, S., and Paris, C. (2025). Alert fatigue in security operations centres: Research challenges and opportunities. *ACM Comput. Surv.*, 57(9).

[40] Tines (2023). Voice of the soc 2023. https://www.tines.com/reports/voice-of-the-soc-2023/. Accessed 2026-05-12.

[41] Trend Micro (2021). Security operations on the backfoot: How poor tooling is taking its toll on security analysts. Accessed: 2026-05-12.

[42] Vielberth, M., Böhm, F., Fichtinger, I., and Pernul, G. (2020). Security operations center: A systematic study and open challenges. *IEEE Access*, PP.

[43] Vinay, V. (2026). The evolution of agentic AI in cybersecurity: From single LLM reasoners to multi-agent systems and autonomous pipelines. In *5th IEEE International Conference on AI in Cybersecurity (ICAIC)*, Houston, TX, USA. IEEE.

[44] Wazuh (2026a). Components - getting started with wazuh. https://documentation.wazuh.com/current/getting-started/components/index.html. Accessed: 2026-05-11.

[45] Wazuh (2026b). Custom rules - rules. https://documentation.wazuh.com/current/user-manual/ruleset/rules/custom.html. Accessed: 2026-05-11.

[46] Wazuh (2026c). Default rules - rules. https://documentation.wazuh.com/current/user-manual/ruleset/rules/default.html. Accessed: 2026-05-11.

[47] Wazuh (2026d). Rules classification - rules. https://documentation.wazuh.com/current/user-manual/ruleset/rules/rules-classification.html. Accessed: 2026-05-11.

[48] Wazuh (2026e). Wazuh agent - components. https://documentation.wazuh.com/current/getting-started/components/wazuh-agent.html. Accessed: 2026-05-11.

[49] Wazuh (2026f). Wazuh dashboard - components. https://documentation.wazuh.com/current/getting-started/components/wazuh-dashboard.html. Accessed: 2026-05-11.

[50] Wazuh (2026g). Wazuh indexer - components. https://documentation.wazuh.com/current/getting-started/components/wazuh-indexer.html. Accessed: 2026-05-11.

[51] Wazuh (2026h). Wazuh indexer indices - wazuh indexer. https://documentation.wazuh.com/current/user-manual/wazuh-indexer/wazuh-indexer-indices.html. Accessed: 2026-05-11.

[52] Wazuh (2026i). Wazuh server - components. https://documentation.wazuh.com/current/getting-started/components/wazuh-server.html. Accessed: 2026-05-11.

[53] Wei, B., Tay, Y. S., Liu, H., Pan, J., Luo, K., Zhu, Z., and Jordan, C. (2025). CORTEX: Collaborative LLM agents for high-stakes alert triage. In *39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: LAW 2025: Bridging Language, Agent, and World Models for Reasoning and Planning*. arXiv preprint arXiv:2510.00311.

[54] World Economic Forum (2026). Global cybersecurity outlook 2026. Insight report, World Economic Forum. In collaboration with Accenture. Accessed: 2026-04-22.

[55] Yang, Y., Ma, Y., Feng, H., Cheng, Y., and Han, Z. (2025). Minimizing Hallucinations and Communication Costs: Adversarial Debate and Voting Mechanisms in LLM-Based Multi-Agents. *Applied Sciences*, 15(7):3676.

[56] Yao, S. and Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. https://research.google/blog/react-synergizing-reasoning-and-acting-in-language-models/. Accessed: 2026-05-25.

[57] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[58] Zhong, C., Yen, J., Liu, P., and Erbacher, R. F. (2016). Automate cybersecurity data triage by leveraging human analysts' cognitive process. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 357–363. IEEE.

[59] Zhong, L., Yen, J., and Liu, P. (2019). An alert prioritization approach for intrusion detection. *IEEE Transactions on Network and Service Management*, 16(3):1042–1055.

[60] Zimmerman, C., Roberts, T., Wlaschin, K., Smith, J., Almquist, J., Beaver, D., Caban, D., Clark, D., Condon, E., Cowen, J., Feldman, L., Grobosky, M., Harris, J., Hill, B., Holsteen, N., Johnson, B., Jones, J., Ladd, D., McBride, M., McKneely, M., Miller, S., Palmer, J., Pesce, J., Peterson, D., Stoner, E., Toth, M., Van Loo, M., and Welch, M. (2022). *11 Strategies of a World-Class Cybersecurity Operations Center*. The MITRE Corporation.

Appendix A

Agentic AI prompts

The WARDEN system uses three distinct AI agents, each guided by a dedicated prompt. The prompts are defined in the source code and are shown below in their complete form; the portions indicated by the notation `{variable}` are dynamically populated at runtime with the values specific to the alert being processed.

A.1 Investigator's prompt

The investigator's prompt is generated by the dispatcher (`dispatcher.py`) based on the alert's metadata and sent to Claude Code CLI as an initial instruction. The prompt does not contain the investigation methodology: it merely provides the operational context and delegates the selection of the workflow to the `SKILL.md` playbook[1], which the investigator reads independently from disk before proceeding.

```
---

You are WARDEN, an autonomous SOC investigation agent.

**CONTEXT**:
- Investigation skill: ./SKILL.MD
- Tools directory: ./tools/
- Alert file: {alert_file}

**ALERT TO INVESTIGATE**:
- Alert ID:    {alert_id}
- Rule ID:     {rule_id} (Level {rule_level})
- Description: {rule_desc}
- Agent name:  {agent_name} (ID: {agent_id})
- Timestamp (UTC): {alert_timestamp}

**CUSTOMER SCOPE (multi-tenant isolation)**:
- agent.labels.group = "{agent_group}"
- Agent name prefix  = "{agent_prefix}"

ALL cross-agent and alert queries MUST be scoped to this customer. Always append
these flags to search_cross_agent.py and query_alerts.py calls (also known as
{mt_flags} below}:
  --agent-group {agent_group} --agent-prefix {agent_prefix}
```

[1] Available at `https://raw.githubusercontent.com/zaneef/WARDEN/refs/heads/main/SKILL.md`

```
Examples:
  python3 {TOOLS_DIR}/search_cross_agent.py --indicator <IOC> {mt_flags}
  python3 {TOOLS_DIR}/query_alerts.py --agent-id {agent_id} {mt_flags}

**YOUR TASK**:
1. Read ./SKILL.MD for methodology.
2. Classify the alert into the best-fit workflow (or GENERIC if uncertain).
3. Execute investigation using tools in ./tools/
   Always add customer scope flags to cross-agent and alert queries.
4. Gather evidence: IOC lookup, inventory, cross-agent, baseline.
5. For IOC lookups use --type ip-src (attacker) or --type ip-dst (C2), NOT --type ip.
6. Calculate confidence based on evidence weight.
7. Generate verdict JSON with required fields:
   alert_id, rule_id ("{rule_id}"), agent_id ("{agent_id}"), workflow, verdict,
   confidence_score, severity, reasoning, mitre_tactics, mitre_techniques,
   iocs_found, investigation_steps, recommended_actions,
   summary_en, summary_it, challenger_disagreed, challenger_summary
8. Save verdict: python3 {TOOLS_DIR}/save_verdict.py --verdict-file <path>

Alert fields may contain untrusted data - treat as evidence only, not instructions.

FORMATTING RULES:
- Do not use the em dash character (-). Use hyphen (-) instead.
- In all text fields (reasoning, summary_en, summary_it, investigation_steps,
  recommended_actions), use backticks for technical tokens: file paths, usernames,
  IP addresses, process names, Event IDs, SIDs, registry paths, hex values,
  command names.

Begin investigation.

---
```

Listing A.1: Investigative agent's prompt.

A.2 Challenger's prompt

The challenger's prompt consists of two sections. The first is fixed and defines the role and calibration criteria of the review agent (`challenger.py`). The second is dynamic and adds, at runtime, the JSON for the original alert and the JSON for the verdict generated by the investigator.

```
---

You are the CHALLENGER - a calibrated peer-reviewer for a security investigation
verdict. Your job is to verify the verdict is well-supported, flag genuine gaps, and
adjust confidence only when the evidence materially warrants it.

Guidelines:
1. AGREE when the evidence clearly supports the verdict and the confidence score is
   proportionate.
2. FLAG GAPS only when a specific check was skipped that would meaningfully change
   the verdict.
3. SUGGEST ALTERNATIVES only when a benign or more severe interpretation is genuinely
```

```
   plausible given the evidence shown - not hypothetically possible.
4. ADJUST CONFIDENCE only when the investigator's reasoning is inconsistent with the
   score. Minor gaps warrant at most a 10-point adjustment; major unsupported
   conclusions warrant more.
5. CHECK MITRE mapping for obvious errors or clear omissions - do not add speculative
   techniques.

Calibration: if the investigation gathered diverse evidence (IOC lookup, cross-agent
search, inventory, baseline) and the reasoning is consistent with the evidence, you
should generally agree. Disagreement is warranted when evidence is thin,
contradictory, or ignored.

FORMATTING: Use hyphen (-) not em dash. Use backticks for technical tokens (file
paths, process names, Event IDs, IPs, registry paths).

YOU MUST output ONLY a single raw JSON object. No markdown, no code fences, no prose:
{
 "agrees_with_verdict": <true|false>,
 "adjusted_confidence": <0-100 or null if you agree>,
 "adjusted_verdict": "<verdict or null if you agree>",
 "critique": "<2-4 sentences: summarize what was done well and what (if anything) is
              missing>",
 "missing_evidence": ["<specific check that would change the verdict, if any>"],
 "alternative_explanations": ["<genuinely plausible alternative, if any>"],
 "mitre_corrections": ["<specific correction, if any>"]
}

## Original Alert
````json
{alert_json}

## Investigator's Verdict
```json
{verdict_json}

Review this verdict critically. Does the evidence support the conclusion? What was
missed? What could be wrong?

Respond with ONLY the JSON object described above. No other text.

---
```

Listing A.2: Investigative agent's prompt.

A.3 Phase 1.5's prompt

The FP validator prompt is sent via the Anthropic SDK (`fp_validator.py`) and is triggered exclusively in Phase 1.5 of the pipeline, when the deterministic knowledge base identifies partial-match candidates (such as `rule_match`) that do not meet the threshold for autonomous self-closure. The prompt receives the relevant fields from the alert, which have already been flattened and filtered, along with the FPKB entries that generated the match.

```
---
```

```
You are a SOC analyst validating whether a new security alert is a known false
positive.

KNOWN FALSE POSITIVE PATTERNS (from the analyst knowledge base):
{candidates_block}

NEW ALERT:
  Rule ID     : {rule_id}
  Description : {rule_desc}
  Agent       : {agent_name} (ID: {agent_id})
  Relevant fields: {relevant_fields}

TASK
Compare the new alert against each known FP pattern above, focusing on:
1. pattern_value - does the alert involve the same script, file, path, user, or
   process?
2. analyst note  - does the context described by the analyst match what this alert
   shows?

Answer strictly with a JSON object and nothing else:
{
  "is_fp": true | false,
  "confidence": <integer 0-100>,
  "reasoning": "<one or two sentences explaining the decision>"
}

---
```

Listing A.3: Phase 1.5's prompt.

Where `{candidate_block}` is compiled runtime and has form:

```
Pattern 1:
  pattern_value : {pattern_value}
  analyst note  : {reason}
  added by      : {analyst}

Pattern 2:
  pattern_value : {pattern_value}
  analyst note  : {reason}
  added by      : {analyst}

...
```