

Departement : STIC

Reference :

**Applied License in Information and Communication Sciences and
Technologies**

Option: RST

End of Studies Project

Autonomous SOC Based on Artificial Intelligence for L1 and L2 Automation

**Realized by: Mohamed Rayen Tayechi
Habib Kchaou**

Class: L3 RST B

**Supervisor(s): Mr. Amir Trigui
Ms. Takoua Kefi**

Host Company : Next Step

Academic year: 2025-2026

Dedication

I dedicate this work to those whose patience, support, and quiet encouragement have been a constant source of strength throughout this journey.

To my family: in loving memory of my father, Meher, for your enduring wisdom and the steadfast support that will always stay with me. My mother, Zahida, for your endless love and belief in me. My sister Khouloud, whose hard-earned experience as an ISET'COM alumna lit the way through these three years, and my sister Amira, who has been a steady guide throughout my academic journey.

To my friends, we carried each other forward through thick and thin. Thank you for your support, your patience, and for bringing relief and laughter when I needed it most. You have always been there for me, and that has made all the difference.

And then finally, to my professors and mentors who shaped my academic path with patience and dedication.

This success is as much yours as it is mine. Thank you for your incredible support throughout this journey.

Habib Kchaou

Dedication

I dedicate this modest work...

To my dear father **Bechir**, whose knowledge in the field of telecommunications and unwavering support have lit every step of this journey. Your wise advice, your patience, and your constant presence have made this work possible. No dedication could ever express the depth of my gratitude and my love.

To my dear mother **Samia**, my source of tenderness and strength. Your prayers, your unconditional love, and your quiet sacrifices have carried me through all these years. I owe you everything I am today.

To my brother **Zack** and my sisters **Emna** and **Fatma**, for their warm presence, their encouragement, and every moment we have shared. You are my strength and my refuge.

To my friends, those who have marked this journey with their loyalty, their good spirits, and their support through the hardest days and the best ones. Thank you for being there.

To the entire pedagogical and administrative staff of **ISSET'COM**, to every professor who passed on their knowledge with rigor and passion, and to every member of the administration whose dedication and availability accompanied us throughout these years of training — may this work reflect the quality of the education I have had the honor to receive within this distinguished institution.

With all my heart, thank you.

Rayen Tayechi

Acknowledgments

We would like to express our gratitude to the people who contributed to the completion of this end-of-studies project. The work presented in this report would not have taken its current shape without their guidance, their time, and the trust they placed in us throughout this journey.

First and foremost, we would like to express our heartfelt thanks to our supervisor at Next Step, Mr. Amir Trigui, for accompanying us from the first day of the internship to the final pages of this report. His availability, often at moments when we were stuck and unsure how to move forward, made the difference between a project that stalled and one that kept advancing. We are also grateful for the trust he placed in us throughout the internship, which allowed us to take ownership of our work and explore directions we would not have ventured into otherwise.

We are also deeply thankful to Ms. Takoua Kefi, whose mentorship, advice, and unwavering belief in our potential have been a tremendous source of motivation. Her support during our supervised project and throughout this end-of-studies project has left a lasting impact on both our academic path and professional mindset.

We extend our sincere appreciation to the members of the jury, for taking the time to evaluate our work and for the valuable insights and comments that will contribute to our growth and learning.

Finally, we thank all our professors and the entire pedagogical team of ISET'COM, whose dedication to teaching has supported us throughout our years of study.

This project reflects all the support, mentorship, and knowledge we have received, and for that, we are grateful.

Contents

General Introduction	1
1 General Context of the Project	2
Introduction	2
1.1 Presentation of the host organisation	2
1.1.1 Company overview and mission	2
1.1.2 Services and Security Operations Centre	3
1.1.3 Internship context	4
1.2 Problem statement	4
1.2.1 The evolving threat landscape	4
1.2.2 The SOC under pressure: alert fatigue and the analyst shortage	4
1.2.3 The automation gap	5
1.2.4 Research question	5
1.3 Study of existing solutions	6
1.3.1 Commercial SIEM and XDR platforms	6
1.3.2 SOAR platforms	6
1.3.3 Open-source SOC bundles	7
1.3.4 Emerging AI-augmented SOC platforms	7
1.3.5 Synthesis and chosen direction	7
1.4 Proposed solution and objectives	8
1.4.1 Solution overview	8
1.4.2 Objectives and scope	9
1.4.3 Expected outcomes and contributions	10
1.5 Methodology	10
1.5.1 Classic and agile methods	11
1.5.2 The project phases	11
Conclusion	12
2 Background and Comparative Study	13
Introduction	13

2.1	SOC fundamentals	13
2.1.1	Definition and objectives	13
2.1.2	Building blocks and roles	13
2.1.3	From manual to AI-augmented SOC	15
2.2	MITRE ATT&CK and threat intelligence	16
2.2.1	The MITRE ATT&CK framework	16
2.2.2	Threat intelligence and enrichment	17
2.3	Artificial intelligence and machine learning	17
2.3.1	Supervised classification for alert triage	17
2.3.2	Language models as writers	18
2.4	Overview and comparative analysis of the tools	18
2.4.1	Selection criteria	19
2.4.2	Suricata	19
2.4.3	Elastic Stack	20
2.4.4	MISP	22
2.4.5	TheHive	23
2.4.6	Cortex	23
2.4.7	n8n	25
2.4.8	Artificial intelligence: the ML classifier and the local LLM	26
2.4.9	Comparative summary	27
2.5	Retained solution	27
2.5.1	The synthesis and rationale	27
2.5.2	Human-in-the-loop governance	28
	Conclusion	29
3	Design and Implementation	30
	Introduction	30
3.1	System architecture	30
3.1.1	Network Architecture	31
3.1.2	Data flow	32
3.2	Work environment	33
3.2.1	System Environment	33
3.2.2	Service endpoints	34

3.3	Detection layer	34
3.3.1	Multi-layer detection	34
3.3.2	Custom detection rules	35
3.4	Orchestration layer	36
3.4.1	Workflow catalogue	36
3.4.2	The alert pipeline (WF1)	37
3.4.3	Supporting workflows	40
3.5	AI Integration	41
3.5.1	Supervised classifier	41
3.5.2	Local language-model	42
3.6	Case management, enrichment and response	43
3.6.1	The case workbench	43
3.6.2	The case-quality model	45
3.6.3	Human-in-the-loop response	46
	Conclusion	47
4	Evaluation	48
4.1	Test plan and environment	48
4.1.1	Attack emulation and MITRE scenarios	48
4.1.2	Evaluation metrics	49
4.2	Functional tests	49
4.2.1	Detection scenarios	49
4.2.2	Anatomy of an automated case	54
4.2.3	Response and ticketing	59
4.3	Autonomous capabilities and AI model performance	68
4.3.1	Threat intelligence to detection rule	68
4.3.2	Self-maintenance and continuous learning	70
4.3.3	ML performance	71
4.3.4	Feature importance	73
4.4	Discussion	74
4.4.1	Operational impact and alert reduction	74
4.4.2	Latency	74
4.4.3	Limitations	74

4.4.4 Future improvements	75
Conclusion	76
General Conclusion	77
Webography	78
A Platform deployment and threat-intelligence automation	i
A.1 SOC-Core host: SIEM, Fleet and orchestration	ii
A.2 Incident-management host: case management and threat intelligence	iv
B The AI decision layer	viii
C The automated response library	xiv

List of Figures

1.1	Next Step logo	3
1.2	Solution architecture	9
2.1	SOC components	14
2.2	SOC roles and tier structure	15
2.3	MITRE ATT&CK Enterprise tactics	16
2.4	Suricata in the network-detection pipeline	20
2.5	The Elastic Stack and its place in the SOC	22
2.6	The incident-management suite: TheHive with Cortex and MISP	24
3.1	SOC architecture	31
3.2	Deployment topology	32
3.3	Data flow	33
3.4	WF1 first segment	37
3.5	WF1 second segment	38
3.6	WF1 third segment	39
3.7	WF1 fourth segment	40
3.8	Enrichment flow	44
3.9	Rule generation flow	45
4.1	Port scan launched from the Kali machine.	50
4.2	SOC-007 alert raised in Kibana Security.	50
4.3	Case #153 created in TheHive from the port scan.	51
4.4	The SSH brute force launched from the Kali host with hydra	51
4.5	The SOC-002 critical alert raised in Kibana on the brute force.	52
4.6	Case #135 created in TheHive from the brute force.	53
4.7	The authorized SSH login from the trusted host.	53
4.8	The SOC-012 alert raised in Kibana on the authorized login.	54
4.9	Case #157 created in TheHive from the authorized login.	54
4.10	Case #135 details	55
4.11	Case task list	56

4.12 IoC observables	56
4.13 MISP report	56
4.14 TTPs tab	57
4.15 Investigation Notes page	58
4.16 Playbook page	58
4.17 Threat Intel pages	59
4.18 The Tier-1 alert email	60
4.19 The alert email (evidence and facts)	61
4.20 The alert email (recommended actions and links)	61
4.21 Acknowledgement confirmation	62
4.22 <code>email-acked</code> tag written to the case on acknowledgement.	62
4.23 The Cortex responder menu	63
4.24 The audit comments	63
4.25 The <code>responder:*</code> tags added to the case.	63
4.26 The Tier-2 brief email (top)	65
4.27 The Tier-2 brief email (bottom)	66
4.28 <code>iptables</code> <code>DROP</code> rule on the victim host	66
4.29 The resolved case	67
4.30 Investigation report	68
4.31 The WF3 pipeline	68
4.32 The rule generated by WF3	69
4.33 TheHive alert logging the AI-generated rule	70
4.34 The weekly-maintenance case filed by WF4	71
4.35 ROC curves for the Random Forest and XGBoost triage models on the held-out test set.	72
4.36 Confusion matrices: Random Forest (left) and XGBoost (right).	73
4.37 Random Forest feature importances	73

List of Tables

1.1	The families of solution against the steps the problem raises	8
1.2	Comparison of the classic and agile methods	11
1.3	The five phases of the project	12
2.1	The open-source tools making up the SOC and their roles	27
3.1	Virtual-machine resource allocation.	33
3.2	Service port map.	34
3.3	13 custom SOC detection rules	36
3.4	n8n workflows.	37
3.5	LLM-assisted features	43
3.6	The eight Cortex responders and the workflows they trigger.	47
4.1	Attack scenarios emulated from the Kali host.	49
4.2	Triage model performance	71

List of Abbreviations and Acronyms

Abbreviation	Full name
AI	Artificial Intelligence
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques and Common Knowledge
CICIDS2017	Canadian Institute for Cybersecurity IDS Dataset 2017
CTI	Cyber Threat Intelligence
DNS	Domain Name System
DVWA	Damn Vulnerable Web Application
ECS	Elastic Common Schema
EDR	Endpoint Detection and Response
EQL	Event Query Language
ES	Elasticsearch
ET	Emerging Threats
FP	False Positive
HTTP	HyperText Transfer Protocol
IDS	Intrusion Detection System
IoC	Indicator of Compromise
IP	Internet Protocol
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
KQL	Kibana Query Language
L1	Level 1 (SOC tier)
L2	Level 2 (SOC tier)
LLM	Large Language Model
LTS	Long-Term Support
MISP	Malware Information Sharing Platform and Threat Sharing
MITRE	The MITRE Corporation
ML	Machine Learning
MTTD	Mean Time to Detect

Abbreviation	Full name
MTTR	Mean Time to Respond
NAC	Network Access Control
NDJSON	Newline-Delimited JSON
NIST	National Institute of Standards and Technology
OSINT	Open Source Intelligence
PAM	Privileged Access Management
PAP	Permissible Actions Protocol
PFE	Projet de Fin d'Études
REST	Representational State Transfer
ROC	Receiver Operating Characteristic
SaaS	Software as a Service
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SOAR	Security Orchestration, Automation and Response
SOC	Security Operations Center
SSH	Secure Shell
STIX	Structured Threat Information Expression
TAXII	Trusted Automated eXchange of Indicator Information
TCP	Transmission Control Protocol
TIP	Threat Intelligence Platform
TLP	Traffic Light Protocol
TLS	Transport Layer Security
TTP	Tactics, Techniques and Procedures
VM	Virtual Machine
VPN	Virtual Private Network
WAF	Web Application Firewall
XDR	Extended Detection and Response
XSS	Cross-Site Scripting

General Introduction

Cyberattacks have grown in volume and sophistication faster than the teams defending against them. At the centre of an organisation's defence sits the Security Operations Center (SOC): the people, processes, and technology that detect attacks and respond to them. Modern SOCs receive daily alert volumes far beyond what analysts can review by hand, so genuine incidents slip past in the noise while analysts spend their days on repetitive work instead of investigation. Set against a worldwide shortage of security professionals, this has made automation a basic requirement rather than a luxury.

Detection and case management are by now well served by mature tools, open-source ones included, but two parts of the analyst's work stay almost entirely manual: responding to a confirmed threat, and writing each incident up as a report and a ticket. These are among the most repetitive and time-critical steps, and exactly the ones that current SOC practice, the stack Next Step itself operates included, does not yet automate or assist with AI.

This end-of-studies project, carried out within Next Step, closes that gap by building a complete, self-contained SOC in which the response is automated and the incident reports and tickets are generated by AI. Rather than altering the system the company already runs, it builds a SOC of its own, so these capabilities can be integrated and tested end to end. An automated triage engine sits at its centre: from its single decision, confident false positives close on their own, every escalated case reaches the analyst already written up as an AI-drafted report and ticket, and a confirmed threat is met with a response that runs once an analyst approves it.

The report is organised into four chapters: Chapter 1 presents the host organisation, frames the problem, reviews existing approaches, and introduces the proposed solution; Chapter 2 covers the technical background and compares the tools and techniques behind it; Chapter 3 details the design and implementation; and Chapter 4 evaluates the system and measures what its automation delivers.

Chapter 1

General Context of the Project

Introduction

This chapter sets the stage for the project. It begins by introducing the host organisation, Next Step, then frames the problem the work addresses: a SOC that detects and stores alerts well, yet still leaves its response and its reporting to be done by hand, under the constant pressure of alert overload and a shortage of analysts. It reviews the approaches available today, identifies the gap they leave, and presents the proposed solution and its objectives: an open, self-hosted SOC that automates the response and generates its incident reports and tickets with AI, with a human kept in control throughout.

1.1 Presentation of the host organisation

This project was carried out at Next Step, a Tunisian IT integrator whose cybersecurity practice is the natural setting for the work that follows.

1.1.1 Company overview and mission

Founded in 2012, Next Step is a Tunisian IT integrator specialised in infrastructure and services built around new technologies. It supports its clients across the full life cycle of a project, from design and integration through development and maintenance, with a team of certified engineers and technicians who deliver tailored solutions [1]. Holding both ISO 9001 and ISO 27001 certifications, and having delivered more than 1,800 projects for over 700 national and international clients, the company has established itself among the recognised IT integrators on the Tunisian market [2].



Figure 1.1: Next Step logo

1.1.2 Services and Security Operations Centre

Next Step's portfolio spans several IT domains [3]:

- **Enterprise networks:** LANs, WiFi, WAN optimisation, and data-centre networking.
- **Cybersecurity:** perimeter and internet security, WAF, IPS, VPN, antivirus, strong authentication, NAC, SIEM, vulnerability management, and PAM.
- **Mobility:** platforms that bring employees and company services into a single shared environment.
- **IP telephony and videoconferencing:** turnkey enterprise telephony and meeting-room setups.
- **Management and supervision:** monitoring, alerting, dashboards, KPIs, service desk, and IT asset management.
- **Cloud services:** hosting, Infrastructure as a Service, and Backup as a Service on managed infrastructure.

Of these, the cybersecurity line is the one this project extends. Next Step already designs and operates security infrastructure for its clients (SIEM, IPS, and the surrounding detection and monitoring stack), which places the autonomous SOC proposed here directly in line with the services the company already delivers.

Within its cybersecurity practice, Next Step operates a Security Operations Centre (SOC) dedicated to the security and integrity of its clients' information systems. Built around a SIEM and the surrounding monitoring stack, the SOC detects, analyses and responds to security incidents in real time, and is organised into a tiered model:

- **Level 1 — Monitoring:** the initial detection and monitoring of security events and alerts.
- **Level 2 — Incident response:** the in-depth analysis of, and response to, confirmed incidents.

- **Level 3 — Platform management:** the management and securing of cloud platforms and critical applications.
- **SOC manager:** the oversight of operations, ensuring effective incident handling and continuous improvement.

Next Step also delivers this capability as a managed service, SOC as a Service (SOCaaS), giving clients on-demand access to its security experts together with round-the-clock threat monitoring and incident management, increasingly supported by artificial intelligence for proactive detection and the early identification of zero-day threats.

1.1.3 Internship context

This work was conducted during the end-of-studies internship at Next Step. The company's security-integration activity gave the project a realistic frame: it designs and operates SOC and security infrastructure for its clients, so the components a working SOC relies on were available to build with and the pressures of running one were familiar from the start. What that infrastructure does not yet include is AI-driven automation of the analyst's work, neither automated response nor AI-generated reporting, and that absence is what the project set out to address. The goal fixed at the outset was to build a complete, self-contained SOC in which the response is automated and the incident reports and tickets are written by AI, and to test what such automation brings to the way a SOC operates, capabilities Next Step does not currently have in its own stack.

1.2 Problem statement

1.2.1 The evolving threat landscape

Cyber risk has grown heavier and more expensive every year. IBM put the global average cost of a data breach at \$4.88 million in 2024, a 10% rise on the previous year and the steepest annual jump since the pandemic, with the average breach taking 258 days to identify and contain [4]. Behind that figure is a steady rise in both the volume of attacks and the automated tooling attackers use, which together push more events toward the teams responsible for catching them.

1.2.2 The SOC under pressure: alert fatigue and the analyst shortage

Security Operations Centers absorb the operational side of this pressure, but the human side cannot scale at the same pace. ISC2 estimated the global cybersecurity workforce gap at 4.8 million

professionals in 2024, while the workforce itself grew by only a fraction of a percent; for the first time, a lack of budget rather than a lack of qualified talent became the leading reason teams stay understaffed [5]. Inside the SOC the result is alert fatigue: analysts face daily alert volumes far beyond what they can review, real incidents slip past in the noise of false positives, and most of the working day goes to repetitive triage instead of investigation.

Automation is the obvious lever, and its payoff is measurable: the same IBM study found that organisations using AI and automation extensively across their security operations detected and contained breaches 98 days faster on average and spent \$2.2 million less per breach. For a SOC, automation has shifted from an optimisation to a baseline requirement.

1.2.3 The automation gap

Automation has reached some parts of the SOC more than others. Detection produces the alerts, case-management tools store and organise them, and a growing range of platforms now help triage the flood. Two steps further down the line, however, have stayed stubbornly manual. The first is the response: once a threat is confirmed, an analyst still moves by hand to block an address, isolate a host, or quarantine a file, one console at a time, at the very moment speed matters most. The second is the write-up: every case that matters has to be documented as a report and a ticket, a slow and repetitive task that nonetheless produces the record an organisation later relies on. These two jobs are where an analyst's time drains away after the alert has already been understood.

They are also the two that current SOC practice, the stack Next Step itself operates included, does not yet automate or assist with AI. A SOC can be fully equipped to detect threats and to store cases and still leave both the response and the reporting entirely to people. That is the gap this project targets: not detection, which is already well covered, but the automation of the response and the AI-assisted generation of the reports and tickets that follow it.

1.2.4 Research question

This gap defines the question the project sets out to answer: how can a SOC automate its response and generate its incident reports and tickets through AI, and what does that automation bring to the way the SOC operates, when it is built and tested from end to end on an open, self-hosted stack? Two constraints frame the answer. The automation must keep a human in control of every consequential action, so that speed is gained without ceding the decision; and the whole system must run on the organisation's own hardware at no recurring cost, both to keep data inside

the perimeter and to make the result reproducible well beyond a single budget.

1.3 Study of existing solutions

Several families of tools already address parts of the problem. This section weighs them at the level of approach rather than individual products, since the specific tools retained for the solution are compared in Chapter 2. Four families stand out, and each is judged on the same questions the problem raises: what it costs, where it runs, whether it makes the triage decision, and how open it is about its reasoning.

1.3.1 Commercial SIEM and XDR platforms

SIEM platforms such as Splunk, IBM QRadar, and Microsoft Sentinel centralise and correlate security events from across the infrastructure, and the XDR products that have grown out of them extend that view across endpoints, network, identity, and cloud. They are mature and give broad visibility, which is why they anchor most large enterprise SOC's. Two things keep them out of reach of smaller teams. Their licensing scales with the volume of data ingested and can reach six figures a year, so the cost grows with the very telemetry a SOC most wants to collect. And for all their breadth, their detection is tuned to raise alerts, not to decide which ones matter, so they hand the analyst more to look at rather than less. The category fits large budgets, and the large teams needed to operate it, far better than small ones.

1.3.2 SOAR platforms

SOAR tools such as Cortex XSOAR, Tines, and the open-source Shuffle automate the response side of the work, chaining playbooks across the SIEM, the ticketing system, and the enrichment tools so that routine handling runs without a human at every step. Where they help, they help a great deal, cutting the repetitive effort that fills an analyst's day. But a playbook only ever does what it was told: each one has to be written, tested, and maintained by hand, and a playbook that fires on the wrong alert simply automates a mistake faster. The deeper point is that SOAR automates the response to a decision; it does not make the decision. The judgement of whether an alert matters still rests with a human or a static rule. The category is also increasingly folded into broader SIEM and XDR suites rather than bought on its own.

1.3.3 Open-source SOC bundles

The standard open-source bundle pairs a free SIEM with a case manager, an enrichment engine, a threat-intelligence platform, and a workflow tool. It carries no licence cost, runs on the organisation's own hardware, and is fully transparent down to the source, which is why it is the usual starting point for small teams and academic labs. The price it asks is paid in effort rather than money: the pieces must be assembled, integrated, and kept running by whoever deploys them. Its deeper limit is the same one that runs through the first two families: triage stays entirely human. The bundle can collect, correlate, enrich, and orchestrate, but nothing in it decides whether an alert is a real threat or just noise, so every alert still lands on an analyst's desk.

1.3.4 Emerging AI-augmented SOC platforms

A newer wave of commercial platforms wraps machine learning and language models around triage itself, investigating each alert and handing the analyst a verdict with the evidence behind it. The capability is real and measured: a 2025 benchmark by the Cloud Security Alliance found that analysts working alongside these tools investigated escalated alerts 45 to 61% faster, and more accurately, than a manual control group [6]. The catch is in how that capability is delivered. These platforms are sold as cloud-hosted subscriptions priced for mid-market and enterprise budgets, so the data they reason over has to leave the organisation, and the cost recurs every year. They are also closed about how their verdicts are produced, which leaves a SOC trusting a decision it cannot inspect. The intelligence is there; the openness, the on-premises option, and a price a small team could absorb are not.

1.3.5 Synthesis and chosen direction

Set side by side, the families point in the same direction without any one of them arriving. The commercial SIEM and XDR platforms see everything but decide nothing: they raise the alerts and leave the triage, the response, and the write-up to people. SOAR adds automated response, but only as scripted playbooks with no judgement of which alert deserves one, and it does not touch the reporting. The open-source bundle is free, on-premises, and open, yet it automates none of the three: triage, response, and reporting all stay manual. The newest AI-augmented platforms come closest, automating triage, drafting written summaries, and assisting with response, but they do so as closed, cloud-hosted subscriptions whose reasoning a SOC cannot inspect and whose data must leave the perimeter. Table 1.1 sets the families against the steps the problem raises.

Table 1.1: The families of solution against the steps the problem raises

Approach	Cost	Deployment	Triage	Response	Reporting
Commercial SIEM / XDR	High	Cloud or on-prem	Manual	Manual	Manual
SOAR	Add-on	Cloud or on-prem	Manual	Automated	Manual
Open-source bundle	None	On-premises	Manual	Manual	Manual
AI-augmented SOC	High	SaaS only	Automated	Assisted	AI-written
Retained design (this project)	None	On-premises	Automated	Automated	AI-written

The empty corner of the table is the opening. No existing family automates the response and the reporting on an open, self-hosted, no-cost stack while keeping a human in control. That corner is where this project places itself. It keeps the open-source bundle as its foundation and adds the intelligence the bundle lacks: an automated triage engine that decides what deserves attention, and, driven by that decision, the two capabilities current practice still leaves to people, a response that runs automatically once an analyst confirms it, and incident reports and tickets drafted by a local language model. The triage engine is the mechanism; the automated response and the AI-written reporting are the result. The tools and techniques behind it are the subject of the next chapter.

1.4 Proposed solution and objectives

1.4.1 Solution overview

The proposed solution is a complete, self-contained Security Operations Center, built from open-source tools and run at zero recurring cost on the organisation's own hardware. Its purpose is to automate the two steps that current practice leaves to people, the response and the reporting, driven by an automated triage engine. Rather than a single product, it is a stack of cooperating layers, each with one job, organised around a single principle: the machine triages, the human decides.

A detection layer watches the protected environment from several complementary angles and turns suspicious activity into alerts. An orchestration layer acts as the conductor, taking every alert and driving it through the rest of the pipeline in a fixed, repeatable order. At the heart of the system sits the triage engine, which decides, for each alert, whether it is a real threat or noise, closing confident false positives on its own and escalating the rest. From that decision the

automated capabilities follow: an enrichment and case-management layer gathers the context an analyst would otherwise collect by hand and opens a case, a reasoning layer drafts a plain-language report of what happened, and a response layer carries out containment, but only once a human has confirmed it.

Two ideas hold the design together. First, the reasoning layer only ever writes; it never chooses a verdict, a severity, or an action, so the automation stays trustworthy even when the model is wrong or offline. Second, every consequence passes through a human: closing an obvious false positive is the only fully automatic outcome, and nothing is contained without a confirmation. Closed cases also feed back into the triage engine, which learns from them and grows sharper over time. Figure 1.2 shows how the layers connect.

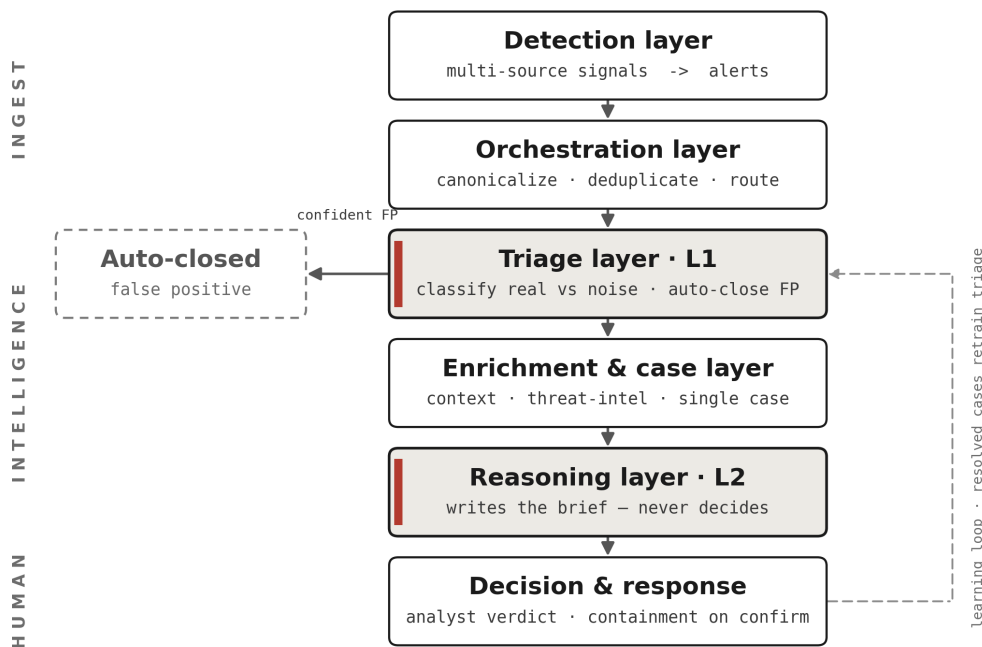


Figure 1.2: Solution architecture

1.4.2 Objectives and scope

The aim is to automate the work that current practice still does by hand, so analysts spend their time on incidents that matter. Concretely, the system should: triage every incoming alert automatically and close confident low-risk false positives without human effort; deduplicate the alerts of a single attack into one case; enrich each case with threat-intelligence and MITRE context;

on escalation, create the ticket and generate a written, decision-ready report with AI; and carry out the response automatically once an analyst has confirmed it. A further objective is to test the result: to measure, against realistic attacks, what this automation actually delivers to the SOC's work.

Two boundaries are deliberate. First, the language model is a writer, not a decision-maker: it never chooses a verdict, a severity, or an action, so a wrong or offline model can never produce a wrong decision. Second, every consequential action stays with a human, so false-positive auto-close is the only fully automatic outcome, and containment runs only after an explicit confirmation. Proactive Level 3 threat hunting and large-scale multi-tenant operation fall outside the scope of this work.

1.4.3 Expected outcomes and contributions

If it works as designed, the SOC handles by itself the two steps that used to consume an analyst's time after an alert was understood: the obvious false positives are gone, every case that matters arrives already written up as a report and a ticket, and a confirmed threat is contained at the click of a button rather than across several consoles. The analyst is left with fewer, richer cases and the decisions that genuinely need a person.

Beyond assembling existing open-source tools, the project's contribution is fourfold: an automated, human-confirmed response that turns containment into a single approved action; AI-generated incident reports and tickets that spare the analyst the write-up; a supervised triage engine that drives both, learning which alerts are real from the cases analysts close; and a governance layer that keeps a human in control of every consequence. The whole is built as a complete, self-contained SOC and tested end to end, so the gains can be measured rather than assumed.

Because the system is open-source and runs at no recurring cost on the organisation's own hardware, the same design is also within reach of the small and mid-sized organisations, Next Step's clients among them, that the commercial market prices out, though that wider reach is a secondary benefit rather than the aim.

1.5 Methodology

The way the work was organised followed from the nature of the project. This section sets out the two broad families of project-management method, classic and agile, explains which one this project followed and why, and presents the phases the work was divided into.

1.5.1 Classic and agile methods

Two families of method dominate software and systems projects.

The *classic method* maps the work as a single, linear sequence of phases, each one completed and validated before the next begins. It is sometimes drawn as a V, with every development phase matched to a later validation phase. It suits projects whose goal and requirements are known from the start, and it rests on firm planning, thorough documentation, and limited change once the specification is fixed.

The *agile method* works the opposite way. Instead of one long sequence, the work is broken into short iterations, each producing something the client can see and react to, so the direction can be adjusted as the project goes. It favours flexibility and continuous feedback over fixed planning, and welcomes change at any stage rather than freezing the specification early.

Table 1.2 sets the two side by side.

Table 1.2: Comparison of the classic and agile methods

	Classic method	Agile method
Life cycle	A single sequence of phases	Short, repeated iterations
Planning	Fixed at the outset from known requirements	Adjusted as the project proceeds
Validation	Mostly at the end of the cycle	Continuous, with the client seeing each step
Change	Hard to accommodate once the specification is set	Welcomed at any stage

For this project, the classic, phased approach was the natural fit. The goal was clear from the outset, an autonomous, human-in-the-loop SOC, and the work fell into phases that each depended on the one before it: the environment had to exist before detection could be built, detection had to produce alerts before triage could be automated, and the whole pipeline had to be in place before it could be evaluated. A sequence of phases, each completed before the next, matched those dependencies better than short iterations would have.

1.5.2 The project phases

The work was divided into five phases over the four months of the internship, each ending in a working result that the next phase built on. They were not sealed off from one another at the

edges: the study continued into the build as tools revealed their constraints, and fixes found during validation fed back into the earlier phases. Table 1.3 lists them.

Table 1.3: The five phases of the project

Phase	Name	Main activities
1	Study and design	Survey of SOC operations, open-source tools, and AI techniques; definition of the target architecture.
2	Environment build	Provisioning the virtual machines and the overlay network; installing and configuring the base stack.
3	Detection and intelligence	Network and host detection; the custom detection rules and their MITRE tagging; the threat-intelligence feeds.
4	Automation and AI	The orchestration workflows; training and serving the triage classifier; the language-model writer.
5	Integration and validation	Case management, enrichment, and human-in-the-loop response; testing against emulated attacks and measuring the results.

Conclusion

This chapter set the stage for the project. It introduced Next Step and its cybersecurity practice, then framed the problem the work addresses: a SOC can detect threats and store cases with mature tools, yet two steps that follow an alert, the response and the write-up, are still done by hand, and current practice, the stack Next Step itself operates included, does not automate them or assist them with AI. From that gap came the proposed solution, a complete, self-contained open-source SOC that automates the response and generates its incident reports and tickets with AI, driven by a triage engine that decides what deserves attention, with a human kept in control of every consequence. The chapter closed by setting out the phased method used to build it. The next chapter examines the tools and techniques behind these layers and explains why each was chosen.

Chapter 2

Background and Comparative Study

Introduction

This chapter lays out the background the solution rests on and the reasoning behind its design. It starts with how a modern SOC works and how it has shifted toward AI augmentation, then covers the MITRE ATT&CK framework and the threat intelligence behind the tagging and enrichment steps, and the two AI techniques the system relies on: supervised classification for triage and a language model for writing. It closes by comparing the open-source tools for each role and justifying the retained stack.

2.1 SOC fundamentals

2.1.1 Definition and objectives

A Security Operations Center is the function responsible for continuously monitoring, detecting, analysing, and responding to security events across an organisation's systems. It is the combination of people, technology, and process that turns raw telemetry from networks, endpoints, and applications into security decisions. Its performance is usually measured by two figures: Mean Time To Detect (MTTD), how quickly an intrusion is spotted, and Mean Time To Respond (MTTR), how quickly it is contained. Lowering both is the core mission, since every hour an attacker stays undetected raises the eventual cost of the breach.

2.1.2 Building blocks and roles

A modern SOC runs on a recognised set of tools: a SIEM to centralise and correlate logs, EDR to watch endpoints, IDS/IPS to inspect network traffic, a Threat Intelligence Platform (TIP) to manage indicators, and a SOAR layer to automate routine response.

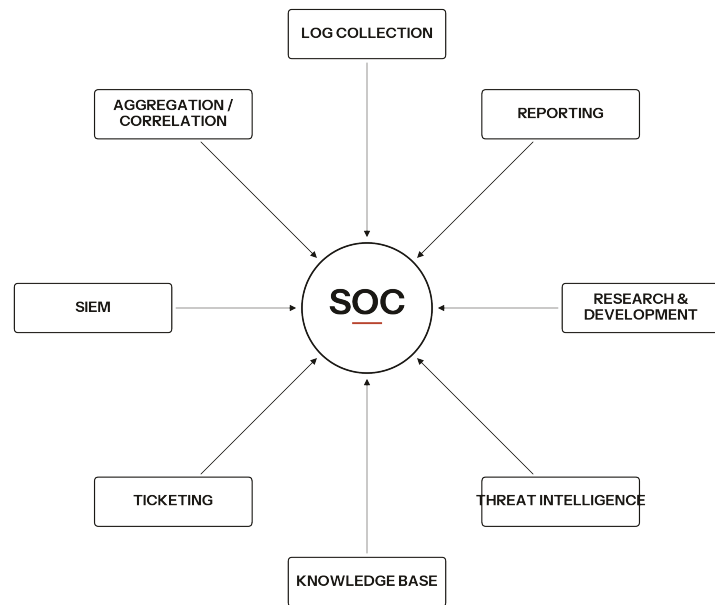


Figure 2.1: SOC components

Figure 2.1 arranges these functions around the SOC as the inputs and capabilities that feed it, from log collection and correlation through ticketing, the knowledge base, reporting, and threat intelligence. Together they form the toolchain on which detection and response depend.

Around that toolchain the team is organised in tiers of increasing expertise: Level 1 analysts handle front-line monitoring and initial triage, Level 2 take on deeper investigation and containment, and Level 3 cover threat hunting and the hardest cases, supported by detection engineers, incident responders, and threat-intelligence analysts.

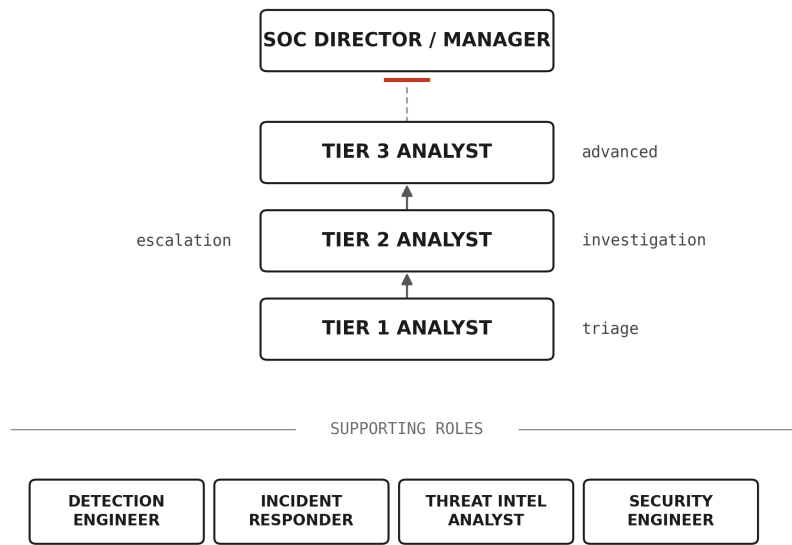


Figure 2.2: SOC roles and tier structure

Figure 2.2 shows this structure: alerts climb from Level 1 triage to Level 2 investigation and Level 3 hunting, with the supporting roles working alongside the tiers. The repetitive load this project automates sits at the bottom two tiers, which is where alert volume concentrates and where analysts are hardest to retain.

2.1.3 From manual to AI-augmented SOC

The SOC has not stood still. Early security operations were handled ad hoc by general IT staff with little central visibility. The arrival of commercial SIEMs brought centralised log collection and real-time correlation, followed by the tiered analyst model and, later, by threat-intelligence feeds, SOAR automation, and the extension of EDR into XDR across endpoints, network, identity, and cloud. The most recent shift, from 2024 onward, embeds machine learning and large language models directly into the alert pipeline: AI assistants now triage alerts, suggest investigation paths, and draft incident summaries. The Cloud Security Alliance’s 2025 benchmark measured investigations 45 to 61% faster with AI assistance [6], though the prevailing view, and the one this project follows, keeps a human in the loop rather than chasing a fully autonomous SOC. This work sits at that latest generation, built entirely on open-source components.

2.2 MITRE ATT&CK and threat intelligence

The solution leans on two bodies of knowledge to turn raw alerts into something an analyst can act on: a framework that describes how attackers behave, and the threat intelligence that says whether a given indicator is already known to be malicious.

2.2.1 The MITRE ATT&CK framework

MITRE ATT&CK is a public, continuously updated knowledge base of adversary behaviour, built from real-world observations rather than theory [7]. It organises that behaviour into *tactics*, the goal an attacker pursues at a given stage (initial access, execution, persistence, privilege escalation, and so on), and *techniques*, the concrete methods used to reach each goal, each carrying an identifier such as T1059. The Enterprise matrix, covering Windows, Linux, macOS, and cloud, is the one relevant here.

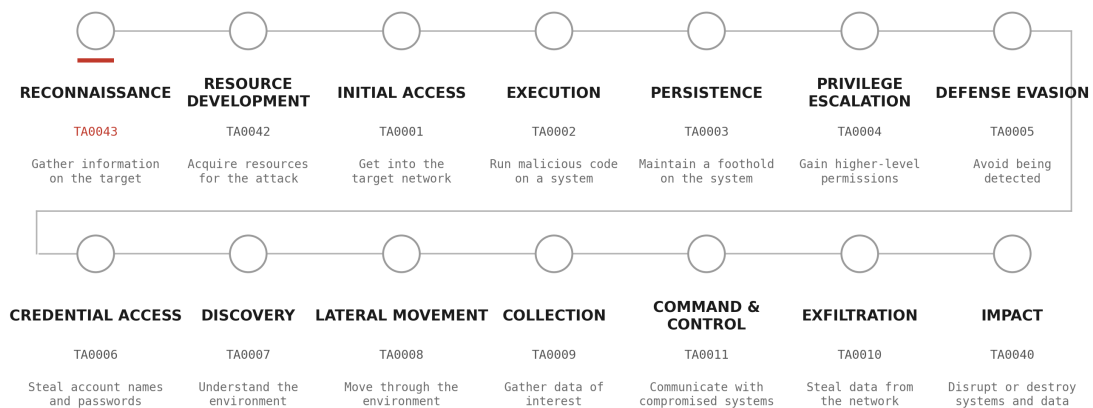


Figure 2.3: MITRE ATT&CK Enterprise tactics

Figure 2.3 lists the fourteen Enterprise tactics in their canonical order. The framework is descriptive rather than prescriptive: unlike the older linear kill chain, it does not assume an attack follows a fixed sequence. For a SOC this matters in two ways. Each detection rule is tagged with the technique it catches, so an alert stops being a bare event and becomes evidence of a specific adversary behaviour; and when several tagged alerts arrive from the same host in a short window, the sequence of tactics shows how far an intrusion has progressed. In this project those technique tags travel with every alert and case, and the mapping from a tag to its tactic and name is a fixed

lookup, never something the AI decides.

2.2.2 Threat intelligence and enrichment

A raw alert rarely answers the questions an analyst actually has: is this IP known to be malicious, has this file hash appeared in a campaign, does the affected host even matter. Cyber Threat Intelligence (CTI) supplies those answers. NIST defines it as information that helps an organisation identify and respond to threats, ranging from high-level strategic trends down to machine-readable indicators such as IP addresses, domains, and file hashes [8]. Most of it comes from open feeds, community platforms, and CERT advisories, exchanged in standard formats like STIX and TAXII.

Enrichment is the act of attaching that context to an alert before it reaches a human: reputation lookups on its observables, the criticality of the affected asset, and any past incident tied to the same indicator. An enriched alert stops being a single data point and becomes a small dossier, which is what makes both the automated triage and the analyst's brief meaningful, since each reasons over context instead of a bare alert. In this project that context is supplied by a threat-intelligence platform holding the indicator database and an enrichment engine running the per-observable lookups.

2.3 Artificial intelligence and machine learning

The solution uses machine learning for two jobs that have to stay separate. The first is a decision: is this alert a real threat or noise? A supervised model makes it. The second is writing: turning an escalated case into a brief an analyst can read in seconds, which a language model does. The rule that holds the design together is plain: the model that decides never writes, and the model that writes never decides.

2.3.1 Supervised classification for alert triage

Deciding whether an alert deserves attention is a classification problem: from a set of measurable properties, the model sorts each alert into threat or false positive. The features are the things an analyst would glance at: the alert's risk score and severity, which rule fired and how often that rule has lately proved a false positive, the MITRE technique, whether the source is an internal address, the time of day, and how many alerts clustered together in a short window. It learns this from examples that already carry the answer, and a SOC produces those for free: every case an analyst closes is a labelled example, its properties recorded when the alert arrived and its verdict

supplied by the human. As new cases close, the model is retrained on them and sharpens over time, which is the feedback loop the architecture relies on.

Alert features are tabular and mixed (categories, counts, and scores), which suits tree-based models rather than the neural networks built for images and text. The project works with the two standard choices, random forests [9] and gradient-boosted trees [10], both of which train on modest amounts of data and can report which features drove a decision, so a closed alert can be explained rather than merely asserted. The two are compared in Chapter 4.

What matters more than raw accuracy is that the two mistakes are not equal. Auto-closing a real threat can leave an intrusion running unseen; flagging a harmless alert costs only a moment of attention. So the classifier reports a confidence alongside its class, and only false positives it is highly confident about are closed automatically; everything else goes to a human. That safety floor is what makes automatic closure acceptable at all.

2.3.2 Language models as writers

A large language model is trained on large amounts of text to produce fluent prose, and that is the only skill the system asks of it. An escalated case already holds everything an analyst needs (alert fields, enrichment results, the order of techniques observed), but scattered; the model gathers it into a short, readable account of what happened and why it was escalated.

It is never allowed to decide. Language models hallucinate, stating unsupported things with full confidence [11], and a confidently wrong verdict that closes a real incident is a security failure, not a stray typo. Every field that carries a consequence (verdict, severity, urgency, recommended action) is set by the pipeline and the classifier; the model only writes them up. Because it is handed a case that has already been decided and is grounded on that case's own data, there is no decision left for it to get wrong: a poor model produces an awkward brief, an offline one produces none, and in both cases the verdict and the fields still stand.

Because nothing may leave the organisation's hardware, the writer runs locally, on an open-weight model small enough for commodity machines, which is enough to give the second analyst tier a ready-written case without sending a single record outside the perimeter.

2.4 Overview and comparative analysis of the tools

The previous section settled the techniques; this one settles the tools that carry them out, and the reasoning behind each choice. Every component is open source and self-hosted, in keeping

with the project’s aims, and carries no recurring cost. Rather than list the tools flatly, the section takes the stack one piece at a time: for each tool it sets out what the tool is and how it is built, what it is normally used for, and the specific job it does in this SOC. It opens with the criteria the tools were weighed against and closes with a summary table.

2.4.1 Selection criteria

Each tool was weighed against a few requirements, all following from the research question in Chapter 1:

- **Open source, no ongoing cost**, so the finished system carries no licence fee.
- **Self-hosted**, so it runs in the project’s own environment and keeps data under its own control.
- **Mature and community-backed**, so it is stable and well documented enough to rely on.
- **Drivable through an interface**, so the orchestrator can chain it with the others without manual steps.

These requirements already rule out the commercial options surveyed in Chapter 1, which fail the first two. The tools that meet them are presented below.

2.4.2 Suricata

Suricata is the network sensor of the stack, the engine that watches traffic on the wire for signs of attack [12].

- **Features and architecture.** An open-source intrusion-detection, intrusion-prevention, and network-security-monitoring engine, developed since its first release in 2010 by the non-profit Open Information Security Foundation. Its defining trait is a multi-threaded design that spreads packet capture, decoding, detection, and logging across worker threads pinned to CPU cores, which lets a single instance inspect multi-gigabit traffic on commodity hardware where the older, single-threaded Snort would need several machines to keep up. Packets run through a pipeline of decoding, flow tracking, stream reassembly, application-layer parsing, and signature matching. Two properties matter most for a SOC: it identifies protocols such as HTTP, TLS, DNS, SMB, and SSH by their content rather than by port number, so a service moved to an unusual port is still inspected; and its rule language is compatible with Snort’s, which opens the door to the largest open signature ecosystem in the field. The same engine

runs passively, reading a copy of the traffic and only raising alerts, or inline, where it can drop the packets that match a blocking rule.

- **Use cases.** Passive network intrusion detection on perimeter or internal traffic; inline prevention at a network choke point; and network security monitoring, where it records protocol metadata and extracts transferred files for threat hunting and forensics.
- **Utility in this SOC.** It provides the network layer of detection that an endpoint-only stack would miss: much of what a SOC cares about, such as lateral movement, command-and-control traffic, and data being carried out, shows up first on the wire. It runs on the target host with the open Emerging Threats ruleset (around 49,900 signatures, refreshed daily), watches the overlay interface, and ships its events into Elasticsearch through the Elastic Agent, where a catch-all rule lifts them into the alert stream beside the host-based detections.

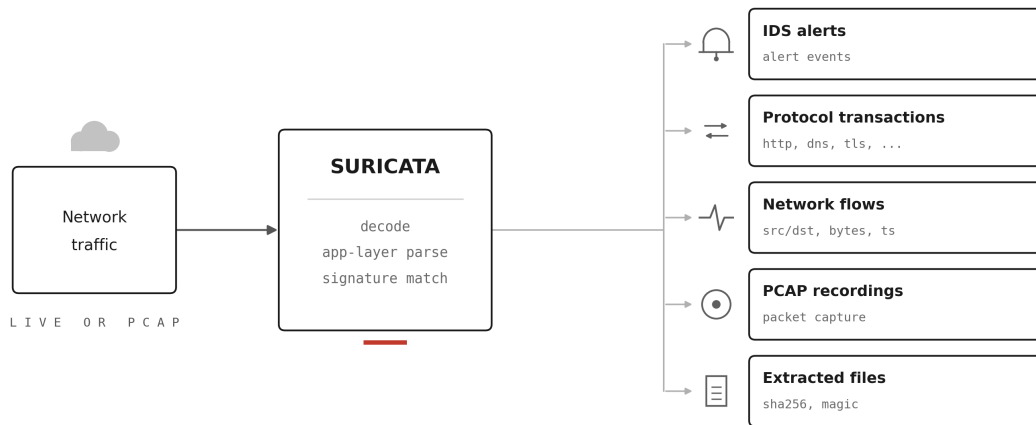


Figure 2.4: Suricata in the network-detection pipeline

Figure 2.4 traces an event from the monitored interface into the alert stream.

2.4.3 Elastic Stack

The Elastic Stack is the backbone of the SOC, the layer that collects, stores, searches, and turns logs into alerts [13].

- **Features and architecture.** What began as the three-tool ELK stack (Elasticsearch, Logstash, Kibana) has grown into a wider suite, built around Elasticsearch, a distributed

search and analytics engine based on Apache Lucene that stores schema-free JSON documents in sharded, replicated indices and exposes everything over a REST API. It was first released in 2010 by Shay Banon, with Elastic N.V. formed around it in 2012. The pieces divide the work cleanly:

- *Elasticsearch* indexes and stores every event and answers queries across very large datasets.
- *Logstash* collects and transforms incoming logs before they are stored.
- *Kibana* is the interface: dashboards, ad-hoc search, alerting, and the Elastic Security app.
- *Beats and the unified Elastic Agent*, managed centrally by *Fleet Server*, ship logs, metrics, and security telemetry from the monitored hosts.

Layered on the same data store, Elastic Security adds SIEM and EDR features: more than 1,300 prebuilt detection rules mapped to MITRE ATT&CK, custom rules written in EQL or KQL, and machine-learning anomaly jobs. Everything collected is normalised to the Elastic Common Schema (ECS), a shared set of field names that lets one detection rule apply uniformly across logs from different sources.

- **Use cases.** Log management and observability, full-text search over large datasets, and, through Elastic Security, a full SIEM that correlates events and runs detection rules over the stored data.
- **Utility in this SOC.** It fills several roles at once, which is its main attraction: the same cluster that stores the data also runs the detection engine and hosts the analyst’s dashboards, removing the integration cost that usually slows a SOC build. Here it collects host and Suricata logs through the Elastic Agent, stores and indexes every event, runs the detection engine (around 1,644 prebuilt rules plus the project’s 13 custom rules, each tagged with its MITRE technique), and forwards the alerts it raises, severity-gated, to the orchestrator, while Kibana stays the analyst’s window onto the data.

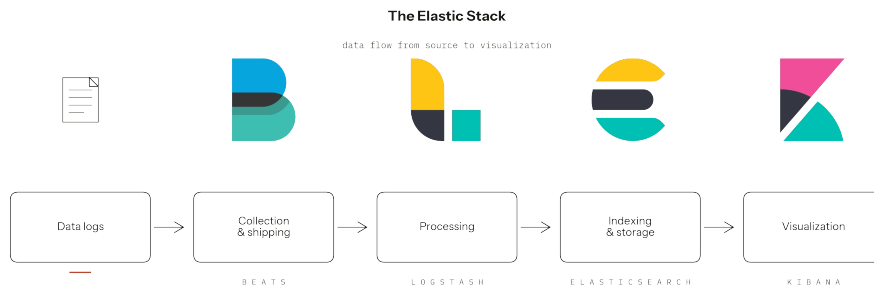


Figure 2.5: The Elastic Stack and its place in the SOC

Figure 2.5 shows how its components fit together around the SOC.

2.4.4 MISP

MISP is the stack’s threat-intelligence platform, the store of known indicators that alerts are checked against [14].

- **Features and architecture.** An open-source platform for storing and exchanging structured threat intelligence, maintained by CIRCL (the Computer Incident Response Center Luxembourg) and backed by a large community of sharing groups. Its data model rests on a few primitives: *events* that group the indicators of a campaign or incident, *attributes* that are the individual indicators, and *objects* that bundle related attributes into compound entities. These are enriched by:
 - *Galaxies*, curated catalogues of threat actors, malware families, and techniques that map to frameworks such as MITRE ATT&CK;
 - *Taxonomies*, shared tagging vocabularies such as the Traffic Light Protocol, so that one organisation’s tag means the same to another.

Its defining property is federation: an instance can synchronise with others under controlled distribution rules, and public feeds from CIRCL, abuse.ch, and AlienVault OTX are ingested out of the box. Data moves in and out in standard formats, including STIX, and can be exported as ready-made Suricata or Snort rules.

- **Use cases.** The central threat-intelligence database for a team; a federated sharing hub for CERTs, sector groups, and trust communities; and a feedback path into detection engineering, where shared indicators become new detection rules.

- **Utility in this SOC.** It holds the local indicator database and refreshes it from public feeds every few hours. The pipeline checks the source address of each case against it and tags the case with what it finds, and a separate workflow turns a newly published MISP event into a candidate detection rule for an analyst to review.

2.4.5 TheHive

TheHive is the analyst's workbench, where every escalated alert becomes a case to be worked and closed [15].

- **Features and architecture.** An open-source Security Incident Response Platform built for collaborative case management. It began in 2014 inside a financial institution's incident-response team and is now developed by StrangeBee, the company its creators founded in 2018, alongside a free Community Edition. A case gathers the alert payload, the observables drawn from it (addresses, domains, hashes, files), the analysts' tasks, and the timeline of their actions in one workspace that several people can work at once. A few features stand out: *case templates* that pre-load the standard task list for a recurring incident type, so a new analyst follows the same path as the most experienced one; ATT&CK tagging that carries through search and reports; and a full REST API that exposes every action in the interface, which is the point at which automation hooks in. Under the surface it runs as a Scala service over a database backend with an Elasticsearch index for search.
- **Use cases.** The case workbench for SOCs and CSIRTs through the life of an incident; multi-tenant case management for managed-security providers serving several customers from one deployment; and a searchable archive of closed cases that gives context to new ones.
- **Utility in this SOC.** It is the single pane of glass the analyst works from. The pipeline opens a case for each alert that is not auto-closed, fills its custom fields (urgency, blast radius, kill-chain stage, and containment window), and the decision workflows complete the case's tasks and close it with a verdict.

2.4.6 Cortex

Cortex is TheHive's companion, the engine that enriches a case's evidence and carries out response actions [16].

- **Features and architecture.** An open-source analysis and response engine from the same team as TheHive, called from TheHive, MISP, or a script through a REST API. It exposes

two kinds of module: *analysers*, which run a read-only query about an observable and return a structured report, and *responders*, which take an action. Its catalogue runs to more than three hundred ready-made modules, covering:

- reputation services such as VirusTotal, AbuseIPDB, and GreyNoise;
- malware sandboxes and OSINT lookups such as WHOIS, Shodan, and GeoIP;
- threat-intelligence connectors for MISP and OpenCTI;
- active-response actions such as blocking an address at a firewall or isolating a host.

Each module runs in isolation and declares the data-handling levels it operates under, so a slow or misbehaving analyser cannot drag down the others, and an analyst is only offered the actions a given observable allows.

- **Use cases.** Analysing observables at scale from one place instead of querying many tools by hand, and automating active response, which is why it is most often run as TheHive’s enrichment back end.
- **Utility in this SOC.** When a case is created, the pipeline asks Cortex to run the analysers that fit each observable and attaches their reports to the case, so the analyst sees the reputation and context of every indicator without looking it up by hand.

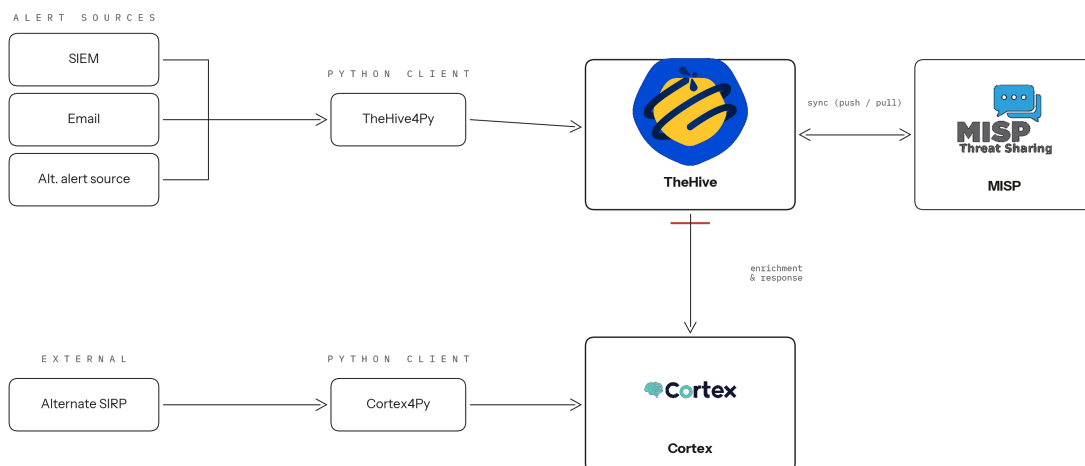


Figure 2.6: The incident-management suite: TheHive with Cortex and MISP

Figure 2.6 shows the suite’s general architecture: alert sources feed TheHive through its

client, TheHive exchanges indicators with MISP, and it passes observables to Cortex for enrichment and response.

2.4.7 n8n

n8n is the conductor of the system, the tool in which the whole automated pipeline is built [17].

- **Features and architecture.** An open-source workflow-automation platform, started in 2019 by Jan Oberhauser, whose name is a numeronym for "nodemation", a contraction of node-based automation. It sits between no-code tools such as Zapier and writing automation by hand. A workflow is a graph of connected nodes that the engine walks at run time, passing JSON from one node to the next, and nodes fall into three kinds:
 - *triggers* that start a workflow on an event, such as a webhook, a schedule, or a new alert in another tool;
 - *actions* that each wrap a single API operation, drawn from a catalogue of more than 400 ready-made integrations;
 - *logic and code nodes* for branching, transformation, and free-form JavaScript or Python.

A generic HTTP Request node reaches any service that lacks a dedicated node, a credential store keeps secrets out of the workflows themselves, and a native set of AI nodes (including one for Ollama) lets a workflow call a language model in line. It runs on Node.js, stores each workflow as JSON, and shows a step-by-step view of every run, which makes a workflow far easier to inspect than most SOAR platforms.

- **Use cases.** Security orchestration and response, lightweight data pipelines, internal tooling, and AI orchestration, anywhere two or more systems need wiring together without building a full application.
- **Utility in this SOC.** It is general-purpose rather than tied to one vendor, with native nodes for the rest of the stack (TheHive, Cortex, MISP, and the Elastic Stack) and the HTTP node for everything else, and its self-hosted, fair-code distribution runs inside the network with no egress and no per-execution metering. The whole of the SOC's automation, the ten workflows that carry every alert from arrival through triage, enrichment, notification, the

analyst's decision, and the confirmed response, lives here, so the pipeline can be read and changed as workflows rather than rewritten as code.

2.4.8 Artificial intelligence: the ML classifier and the local LLM

The two pieces the open-source bundle does not provide, a model that triages and a model that writes, rest on two further tools.

The supervised classifier.

- **Features and architecture.** A supervised model built with established machine-learning libraries: a random forest as the primary model, with gradient-boosted trees kept alongside as a benchmark, wrapped as a small web service that the orchestrator calls over HTTP and that can be retrained on new data without touching the rest of the stack.
- **Use cases.** Models of this kind are the standard way to sort tabular records into categories from labelled examples, which is exactly the shape of the triage problem: each alert is a row of features, and the label is the verdict a human eventually gives.
- **Utility in this SOC.** It triages every alert, returning a verdict and a confidence, which is what lets the pipeline close confident false positives on its own and pass the rest to a human, and because it is retrained on the cases analysts close, its judgement sharpens with use.

The local language model.

- **Features and architecture.** A compact open-weight model served through Ollama, an open-source tool released in 2023 that runs language models locally behind a simple HTTP interface. It is built on the llama.cpp inference engine, manages models by name, and can run families such as Llama, Gemma, and Mistral on ordinary hardware.
- **Use cases.** Running a model this way suits any setting where data cannot be sent to a hosted service or where per-token costs are unwelcome, for narrow text tasks such as summarising and rephrasing.
- **Utility in this SOC.** It is the writer, turning an escalated case's gathered context into a readable brief and nothing more; keeping the model local means no case content leaves for a third-party service, and the writing-only role keeps it clear of any decision.

2.4.9 Comparative summary

Table 2.1 gathers the tools and the role each one fills in the stack.

Table 2.1: The open-source tools making up the SOC and their roles

Tool	Role in the stack
Suricata	Network intrusion detection, signature-based inspection of traffic
Elastic Stack	Log collection, storage, the detection engine, and the analyst dashboards
MISP	Threat-intelligence feeds and indicator matching
TheHive	Case management for escalated alerts
Cortex	Automated enrichment of a case’s observables
n8n	Orchestration of the whole alert pipeline
ML classifier and LLM	Alert triage and the written incident brief

Together these tools cover detection, threat intelligence, case management, orchestration, and the two AI roles. How the triage classifier and the writer are designed to work within the pipeline, and how a human stays in control of every consequence, is the subject of the next section.

2.5 Retained solution

Chapter 1 proposed an open-source, locally run SOC; the survey in this chapter settled the techniques and the tools that make it possible. This section states the solution retained from that survey and the reasoning behind it, then sets out how the result is kept under human control. The detailed design follows in Chapter 3.

2.5.1 The synthesis and rationale

The retained solution is the standard open-source bundle with the missing intelligence added on top. The bundle is already capable: Suricata and the Elastic Stack detect, MISP supplies threat intelligence, TheHive and Cortex manage and enrich cases, and n8n orchestrates the flow between them. All of it is mature, free, and runs on the organisation’s own hardware. What it does not do is automate the two steps that follow an alert, the response and the write-up. The project adds both, driven by a supervised classifier placed between detection and the analyst: from its triage decision, escalated cases are written up as a report and a ticket by a local language model, and confirmed

threats are contained by an automated response, all carried by the same n8n pipeline.

Three choices shape that design, each following from the research question rather than from preference. Triage is given to a supervised model rather than a set of hand-written rules, because a model learns the team's own judgement from the cases analysts close and grows sharper over time, where static rules would need constant tuning to keep pace with a changing alert stream. The language model is confined to writing and never allowed to decide, because a model that only phrases what has already been determined cannot produce a wrong verdict, whether it is mistaken or entirely offline. And the whole system is kept open and local, because a SOC's data should not have to leave the perimeter to be triaged or written up, and because a system that depends on no vendor and no subscription can be run and reproduced anywhere, by the organisation that built it and by the smaller ones the commercial market prices out alike. Taken together, the combination automates the response and the reporting that the open-source bundle left to people, while keeping the control and the openness the commercial AI platforms do not offer.

2.5.2 Human-in-the-loop governance

The design rests on a single rule: the machine triages, the human decides. The classifier's only automatic action is closing a false positive, and even that is fenced. A case is auto-closed only when the model is confident, the alert is low-risk, it is not part of a burst, and it is not flagged for blocking; if any of these fails, the alert goes to a human. This safety floor means the system can never quietly close a high-severity or suspicious alert, so its one automatic outcome is also its safest.

Every consequence beyond that waits for a person, though the work of carrying one out does not. Analysts act through a set of one-click surfaces: at the first tier, a triage decision to confirm, dismiss, or escalate an alert, where escalation, once chosen, runs on its own; and at the second tier, a vetted library of response actions such as blocking an address, isolating a host, or quarantining a file, which can be extended or refined as new needs arise. Choosing the action is the analyst's part; once chosen, a workflow carries it out automatically. Containment shows the discipline clearly: an alert may be marked for blocking, but the block never fires by itself, it is held until an analyst confirms it, and even then it is applied narrowly, to the offending source alone. The verdicts analysts give as they close cases then return to the classifier as fresh training data, closing the loop between the people who run the SOC and the model that learns from them. The result is automation that does the work while leaving every consequential decision to a human, which is what makes the system safe to leave running.

Conclusion

This chapter laid the groundwork the solution rests on. It described how a modern SOC works and how it has moved toward AI augmentation, set out the MITRE ATT&CK framework and the threat intelligence behind the tagging and enrichment, and explained the two techniques the system relies on: a supervised classifier that triages alerts and a language model that writes the analyst's brief. It then examined the open-source tools that make up the stack, one by one, weighing what each is and the role it fills, before drawing them together into the retained solution: the standard open-source bundle, kept whole, with a triage engine added between detection and the analyst so that escalated cases are written up as reports and tickets by the language model and confirmed threats are met by an automated response, while a human stays in control of every consequence. With the background, the tools, and the design choices now settled, the next chapter turns to design and implementation, and shows how these pieces are built and connected.

Chapter 3

Design and Implementation

Introduction

This chapter covers the implementation of the SOC. The previous chapters established the theoretical foundations and reviewed the open-source tools and AI techniques behind the proposed solution, here we focus on technical execution, configuration, and integration. We begin with the deployment topology and the data flow, then move through the detection layer, the n8n orchestration backbone and its alert pipeline, the AI components, case-management and human-in-the-loop response layer. The goal is a clear and replicable account of how the system was built.

3.1 System architecture

The SOC is built entirely from open-source components, organised into four functional layers: telemetry collection, detection, orchestration, and incident response. Figure 3.1 shows the whole system at a glance, from the sensors that feed it to the case workbench where an analyst closes an incident. Telemetry from the monitored hosts is detected by the Elastic stack, orchestrated by n8n with the machine-learning classifier and the language-model writer alongside it, and turned into managed cases in TheHive, Cortex and MISP. The two sections that follow detail the physical side of this picture, the four virtual machines and their overlay network (Section 3.1.1), and the data flow that connects the layers (Section 3.1.2).

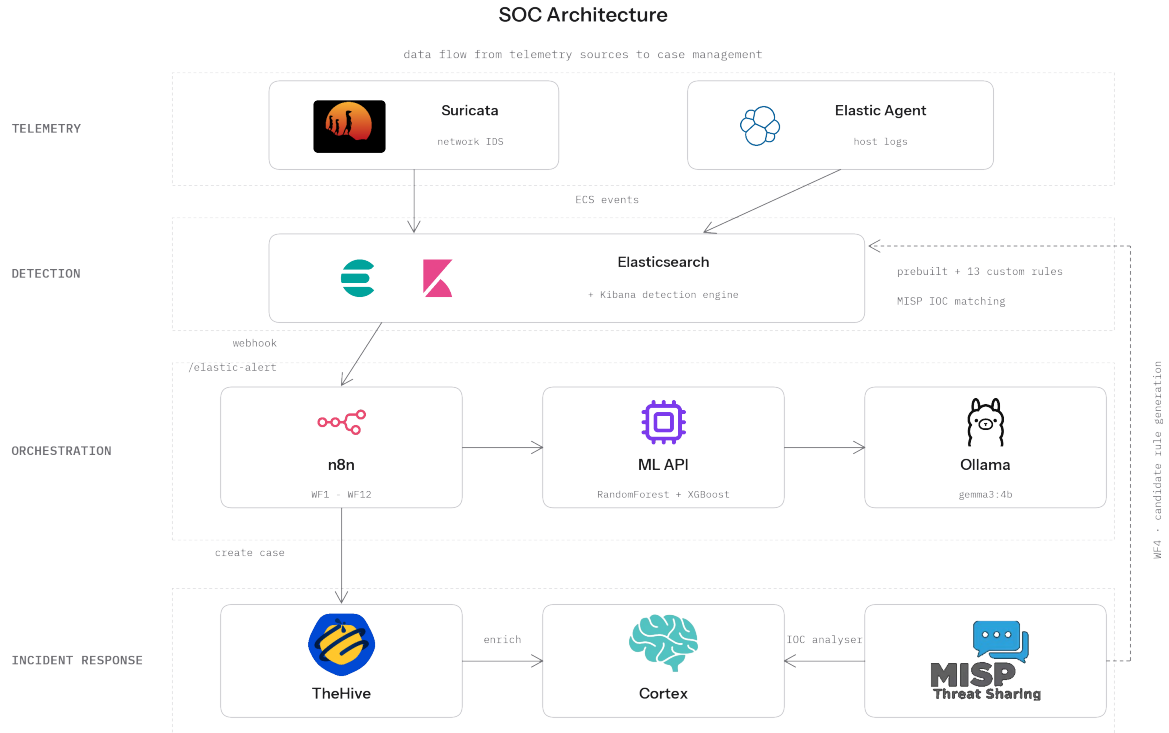


Figure 3.1: SOC architecture

3.1.1 Network Architecture

The SOC runs on four virtual machines spread across two physical hosts (as shown in Figure 3.2). The first host carries the operational core, **SOC-Core**, which runs the SIEM, the n8n orchestration engine and the machine-learning classifier, together with an attacker machine, **kali-attacker**, used only to generate traffic during validation. The second host carries the incident-management machine, **incident-mgmt**, which runs TheHive, Cortex and MISP, together with the deliberately vulnerable target, **victim-lab**, which also runs the network IDS. This pairing keeps the production path, detection on the first host and case management on the second, separate from the lab path formed by the attacker and its target.

The four machines are joined by a ZeroTier overlay network on the flat subnet **192.168.1.0/24**, so they address each other by fixed private IPs no matter where the two hosts physically sit. We chose an overlay over bridged networking because the hosts may be on different physical networks. The local language model is not one of these four machines: it runs on a separate host reached over an SSH tunnel due to resource constraints.

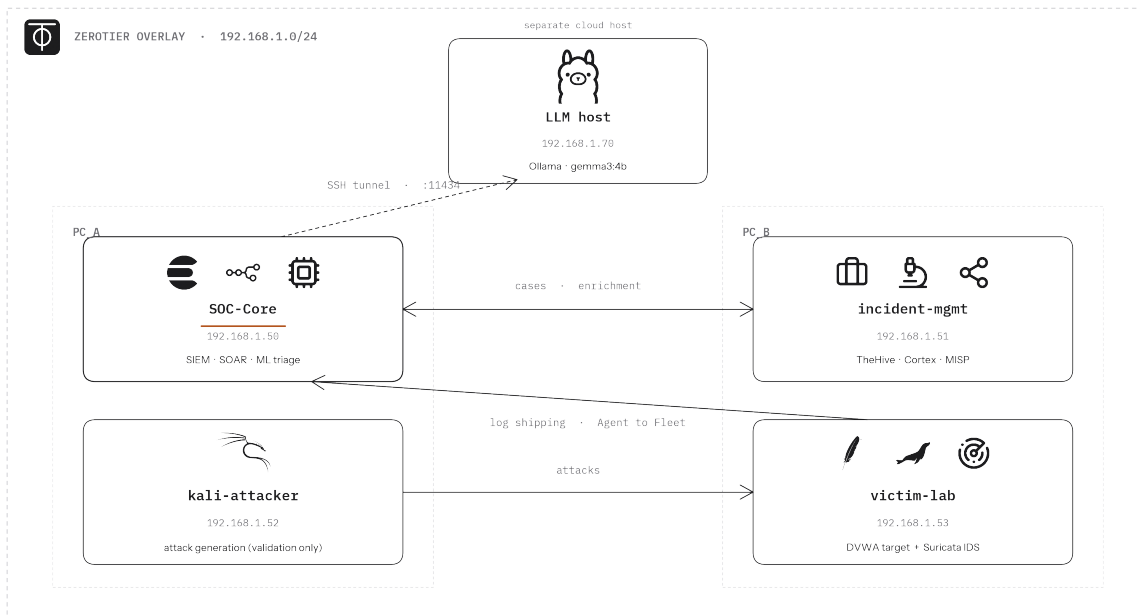


Figure 3.2: Deployment topology

3.1.2 Data flow

Two sources feed the platform (as shown in Figure 3.3): telemetry from the monitored hosts and network, which the Elastic Agent ships into Elasticsearch, and external threat-intelligence feeds pulled into MISP. When Kibana’s detection engine matches a rule, a severity-gated webhook hands the alert to n8n. The alert pipeline normalises it, deduplicates it against the open cases in TheHive, and classifies it with the machine-learning model; a deterministic step then sets the verdict, severity, MITRE mapping and tags, and TheHive either opens a case or auto-closes a confident false positive, with observables added and source addresses annotated from MISP.

From there the data fans out: Cortex enriches the observables, the language-model writer adds a grounded brief on escalation, and a new MISP indicator can become a candidate Kibana rule, imported disabled for review. A human controls the outcome, since the analyst’s verdict closes the case and any host containment runs only after a confirmation click. Two loops close the system: every resolved case retrains the classifier and produces an investigation report. The language model only writes; it never decides.

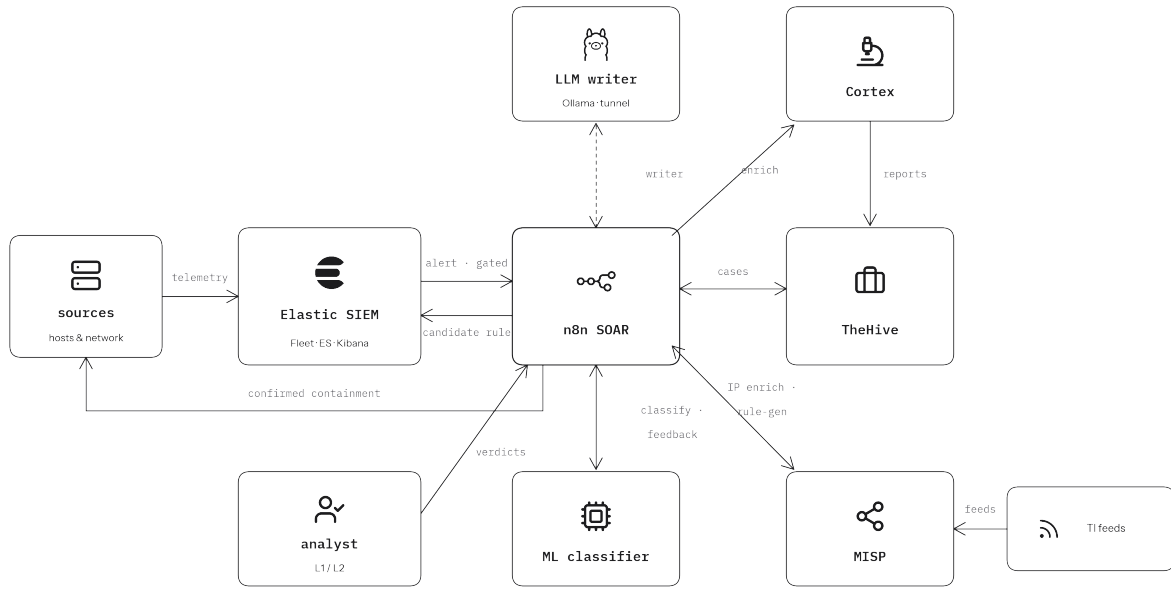


Figure 3.3: Data flow

3.2 Work environment

3.2.1 System Environment

All four lab machines run Ubuntu 26.04 LTS, except the attacker, which runs Kali Linux for its testing tools.

Table 3.1: Virtual-machine resource allocation.

Machine	Hostname	IP	OS	Host	RAM	Disk	Role
VM_A1	SOC-Core	192.168.1.50	Ubuntu 26.04 LTS	PC_A	14 GB	500 GB	SIEM core, SOAR, and AI services
VM_A2	kali-attacker	192.168.1.52	Kali Linux	PC_A	4 GB	80 GB	Attack generation (validation only)
VM_B1	incident-mgmt	192.168.1.51	Ubuntu 26.04 LTS	PC_B	14 GB	200 GB	Incident management and threat intelligence
VM_B2	victim-lab	192.168.1.53	Ubuntu 26.04 LTS	PC_B	4 GB	80 GB	Vulnerable target and network IDS
LLM host	—	192.168.1.70	Ubuntu 26.04 LTS	cloud	8 GB	80 GB	Language-model (Ollama <code>gemma3:4b</code>)

Each machine is sized for the services it carries. The incident-management host is the

tightest, since it stacks Cassandra, a local Elasticsearch, TheHive, Cortex and MISP on one machine. To keep it stable, the JVM heaps are capped and the services are brought up in a fixed staged order rather than in parallel, which avoids out-of-memory failures at boot. For the same reason the language model is kept off the SIEM host, on the separate machine reached over the tunnel, leaving that host's memory for the Elastic stack.

3.2.2 Service endpoints

Table 3.2 lists the open ports and their exposure. Two services are locked down further: the machine-learning classifier and TheHive's backing stores (Cassandra and the local Elasticsearch) bind to localhost, and MISP binds to its ZeroTier address alone. The one port left open to any network is ZeroTier's own (9993/UDP), so the overlay can form over whatever path the two hosts have.

Table 3.2: Service port map.

Host	Port	Service
A1 (.50)	5000	ML classifier API
A1 (.50)	5044 / 5140	Logstash (beats / syslog)
A1 (.50)	5601	Kibana
A1 (.50)	5678	n8n (webhooks)
A1 (.50)	8220	Fleet Server
A1 (.50)	9200	Elasticsearch
B1 (.51)	9000 / 9001	TheHive / Cortex
B1 (.51)	8443 / 8080	MISP
B1 (.51)	9042 / 9200	Cassandra / local ES
B2 (.53)	21 / 22 / 80	vsftpd / SSH / Apache (DVWA)
(.70)	11434	Ollama <code>gemma3:4b</code>

The language model is reached differently. Ollama listens on port 11434 on the separate host, and **SOC-Core** reaches it through an SSH tunnel rather than a direct overlay route, which keeps the endpoint private to that one host and lets the model stay loaded between calls.

3.3 Detection layer

3.3.1 Multi-layer detection

All telemetry reaches the SIEM the same way: an Elastic Agent on the victim host, managed by Fleet Server on **SOC-Core**, ships its host logs and Suricata's events into Elasticsearch, where

Kibana’s detection engine evaluates them. On that stream, detection works in three complementary layers, and because they overlap an attack usually trips more than one. At the network layer, Suricata inspects packets on the victim host against the Emerging Threats Open ruleset (roughly 49,900 signatures, updated daily); a dedicated Kibana catch-all rule promotes its events into the alert stream. At the host and SIEM layer, Kibana’s detection engine runs the 13 custom SOC rules (Section 3.3.2) alongside a selection of the 1,644 installed Elastic prebuilt rules, all carrying native MITRE ATT&CK tags. At the threat-intelligence layer, MISP matches indicators against feeds refreshed every few hours (CIRCL OSINT, Abuse.ch, AlienVault OTX, and the ET compromised-IP list), and a newly published indicator can be turned into a candidate detection rule for an analyst to review.

Two mechanisms keep this volume manageable. A severity gate at the entry to n8n admits only high- and critical-class alerts; lower-severity matches are recorded but go no further. And because the layers overlap, the alert pipeline deduplicates each alert against the open cases in TheHive, so one attack that trips several rules becomes a single case rather than several.

3.3.2 Custom detection rules

Alongside the prebuilt Kibana rules, we author thirteen custom detection rules, SOC-001 to SOC-013, tuned to the services exposed in the lab and to the techniques we expect against them. We write our own because the prebuilt rules target generic enterprise telemetry rather than the lab’s Linux web stack, and because every downstream stage keys off each rule’s identifier and MITRE tag, which a custom rule sets explicitly. Each rule is installed as a Kibana detection rule carrying a native MITRE ATT&CK **threat**, so every alert it raises is mapped to a tactic and technique at the moment of detection rather than enriched afterwards. The set spans credential access, initial access, execution, discovery, persistence, privilege escalation, lateral movement, and exfiltration. Table 3.3 lists them.

Table 3.3: 13 custom SOC detection rules

ID	Name	Tactic	Technique	Severity	Response
SOC-001	SSH brute force (8+/2min)	Credential Access	T1110.001	Medium	webhook
SOC-002	SSH brute force critical (15+/2min)	Credential Access	T1110.001	Critical	webhook + auto-block*
SOC-003	SQL injection attempt	Initial Access	T1190	Medium	webhook
SOC-004	SQL injection critical payload	Initial Access	T1190	High	webhook + auto-block*
SOC-005	XSS attempt	Execution	T1059.007	Medium	webhook
SOC-006	OS command injection	Execution	T1059.004	High	webhook + auto-block*
SOC-007	Port scan (10+/1min)	Discovery	T1046	Low	webhook
SOC-008	Critical system file modified	Persistence	T1098	High	webhook
SOC-009	Web shell detected	Persistence	T1505.003	Critical	webhook + auto-block*
SOC-010	Sudo privilege escalation	Privilege Escalation	T1548.003	High	webhook
SOC-011	Reverse shell command	Execution	T1059	Critical	webhook + auto-block*
SOC-012	SSH login from attacker IP	Lateral Movement	T1021.004	High	webhook
SOC-013	Data exfiltration indicator	Exfiltration	T1048	High	webhook

* Auto-block marks a containment *candidate* only; the block is applied after analyst confirmation, never automatically.

3.4 Orchestration layer

n8n is the orchestration layer: it receives every alert, drives each case through detection, triage, enrichment and response, and links the SIEM to the case workbench. Ten workflows are active. One, the alert pipeline, carries the main path; the rest support it.

3.4.1 Workflow catalogue

Table 3.4 lists the active workflows, each triggered by a webhook or a schedule, with its node count as a rough measure of complexity. The numbering follows the development history; the ten below are the active set.

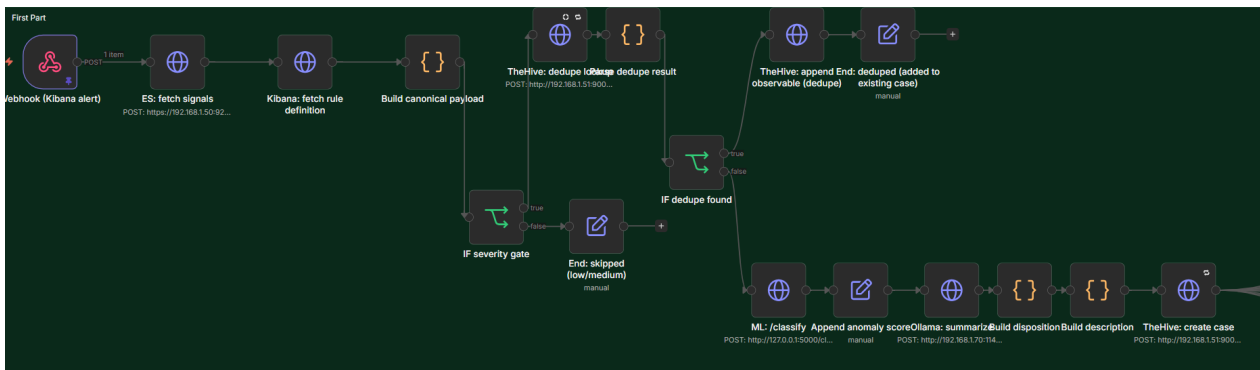
Table 3.4: n8n workflows.

WF	Name	Nodes	Trigger
WF1	Alert pipeline	53	POST /webhook/elastic-alert
WF2	Cortex enrichment	16	TheHive case-created (bridge)
WF3	MISP rule generation	8	POST /webhook/misp
WF4	Weekly maintenance	7	Schedule (Mon 04:00)
WF5	Investigation report	11	Case resolved (webhook)
WF6	Email acknowledgement	7	GET /webhook/email-ack
WF7	L2 escalation writer	11	POST /webhook/l2-escalation
WF8	L2 verdict	12	GET /webhook/l2-verdict
WF9	L1 decision router	20	POST /webhook/l1-decision
WF10	Response action	9	POST /webhook/response-action

3.4.2 The alert pipeline (WF1)

WF1 is the centerpiece, the largest workflow at 53 nodes. It is presented in segments, each a slice of the n8n canvas, following an alert from the moment Kibana fires it to the moment the case is shaped and routed.

- **First segment (as shown in Figure 3.4)** The **Webhook (Kibana alert)** node receives the rule's JSON body; **ES: fetch signals** and **Kibana: fetch rule definition** pull the full alert and its **threat[]** MITRE tags, and **Build canonical payload** normalises it into one rule-agnostic object. The severity gate drops low and medium alerts, dedup folds repeats into an existing case, and new alerts pass through **ML: /classify (p_tp)** and **Ollama: summarize**. Finally **Build disposition**, **Build description** and **TheHive: create case** set the verdict, MITRE, TLP, PAP, assignee and tasks, then open the case.

**Figure 3.4:** WF1 first segment

- **Second segment (as shown in Figure 3.5)** Three branches run off the new case: **Build**

observables and **TheHive: add observable** attach the IoCs (source and destination IPs, victim host, hostname, rule id, file hash, URL); **Build MITRE procedures** and **TheHive: add MITRE procedure** attach the ATT&CK procedure context, and **MISP: search source_ip** (MISP at **192.168.1.51:8443**) queries threat intel, with **MISP: decide push** gating whether a genuinely new IP is pushed back via **MISP: add new IP** and the source tagged **misp:known/new**.

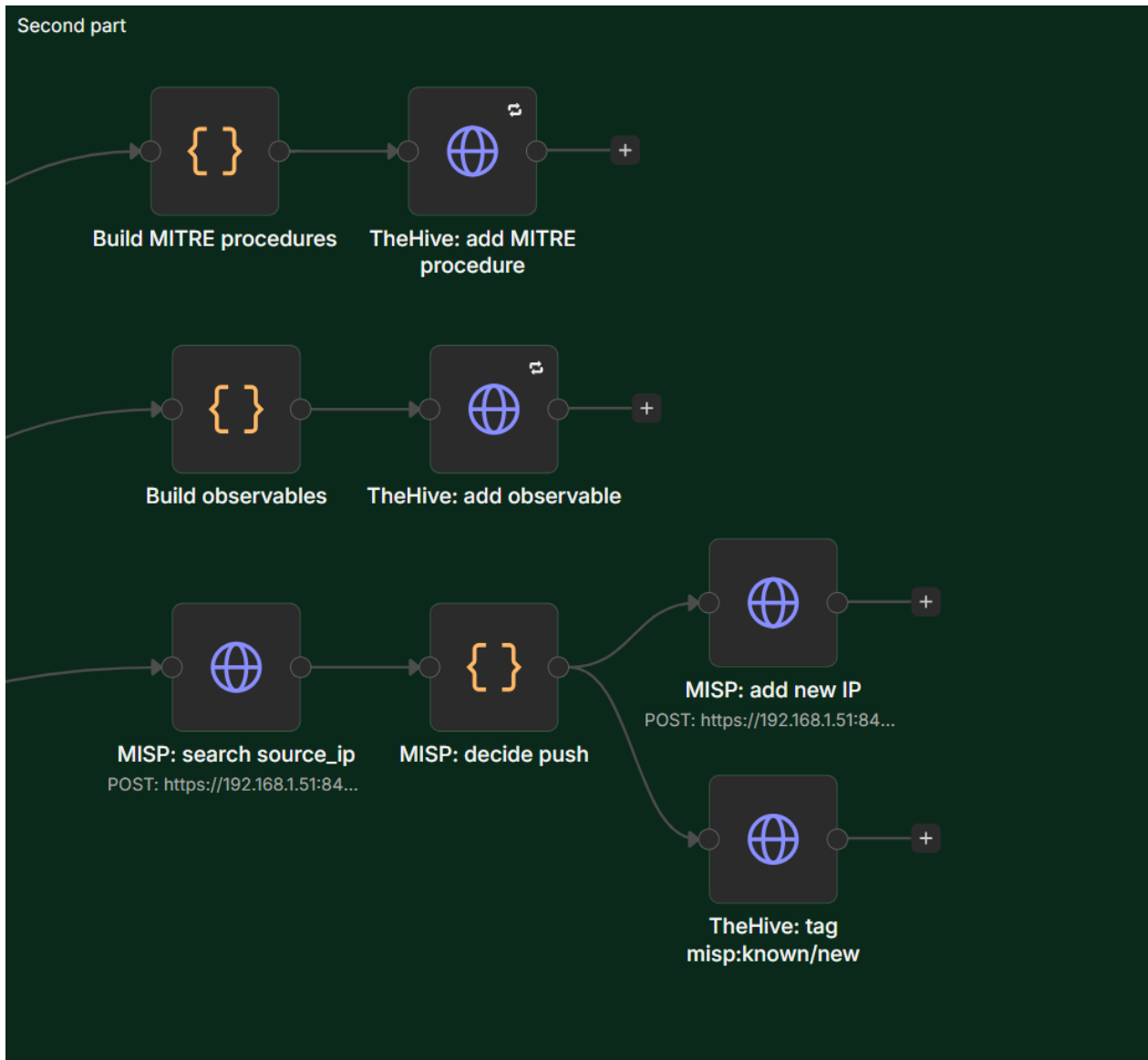


Figure 3.5: WF1 second segment

- **Third segment** (as shown in Figure 3.6) With the case created, the pipeline forks on its disposition: **IF auto-close FP?** sends a confident low-risk false positive straight to **TheHive: PATCH close (FP)**; Ollama: **tag case (192.168.1.70:11434)**, **Build merged tags** and **TheHive: PATCH case tags** append the grounded ML and MITRE tags; **IF needs L2?**

fires **WF7: trigger L2 escalation** (192.168.1.50:5678) on escalation dispositions; and **IF auto_block?** never blocks on its own — it posts **TheHive: comment awaiting-L1** and ends at **End: case created (no auto-block)**, leaving containment for a human click.

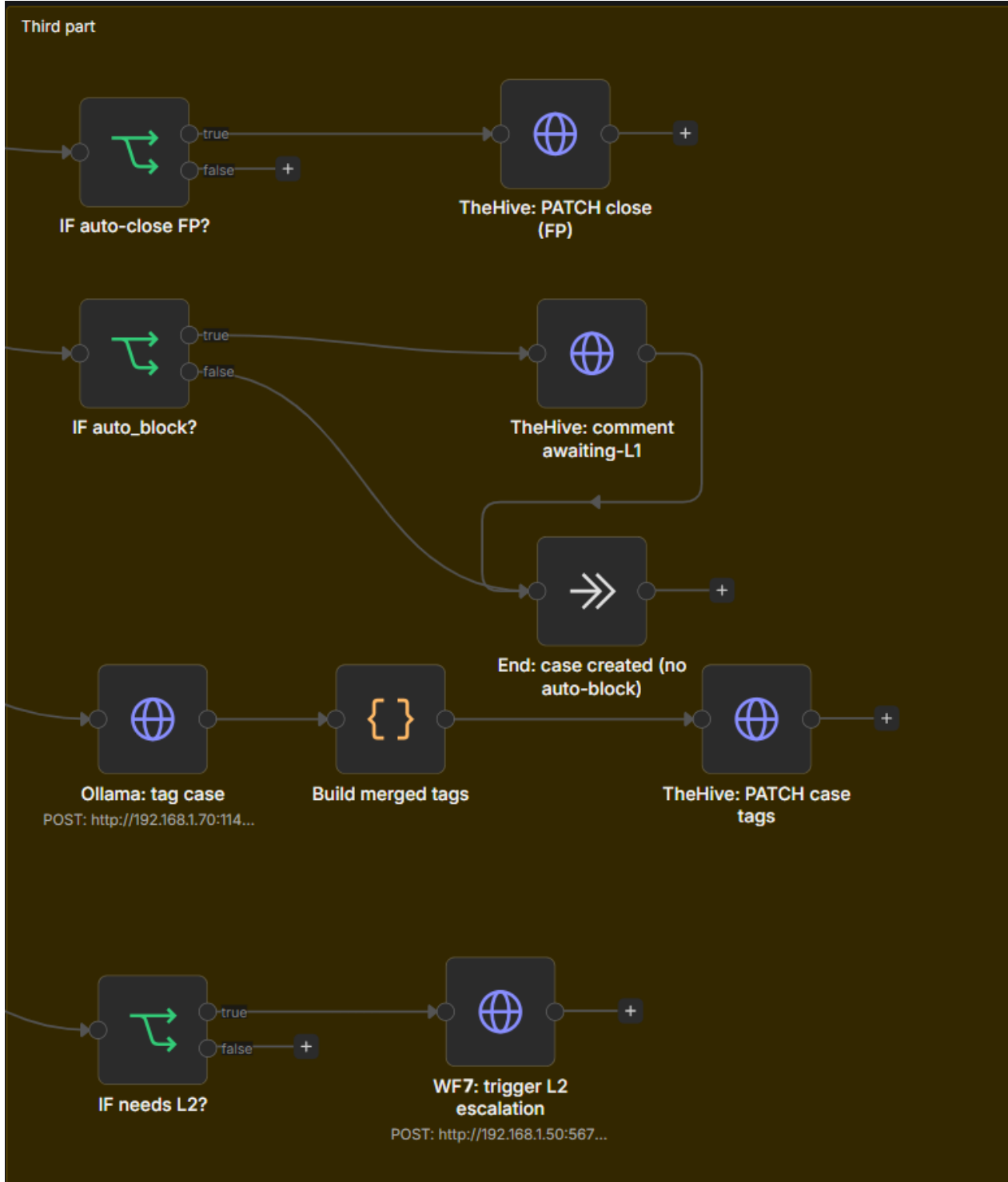


Figure 3.6: WF1 third segment

- **Fourth segment** (as shown in Figure 3.7) For a case that needs an analyst, **Notification config** and **IF severity high/critical** gate the notification: high and critical alerts go to **Ollama: write alert email** (192.168.1.70:11434) for a grounded body and **Build email HTML** for the layout, while everything else ends at **End: notify skipped**. **IF env (test/prod)** then routes delivery to **SMTP: send to analyst** in test or **SMTP: send via Mailtrap Live (prod)** in production. The email carries the confirmation link the analyst clicks to acknowledge the alert or authorise containment.

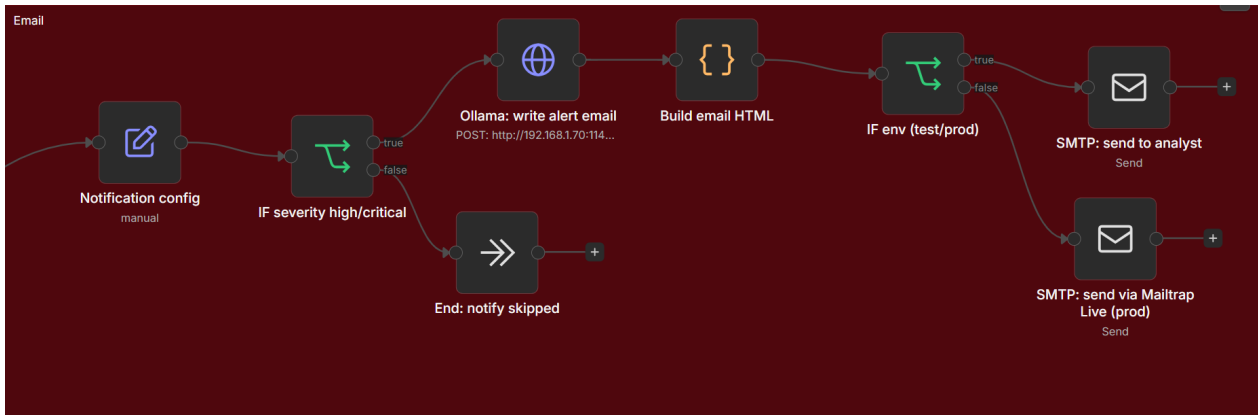


Figure 3.7: WF1 fourth segment

3.4.3 Supporting workflows

The other nine workflows support the alert pipeline, each triggered by a webhook or a schedule (Table 3.4):

- **WF2: Cortex enrichment.** When a case is created it dispatches each observable to the Cortex analyzers that accept its data type (`cortexId=SOC-LAB-Cortex`), waits for the jobs, and posts a grounded threat-intelligence page back to the case.
- **WF3: MISP rule generation.** On a published MISP event (`POST /webhook/misp`) it extracts the indicators, has the language model emit a `{kibana_rule, suricata_rule, mitre}` object, validates it, and imports the Kibana rule *disabled* for analyst review.
- **WF4: Weekly maintenance.** Every Monday at 04:00 it retrains the classifier on the last 30 days, checks ML and Elasticsearch health, counts recent alerts, and opens a maintenance case as an audit record.
- **WF5: Investigation report.** When a case is resolved it calls `/feedback` with the resolved label, gathers the full case context, and posts a language-model investigation report to the

case.

- **WF6: Email acknowledgement.** The analyst’s click on the email link (GET `/webhook/email-ack`) is recorded as a queryable **email-acked:<user>** tag rather than a comment.
- **WF7: L2 escalation writer.** Fired by WF1 on escalation (POST `/webhook/l2-escalation`) it writes a L2 brief and sets four **soc-*** custom fields (detailed in Section 3.6.1).
- **WF8: L2 verdict.** The verdict link (GET `/webhook/l2-verdict`) completes the case’s mandatory tasks, sets the resolution status, posts a verdict comment, and feeds the label to `/feedback`.
- **WF9: L1 decision router.** The L1 decision (POST `/webhook/l1-decision`) switches on `tp/fp/escalate_l2`: a verdict runs the same close-then-feedback recipe as WF8, while an escalation tags the case and emails the L2 on-call.
- **WF10: Response action.** Runs a vetted response script (block or isolate a host, kill a process, disable a user, quarantine a file, or delete a registry key) only when **confirm:true** after an analyst click (Section 3.6.1).

3.5 AI Integration

Two models sit in the pipeline under one rule: the machine-learning classifier decides triage, the language model only writes, and an analyst approves every consequence.

3.5.1 Supervised classifier

The classifier is a Flask service (systemd `ml-api`) on `192.168.1.50:5000`. It frames triage as supervised binary classification: true positive versus false positive, and emits a true-positive probability `p_tp` for every alert. RandomForest is the primary model and XGBoost is trained alongside it as a benchmark, both served from the one process.

The base corpus is the CICIDS2017 intrusion-detection dataset, about 2.83 M labelled flow records, split 70/15/15 into training, validation and test sets. That public base is then augmented with *TheHive-resolved labels*: every case an analyst closes as TruePositive or FalsePositive is pulled back by `pull_thehive_labels.py` as a real label, drawn from the same upstream the live pipeline sees. The two sources play different roles: CICIDS2017 is a large, openly citable base to learn general

attack-versus-benign structure, but its features are only a proxy for the SOC’s own true/false-positive judgements and carry a real risk of label leakage, so the operationally defensible signal comes from the resolved-case labels, which describe exactly the decisions the classifier must reproduce.

The 15 features are deliberately rule-agnostic: risk score, severity, rule source, rule category, MITRE tactic and technique encodings, source-IP properties (internal / management), time-of-day, the `auto_block` flag, the alert count in the window, and a learned `rule_false_positive_rate_30d`, a per-rule trust score updated on every `/feedback`. The model generalises across the thirteen custom rules, the prebuilt rules and Suricata SIDs. That continuously learning per-rule trust score is the project’s central contribution. The complete feature list and the model’s inference path are reproduced in Appendix B (Figures B.1 and B.2).

- **Endpoints:** `/health`; `/classify` (returns `predicted_label`, `confidence`, `recommended_action` and the true-positive probability `p_tp`); `/feedback` (the learning loop, called by WF5, WF8 and WF9); `/metrics`; and `/train` (WF4).
- **Thresholds:** `CONFIDENCE_HIGH` = 0.85, `CONFIDENCE_LOW` = 0.60, `RISK_HIGH` = 70, with `recommended_action` drawn from `{auto_close_fp, auto_escalate, send_to_analyst, queue_low_priority}`.
- **Safety floor.** `auto_close_fp` is permitted only when the model predicts a false positive and $\text{confidence} \geq 0.85$ and $\text{risk} < 70$ and the alert is not bursty and carries no `auto_block` flag; anything else goes to an analyst. The model can never auto-close a high, critical, bursty or blocked alert. The floor’s logic is reproduced in Figure B.3.

3.5.2 Local language-model

The language model supplies the generative features the n8n workflows depend on: alert summarisation, the analyst email and auto-tagging in WF1, the Cortex report synthesis in WF2, the candidate-rule drafting in WF3, the investigation report in WF5, the escalation brief in WF7, and the escalation email in WF9. It is built on Ollama, an open-source runtime that exposes an HTTP inference API, and serves Gemma 3 4B, an open-weight instruction-tuned model from Google DeepMind. Its role is confined to writing: it produces grounded text and draft artefacts but takes no decision, since severity, verdict, MITRE mapping and response actions are all fixed by deterministic logic. These features are summarised in Table 3.5.

Table 3.5: LLM-assisted features

Feature	Workflow	Output	Purpose
Alert summarisation	WF1	free text	One-line grounded context in the case and alert email
Auto-tagging	WF1	JSON	Proposed case tags
Analyst email	WF1	free text	Body of the alert email sent to the analyst
Cortex synthesis	WF2	free text	Threat-intelligence paragraph from the analyzer reports
Rule generation	WF3	JSON	Candidate { <code>kibana_rule</code> , <code>suricata_rule</code> , <code>mitre</code> }
Investigation report	WF5	JSON	Structured multi-section report written on case close
Escalation brief	WF7	free text	The grounded “Situation” section
Escalation email	WF9	free text	Email sections for the on-call analyst

Every feature follows the same pattern. The workflow assembles the relevant context (the alert payload, the case, or, for the richer features, the result of an Elasticsearch related-alert search) and passes it to the model under a prompt template that fixes the model’s role and the expected output. The model writes grounded text or a structured draft. The generated text is then written back into the case as the relevant field, comment or page. The arrangement is empty-safe: if the model is unavailable, the workflow still produces its result from the deterministic logic alone, with the generated text omitted. Representative prompt templates — alert correlation, the L2 escalation brief and the remediation recommendation — are reproduced in Appendix B (Figures B.4–B.6).

The model is not fine-tuned on SOC data. All customisation lives at the prompt layer. The decision keeps the inference setup portable (any Ollama-compatible model can be swapped in without retraining), keeps the training cost at zero, and avoids the data-privacy questions that come with shipping internal SOC data to a fine-tuning pipeline.

3.6 Case management, enrichment and response

The incident side of the SOC runs TheHive as the case workbench, Cortex for observable enrichment, and MISP for threat intelligence. Cases are created by the pipeline, enriched automatically, and closed only by an analyst.

3.6.1 The case workbench

- **TheHive 5.7.1.** The analyst’s single pane of glass: cases are created by WF1, enriched by WF2 and the writer, and closed by WF8 or WF9. In TheHive 5.7 a case is closed not by a `Closed` status but by a resolution value (`TruePositive`, `FalsePositive`, `Indeterminate`), and any mandatory task blocks closing, which is why the decision workflows complete the

open mandatory tasks before setting the status. Because task fields (status, assignee, group, due date, mandatory flag, description and order) can be set at creation time, the entire two-class task list is materialised in the single `create case` call. Beyond its fields and tasks, each case carries a **TTPs** tab, populated with the MITRE ATT&CK procedures WF1 attaches from the alert's `threat[]`, and wiki-style **pages**: a Threat Intel page written by WF2 from the Cortex synthesis, and an Investigation Report page written by WF5 when the case is closed. Four custom fields are registered in its schema and populated by WF7 on escalation: `soc-urgency` (float), `soc-blast-radius` (string, one of single-host / subnet / crown-jewel), `soc-kill-chain-stage` (string) and `soc-containment-window-min` (integer minutes).

- **Cortex 4.0.1.** Cortex provides two faculties: analyzers for enrichment and responders for action. For enrichment, WF2 dispatches each observable to the analyzers that accept its data type (IP, hash, URL, domain), using free analyzers such as AbuseIPDB, VirusTotal and MaxMind GeoIP (as shown in Figure 3.8).



Figure 3.8: Enrichment flow

Cortex also exposes eight SOC responders an analyst runs from a case; each is a thin shim that posts to an n8n webhook (the verdict responders to WF9, the containment responders to WF10), so the responder is only the button and the workflow holds the logic (Section 3.6.3).

- **MISP 2.5 Feeds** (CIRCL OSINT, Abuse.ch, AlienVault OTX, the ET compromised-IP list) refresh every six hours. MISP is used two ways: WF1 annotates the source IP with a `misp:known/new` tag and pushes genuinely new IPs back, and WF3 turns a new event into a candidate detection rule as demonstrated in Figure 3.9.

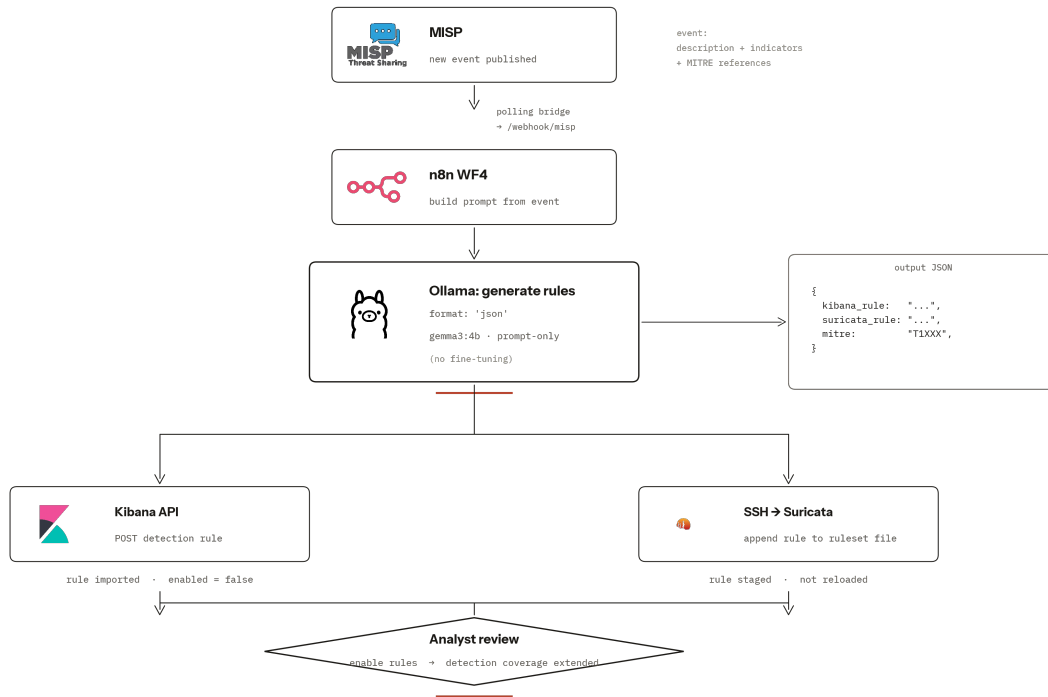


Figure 3.9: Rule generation flow

3.6.2 The case-quality model

Every auto-created case is shaped deterministically by WF1’s `Build disposition` node, never by the model:

- **Title:** the rule id and name, the source and destination and the acting user, with a (×N) suffix when N alerts were aggregated.
- **TLP:** AMBER+STRICT by default.
- **PAP:** RED for stealthy or externally-sourced alerts, GREEN for a confident true positive that is an auto-block candidate, AMBER otherwise.
- **Assignee:** always set: a false positive to the `soc-bot` account, a true positive or override to L2, a mid-confidence case to L1.

- **Tags:** a namespaced set (`project:`, `env:`, `rule:`, `mitre:<tactic>/<technique>`, `disposition:`, `ml:*`).
- **Description:** deterministic Markdown with a summary, a facts table, MITRE names are looked up from the alert's `threat[]` (`Unmapped` when absent) and pivot links to the kibana alert and the victim.

Tasks follow a two-class model, keyed by a vetted playbook rather than generated:

- **Class-1 (automation).** Always present, auto-run and auto-completed, owned by the automation account (ingestion, enrichment, classification, disposition, and closure if auto-closed).
- **Class-2 (human).** Created only on escalation, assigned to L1 or L2, left open with an SLA, and mandatory at the decision points (review, containment approval, scope/hunt, closure).

A confident false positive therefore carries Class-1 tasks only and no human work; an escalated case carries the completed Class-1 tasks plus open Class-2 tasks.

3.6.3 Human-in-the-loop response

An analyst acts on a case from the TheHive interface by running one of eight Cortex responders (Table 3.6). The responders are thin shims: each one only posts the chosen verdict or action to an n8n webhook and returns, so all the logic lives in the workflow it triggers and there is a single decision path with no duplicated logic. Three responders drive the L1 decision through WF9 (`/webhook/l1-decision`) and five drive the containment library through WF10 (`/webhook/response-action`). The responder source, its manifest, the orchestrator and the action scripts are reproduced in Appendix C.

Table 3.6: The eight Cortex responders and the workflows they trigger.

Responder	Routes to	Effect
SOC_Confirm_TP	WF9	Completes the mandatory tasks, closes the case as TruePositive, sends feedback to the classifier
SOC_Confirm_FP	WF9	Completes the mandatory tasks, closes the case as FalsePositive, sends feedback
SOC_Escalate_L2	WF9	Tags the case escalated, generates the L2 brief and email, notifies the on-call analyst; the case stays open
SOC_Autoblock_Isolate	WF10	Blocks the attacker's source IP or isolates the host
SOC_Kill_Process	WF10	Terminates the offending process
SOC_Disable_User	WF10	Disables the implicated account
SOC_Quarantine_File	WF10	Quarantines a malicious file
SOC_Delete_Registry	WF10	Removes a malicious Windows registry key

Containment is never automatic. A rule flagged for auto-block does not fire on its own: WF1 posts an awaiting-confirmation comment and surfaces the action both as a link in the alert email and as a responder, and only an explicit click triggers WF10. For a network block, WF10 applies a selective `iptables -I INPUT -s <attacker> -j DROP` on the victim host through the `soc-response` key, so only the offending source is cut off and the SOC host keeps its access; every action posts its result as a comment and appends the response tags without overwriting.

Conclusion

This chapter documented the deployment of the autonomous SOC, including the system design, the interaction of its core components, which are the Elastic Stack, Suricata, TheHive with Cortex and MISP, n8n, and the AI layer of a supervised triage classifier and a local Gemma 3 4B writer, and the configurations used. The completed system is a working pipeline that automatically ingests alerts, deduplicates and enriches them, triages and summarises them with AI, and routes them into TheHive for analyst review. With the implementation described, the next chapter evaluates the solution, its effectiveness, and its impact on Security Operations Centre capabilities.

Chapter 4

Evaluation

This chapter presents the evaluation of the autonomous SOC described in the previous chapter. Its purpose is to validate the operational behaviour of the integrated system and to assess its ability to detect, triage and respond to realistic attacks. The evaluation proceeds in four stages. The test plan and environment define how the system is exercised and which metrics are collected. The functional tests then run each automation end to end on the live platform, from a raw attack to a resolved case, so that every link of the detection-to-response chain can be observed in isolation. The performance of the supervised triage model is then measured in the classical terms of detection engineering. The chapter closes with a discussion of the operational impact, the limitations surfaced during validation, and the work that remains.

4.1 Test plan and environment

The system is evaluated on the four-virtual-machine laboratory presented in Chapter 3, connected over a ZeroTier overlay and divided into a SOC plane and a target plane, with the language model hosted separately. The SOC plane carries the detection, orchestration and intelligence services; the target plane hosts the deliberately vulnerable victim and the attacker.

4.1.1 Attack emulation and MITRE scenarios

All offensive actions are launched from the Kali host (192.168.1.52) against the victim lab. The attacker is not part of the production data path; it exists only to generate controlled, repeatable traffic, so that the measurements reflect the behaviour of the pipeline under known conditions. Each scenario is mapped to the MITRE ATT&CK technique it exercises and to the custom detection rule expected to fire, which lets detection be reported as coverage against the framework rather than as an isolated event. Table 4.1 lists the scenarios used in the evaluation.

Table 4.1: Attack scenarios emulated from the Kali host.

Scenario	Tool / action	Rule	MITRE
SSH brute force	hydra	SOC-002 (critical)	T1110.001 Password Guessing (TA0006)
Port scan	nmap	SOC-007 (low)	T1046 Network Service Discovery (TA0007)
Authorized SSH login (FP)	ssh from a trusted host (.51)	SOC-012 (high)	T1021.004 Remote Services: SSH (TA0008)
Threat-intel rule	publish a malicious-IP event in MISP	WF3 (AI rule generation)	—

4.1.2 Evaluation metrics

Four families of metric are collected, each answering a distinct question. Detection efficacy records whether the expected rule fires on each attack. Routing correctness records whether the orchestrator places the resulting case in the correct disposition, namely auto-close, Tier 1, Tier 2 or an override. The quality of the triage model is measured with precision, recall, the F1 score, the area under the ROC curve, the confusion matrix and the feature importances, all reported in Section 4.3. Finally, the operational metrics, treated in Section 4.4, capture the alert-reduction ratio, the per-stage latency and the analyst effort relative to a manual baseline.

4.2 Functional tests

This section validates each automation of the pipeline end to end, from a raw attack to a resolved case. Every scenario was executed on the live platform, and the case, rule and execution identifiers reported below are taken directly from it. Together the scenarios cover the full decision space of the orchestrator: the deterministic routing of alerts, the handling of a false positive and a true positive at Tier 1, the escalation of a confirmed incident to Tier 2, the continuous-learning feedback, the confirmed containment of an attacker, the generation of a detection rule from threat intelligence, the weekly self-maintenance and the automated investigation report.

4.2.1 Detection scenarios

Port scan: A full TCP SYN scan is launched against the victim lab with `nmap -sS -p- 192.168.1.53`, probing every port for an open service as demonstrated in Figure 4.1.

```
(kali㉿kali)-[~]
$ sudo nmap -Pn -sS -T4 -p- 192.168.1.53
Starting Nmap 7.99 ( https://nmap.org ) at 2026-06-01 12:17 -0400
Nmap scan report for 192.168.1.53
Host is up (0.074s latency).
Not shown: 65401 closed tcp ports (reset), 130 filtered tcp ports (no-response)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
9993/tcp  open  palace-2
MAC Address: D2:4E:4A:DE:0D:DB (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1169.81 seconds
```

Figure 4.1: Port scan launched from the Kali machine.

The scan trips rule SOC-007 (port scan, more than ten connection attempts in one minute), which Kibana raises and maps to T1046 (Network Service Discovery, tactic TA0007).

Columns 18 Sort fields 1 41 alerts Fields Updated 4 minutes ago Grid view Additional filters Group alerts by: None							
Actions	@timestamp	Rule	Assignees	Severity	Risk Score	Reason	
   	Jun 1, 2026 @ 17:46:51.403	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:41:49.014	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:38:13.237	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:37:15.004	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:36:50.722	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:36:50.719	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:36:50.714	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:36:50.710	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:36:13.934	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:35:14.275	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:34:13.512	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:33:13.487	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:32:12.785	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:31:52.023	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:31:52.000	SOC: Suricata alert (catch-...		medium	50	network, intrusion_detection evi	
   	Jun 1, 2026 @ 17:31:12.369	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:30:16.438	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	
   	Jun 1, 2026 @ 17:29:13.446	SOC-007 Port scan (10+ di...		low	21	event with source 192.168.1.5;	

Figure 4.2: SOC-007 alert raised in Kibana Security.

WF1 receives the alert through the detection webhook, canonicalises it into the rule-agnostic schema, and creates case #153 in TheHive. The triage model scored the alert as a true positive with a confidence of 0.99, and the case is routed to Tier 2.

Chapter 4. Evaluation

The screenshot displays the TheHive interface for case #153, titled "[SOC-007] Port scan (10+ distinct destination ports / 1 min) — 192.168.1.52 → victim-lab (x15)". The left sidebar shows the case details, including the assignee (L2 Analyst), status (True Positive), and time metrics (57 minutes, 21s). The main panel shows the case description, which states: "RESOLVED → TruePositive - ML verdict TP (P(TP)=99.0%) confirmed by L1 - 0 overrides — ML correct, no override needed". Below this, a table of facts is provided:

Field	Value
Verdict (ML)	TP (confidence 99.0%) — correct
P(true positive)	99.0%
Source IP	192.168.1.52 (A2 / Kali)
Destination / victim	victim-lab (192.168.1.53)
Scan	full TCP SYN scan, all 65,535 ports (nmap -sS -T4 -p-)
Rule	SOC-007 — Port scan (10+ distinct destination ports / 1 min)
MITRE	T1046 Network Service Discovery / TA0043 Reconnaissance
Severity	low (risk_score=21)
Alerts in window	15

The right sidebar shows the comments section, with two comments from L1 Analysts. The first comment states: "**L1 triage note** ML scored this **true_positive** (P(TP)=99.0%) and it is correct — a single source (192.168.1.52) sweeping the full TCP port range on victim-lab in ~1 min is unambiguous reconnaissance (T1046). **No override needed: verdict CONFIRMED TruePositive.** Early kill-chain; monitor 192.168.1.52 for follow-on exploitation, block if scanning persists." The second comment states: "**L1 closure** ML verdict TruePositive (99%) is correct — confirmed reconnaissance (full TCP port scan from 192.168.1.52 against victim-lab). No override needed; closing as **TruePositive**. Monitoring 192.168.1.52 for follow-on exploitation."

Figure 4.3: Case #153 created in TheHive from the port scan.

SSH brute force: A credential-guessing attack is launched against the SSH service with hydra, producing 58 failed logins against the account `testuser1` on the victim lab.

```
(kali@kali) ~$ hydra -l testuser1 -P /tmp/rock50.txt -t 4 -W 1 ssh://192.168.1.53 -v
Hydra v9.6 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2026-05-31 14:34:37
[DATA] max 4 tasks per 1 server, overall 4 tasks, 50 login tries (l:1/p:50), ~13 tries per task
[DATA] attacking ssh://192.168.1.53:22/
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "123456" - 1 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "12345" - 2 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "123456789" - 3 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "password" - 4 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "iloveyou" - 5 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "princess" - 6 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "1234567" - 7 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "rockyou" - 8 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "12345678" - 9 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "abc123" - 10 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "nicole" - 11 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "daniel" - 12 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "babygirl" - 13 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "monkey" - 14 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "lovely" - 15 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "jessica" - 16 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "654321" - 17 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "michael" - 18 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "ashley" - 19 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "qwerty" - 20 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "111111" - 21 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "iloveu" - 22 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "000000" - 23 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "michelle" - 24 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "tiger" - 25 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "sunshine" - 26 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "chocolate" - 27 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "password1" - 28 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "soccer" - 29 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "anthony" - 30 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "friends" - 31 of 50 [child 2] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "butterfly" - 32 of 50 [child 3] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "purple" - 33 of 50 [child 0] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "angel" - 34 of 50 [child 1] (0/0)
[ATTEMPT] target 192.168.1.53 - login "testuser1" - pass "jordan" - 35 of 50 [child 2] (0/0)
```

Figure 4.4: The SSH brute force launched from the Kali host with hydra.

After fifteen failures within two minutes the activity trips rule SOC-002 (SSH brute force,

critical), which Kibana raises with a critical severity and marks as an auto-block candidate, mapped to T1110.001 (Password Guessing, tactic TA0006).

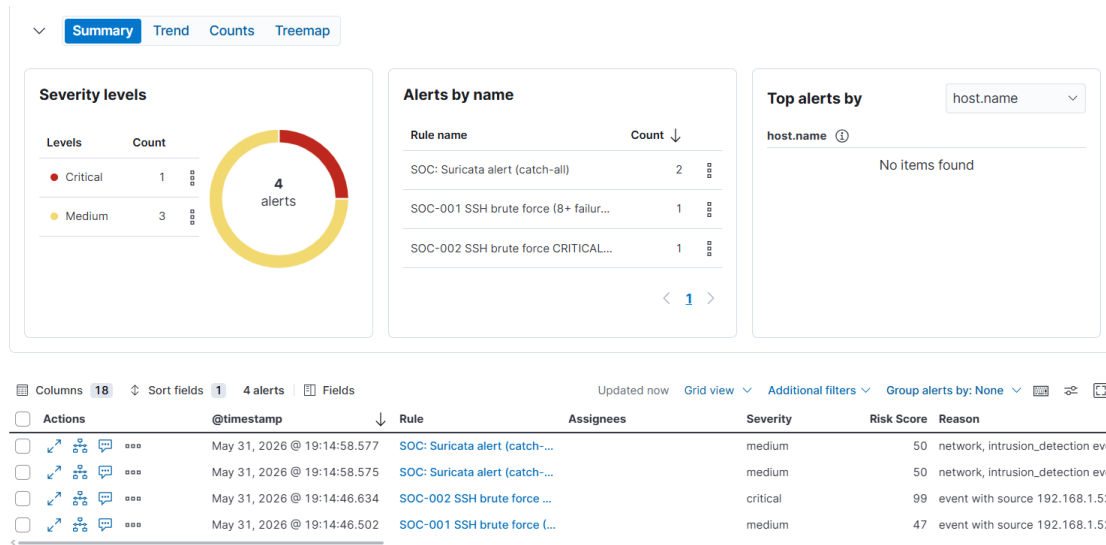


Figure 4.5: The SOC-002 critical alert raised in Kibana on the brute force.

WF1 canonicalises the alert and creates case #135. The triage model scored it as a false positive with a confidence of 0.879, a miss examined in Section 4.3; because the alert is critical and carries the auto-block flag, the safety floor refused to auto-close it and routed the case to Tier 1 for confirmation.

Case #135 [SOC-002] SSH brute force CRITICAL (15+ failures / 2 min) — 192.168.1.52 → victim-lab (as testuser1) (*58)

Tags: ai-confidence:high, asset:critical, assignee:l1, chain:initial-access, dedupe:custom_soc:SOC-002|192.16..., disposition:analyst-review-l1, env:lab, events:58, l1:awaiting-decision, logsource:custom-soc, mitre:T1110.001, mitre:TA0006, ml:action:send-to-analyst, ml:confidence:0.88, ml:confidence:high, ml:model:randomforest-v2, ml:verdict:fp, pipeline:v2, project:pfe, provenance:auto-created, rule:SOC-002, target-account:testuser1, tlp:amber-strict

Description: OPEN → L1 triage · ML verdict FP · P(TP)=12.1% · awaiting L1 decision (override pending)

What happened: Detection SOC-002 fired: SSH brute force CRITICAL (15+ failures / 2 min). 58 failed SSH authentication attempts originated from 192.168.1.52 (A2 / Kali attacker host) against victim-lab (192.168.1.53), all targeting the account testuser1, inside a ~2-minute window — far above the 15-failure critical threshold.

Facts:

Field	Value
Verdict (ML)	FP (confidence 87.9%) — under L1 review
P(true positive)	12.1% (ML)
Source IP	192.168.1.52 (A2 / Kali)
Destination / victim	victim-lab (192.168.1.53)
Targeted account	testuser1
Failed attempts	58
Rule	SOC-002 — SSH brute force CRITICAL (15+ failures / 2 min)
MITRE	T1110.001 Password Guessing / TA0006 Credential Access
Severity	critical (risk_score=99) · auto-block requested

Machine triage (L1 automation):
ML classifier (RandomForest): false_positive, confidence 87.9%. recommended_action = send_to_analyst — the confident-FP was withheld from auto-close by the safety floor (auto_block + critical severity) and routed to L1. The model saw alert_count_in_window=1 (the threshold rule emits a single aggregated signal) while the true burst was 58; alert_count is the model's dominant feature, which drove the FP score. Documented data-fidelity limitation.

Comments:

- # Autoblock requested by rule
Rule 'SOC-002' flagged 'auto_block=true'.
Action required an L1 analyst must confirm by clicking the **SOC_Autoblock_Isolate** Cortex Responder on this case. The block is no longer fire-and-forget (Phase 20).
Source IP: '192.168.1.52'
31/05/2026 19:16
- **L1 triage note** ML scored this **false_positive** (P(TP)=12.1%) but under-counted the burst (alert_count=1 vs 58 actual - threshold-rule aggregation). Raw events show 58 failed SSH logins from 192.168.1.52 against testuser1 on victim-lab in ~2 min - a clear credential-access brute force (T1110.001). **Recommended L1 decision: overturn to TruePositive + ssh_autoblock on the source.**
01/06/2026 17:40 - Edited

Figure 4.6: Case #135 created in TheHive from the brute force.

Authorized SSH login: A legitimate SSH login is performed from a trusted, non-management host (.51) to the account testuser1 on the victim lab, representing routine administrative activity rather than an attack. It is included to exercise the pipeline's handling of a benign trigger.

```

Jun 1 22:07
testuser1@victim-lab: ~ — ssh testuser1@192.168.1.53

vboxuser@incident-mgmt:~$ sudo ssh testuser1@192.168.1.53
[sudo: authenticate] Password:
testuser1@192.168.1.53's password:
Welcome to Ubuntu 26.04 LTS (GNU/Linux 7.0.0-15-generic x86_64)

 * Documentation:  https://docs.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

31 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

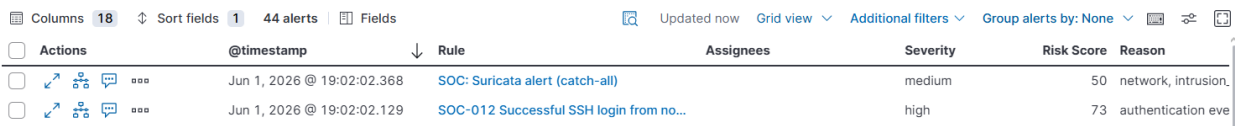
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Mon Jun  1 18:44:55 2026 from 192.168.1.51
testuser1@victim-lab:~$

```

Figure 4.7: The authorized SSH login from the trusted host.

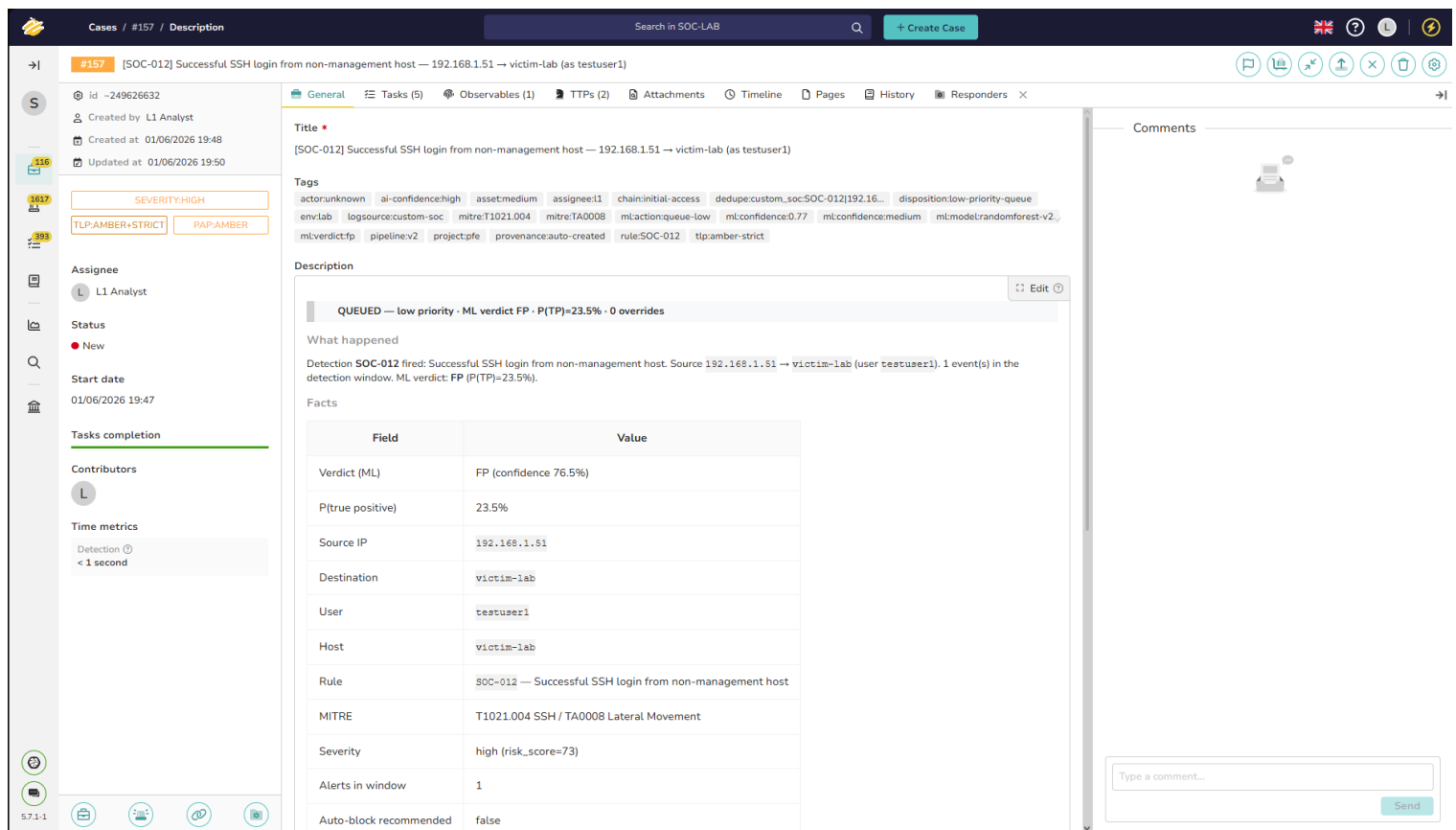
The login trips rule SOC-012 (SSH login from a flagged source), which Kibana raises with a high severity and maps to T1021.004 (Remote Services: SSH, tactic TA0008).



Actions	@timestamp	Rule	Assignees	Severity	Risk Score	Reason
	Jun 1, 2026 @ 19:02:02.368	SOC: Suricata alert (catch-all)		medium	50	network, intrusion.
	Jun 1, 2026 @ 19:02:02.129	SOC-012 Successful SSH login from no...		high	73	authentication eve

Figure 4.8: The SOC-012 alert raised in Kibana on the authorized login.

WF1 creates case #157. The triage model scored the login as a false positive with a confidence of 0.77 (a true-positive probability of 23.5%); because the confidence is below the auto-close threshold and the rule's risk of 73 exceeds the threshold of 70, the safety floor declined to auto-close it and routed the case to the Tier-1 low-priority queue. The scenario illustrates the human-in-the-loop principle of the design: the model may propose a false positive, but a human confirms it for any alert that is not trivially safe.



Case #157: [SOC-012] Successful SSH login from non-management host — 192.168.1.51 → victim-lab (as testuser1)

General | Tasks (5) | Observables (1) | TTPs (2) | Attachments | Timeline | Pages | History | Responders

Title
[SOC-012] Successful SSH login from non-management host — 192.168.1.51 → victim-lab (as testuser1)

Tags
actorunknown ai-confidence:high asset:medium assignee:l1 chain:initial-access dedupe:custom_soc:SOC-012[192.16... disposition:low-priority-queue env:lab logsource:custom-soc mitre:T1021.004 mitre:TA0008 mlaction:queue-low mlconfidence:0.77 mlconfidence:medium mlmodel:randomforest-v2 mlverdict:fp pipeline:v2 project:pfe provenance:auto-created rule:SOC-012 tlp:amber-strict

Description
QUEUED — low priority · ML verdict FP · P(TP)=23.5% · 0 overrides

What happened
Detection **SOC-012** fired: Successful SSH login from non-management host. Source 192.168.1.51 → victim-lab (user testuser1). 1 event(s) in the detection window. ML verdict: FP (P(TP)=23.5%).

Facts

Field	Value
Verdict (ML)	FP (confidence 76.5%)
P(true positive)	23.5%
Source IP	192.168.1.51
Destination	victim-lab
User	testuser1
Host	victim-lab
Rule	SOC-012 — Successful SSH login from non-management host
MITRE	T1021.004 SSH / TA0008 Lateral Movement
Severity	high (risk_score=73)
Alerts in window	1
Auto-block recommended	false

Comments

Type a comment... Send

Figure 4.9: Case #157 created in TheHive from the authorized login.

4.2.2 Anatomy of an automated case

Case #135, the SSH brute force of Section 4.2.1, is used to show how the pipeline assembles a complete case. Every tab and field below is populated by the workflows rather than by the analyst,

whose only contributions are the verdict and the confirmations.

Case details and disposition: WF1’s Build disposition node opens the case and sets its deterministic fields: the title, built from the rule, the source, the destination and the account; the severity, inherited as critical from SOC-002; the TLP, set to AMBER+STRICT; the PAP, derived from the alert; the assignee; and the namespaced tag set that records the provenance, the rule, a generated description of the case, the MITRE mapping and the model verdict. The triage result is recorded on the case as well: for #135 the model returned a false positive at a confidence of 0.879 with its recommended action, which the safety floor overrode by routing the critical, auto-block-flagged case to Tier 1 rather than auto-closing it.

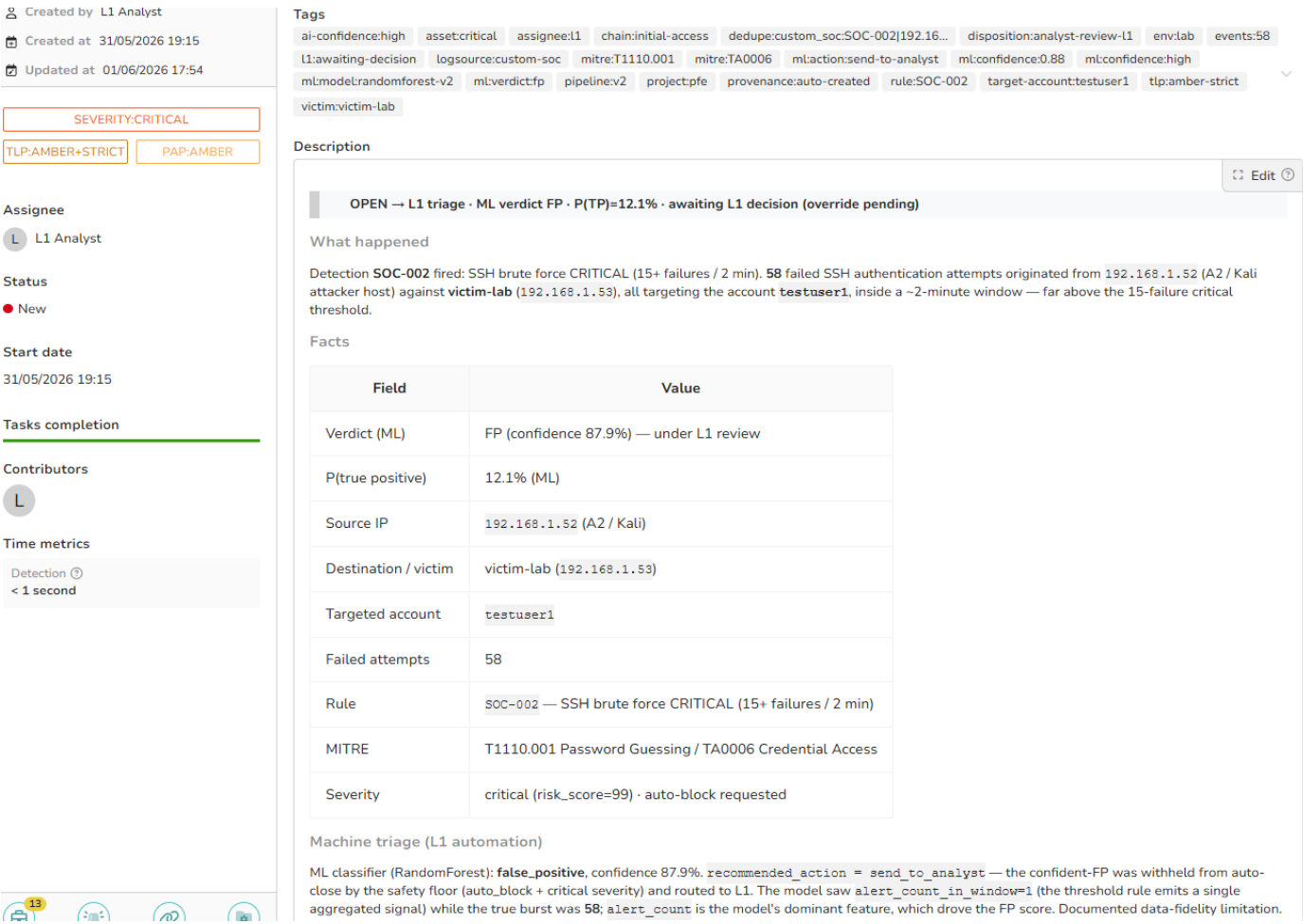


Figure 4.10: Case #135 details

Tasks: The case carries the task list. The Class-1 automation tasks (ingestion, enrichment, classification, disposition) are present and completed by the automation account; since the case is not yet escalated, the Class-2 human tasks were not created.

Chapter 4. Evaluation

	Status	Task	Order	Due date	Details	Assignee	Dates	S	C	U	
☐	Completed Date does not exist	automation	Ingestion		Activity	0	L	C. 31/05/2026 19:15			...
☐	Completed Date does not exist	automation	Enrichment		Activity	0	L	C. 31/05/2026 19:15			...
☐	Completed Date does not exist	automation	Correlation		Activity	0	L	C. 31/05/2026 19:15			...
☐	Completed Date does not exist	automation	Classification		Activity	0	L	C. 31/05/2026 19:15			...
☐	Completed Date does not exist	automation	Disposition		Activity	0	L	C. 31/05/2026 19:15			...

Figure 4.11: Case task list

Observables: The indicators extracted from the alert, including the attacker source address, are attached to the case as observables and annotated against MISP, so the evidence supporting the verdict stays on the case. Since the kali machine has an internal address, the Cortex analysers won't show their full reports, so the address 192.168.1.52 was added as an indicator of compromise in MISP, where the lookup returns a threat-intelligence report that is surfaced back on the case.

Flags	Data type	Value/Filename	Dates	S	C	U	
☐	other	testuser1	S. 01/06/2026 17:43				...
☐	ip	192.168.1.153	S. 01/06/2026 17:39				...
☐	ip	192.168.1.152	S. 31/05/2026 19:16				...

Figure 4.12: IoC observables

Analysis report

Show raw result

SOC-LAB MISP - 1 results

2310 - PFE LAB-SEEDED — validation: malicious attacker IP (Kali)

Figure 4.13: MISP report

TTPs: The MITRE ATT&CK procedures attached by WF1 from the alert's `threat[]` populate

the TTPs tab; for #135 this is the password-guessing technique behind the brute force.

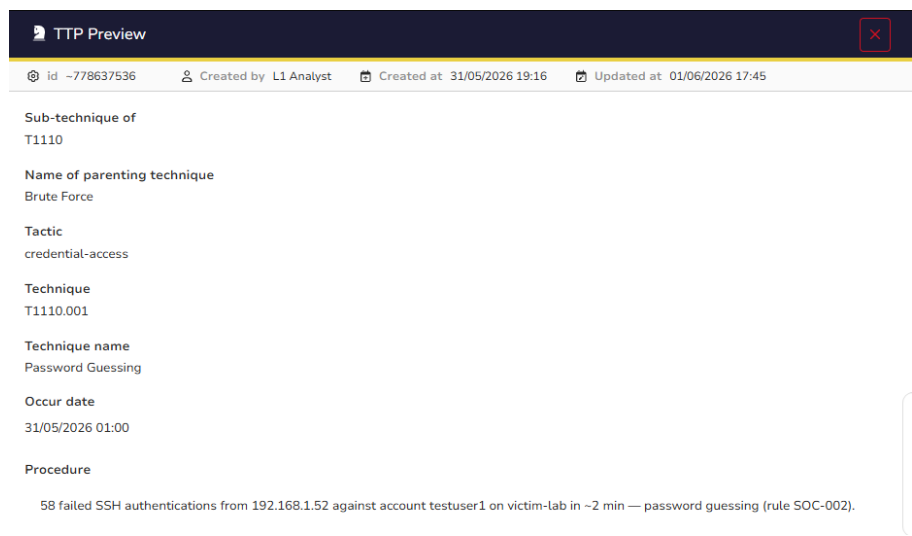


Figure 4.14: TTPs tab

Pages: 3 wiki-style pages are written by the LLM: an investigation notes page, a playbook page for the specific rule fired and the Threat Intel page produced by WF2 from the Cortex synthesis.

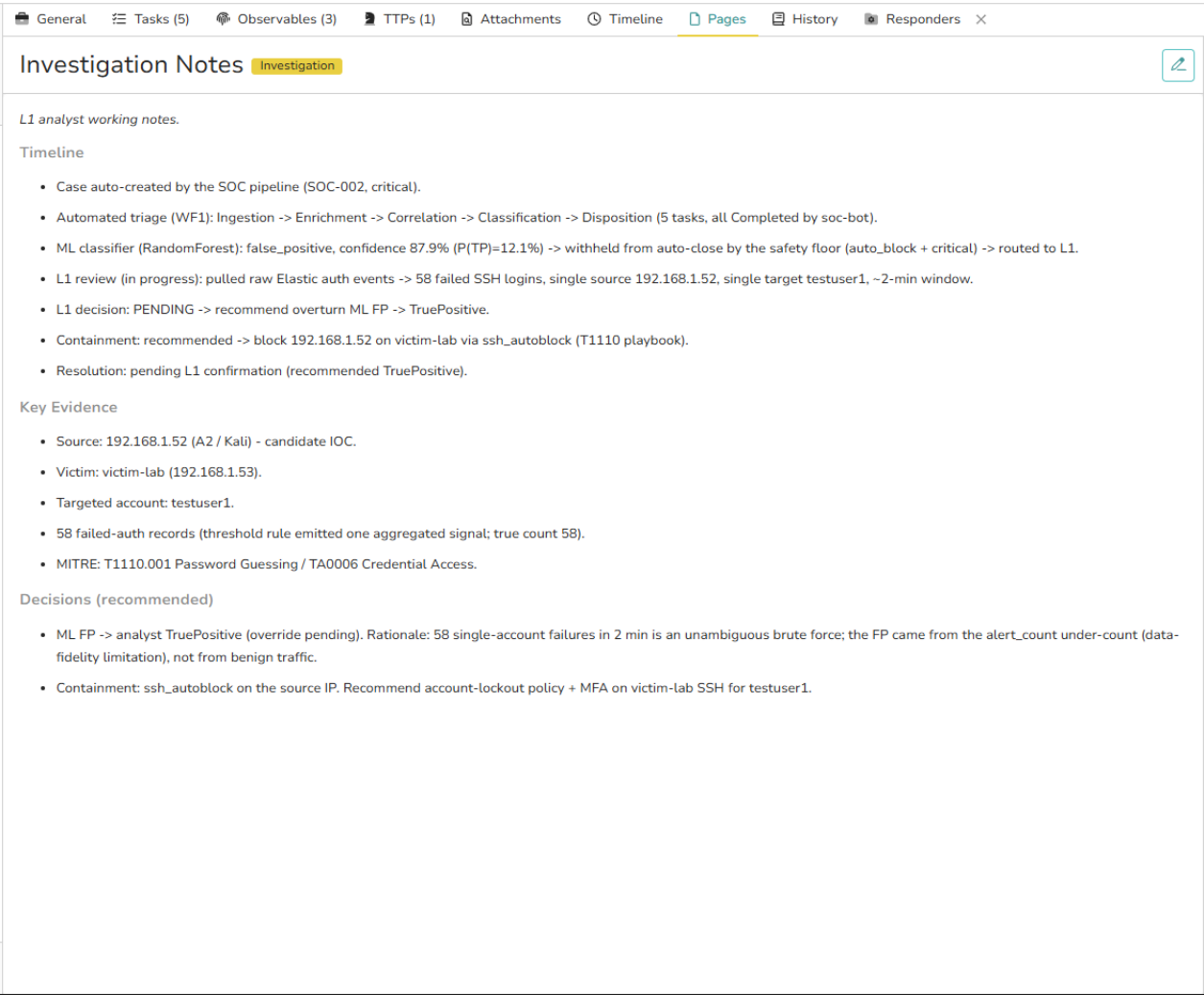


Figure 4.15: Investigation Notes page

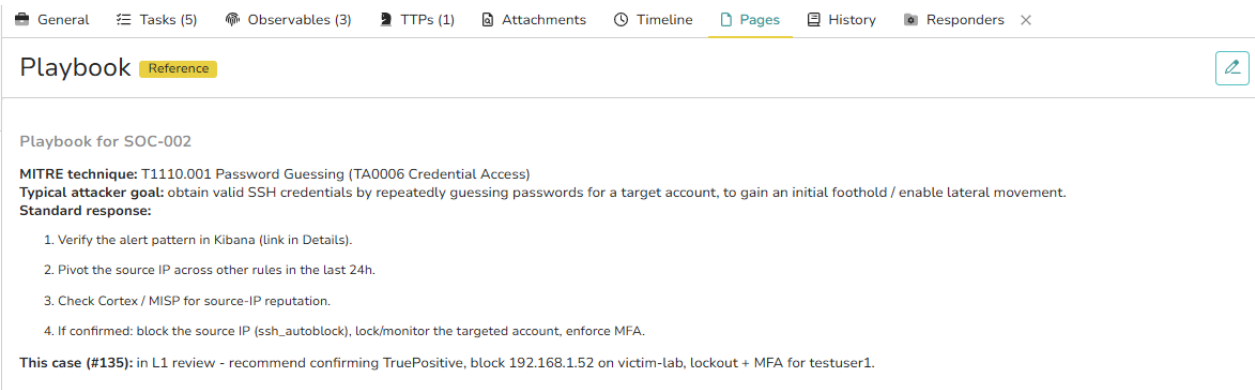


Figure 4.16: Playbook page

General Tasks (5) Observables (3) TTPs (1) Attachments Timeline **Pages** History Responders ×

Threat Intel Report Reference

Threat Intel Report
Cortex analyzers (MISP_2_1, DShield) on case observables + verified MISP lookup.

AI Threat Intel Synthesis

Source 192.168.1.52 (lab attacker, A2 / Kali) is a **known-malicious** indicator in MISP — event 2310, type `ip-src` (Network activity), `to_ids=True`. It conducted an SSH brute force (T1110.001) against victim-lab (192.168.1.53), targeting `testuser1` (58 failed auths in ~2 min). The victim IP is clean across analyzers — it is the target, not an indicator. Assessment: confirmed malicious source with prior MISP attribution; block 192.168.1.52 and enforce account lockout + MFA on the target.

Per-observable analyzer reports

ip: 192.168.1.52 — MALICIOUS

- **MISP_2_1:** MATCH — event 2310, type `ip-src`, category `Network activity`, `to_ids=True`. Known malicious source IP.
- **DShield:** no external data
- **Verdict:** malicious (MISP-attributed attacker).

ip: 192.168.1.53

- **MISP_2_1:** no match (0 events) — not a threat indicator (this is the victim).
- **DShield:** no external data (RFC1918 private address).

other: `testuser1`

- Account targeted by the brute force — no IP/host analyzer applies.

Assessment

192.168.1.52 is the malicious IOC (MISP event 2310); 192.168.1.53 is the clean victim. Classification corroborated by MISP attribution on the source IP, plus the SOC detection rule, the ML verdict, and analyst review.

Figure 4.17: Threat Intel pages

4.2.3 Response and ticketing

When a case is not auto-closed, the analyst is reached by email and acts on the case through the Cortex responders, which are thin shims that post the chosen decision or action to the orchestrator (Section 3.6.3). Case #135 exercises the full path, from the first notification to containment. **Alert email and acknowledgement:** Because #135 was routed to Tier 1 rather than auto-closed, WF1 sent the alert email shown in Figures 4.18–4.20. Sent from the SOC bot address `soc.bot.pfe@gmail.com` under the subject “[CRITICAL] SOC-002 — SSH brute force CRITICAL,” the email is the ticket that carries everything the analyst needs to act. It opens with the triage verdict, a false positive at a true-positive probability of 12.1%, with the model’s confidence in that verdict at 87.9% and a risk of 99/100, followed by a context paragraph written by the language model, which notes that the critical SOC-002 event was detected from 192.168.1.52.

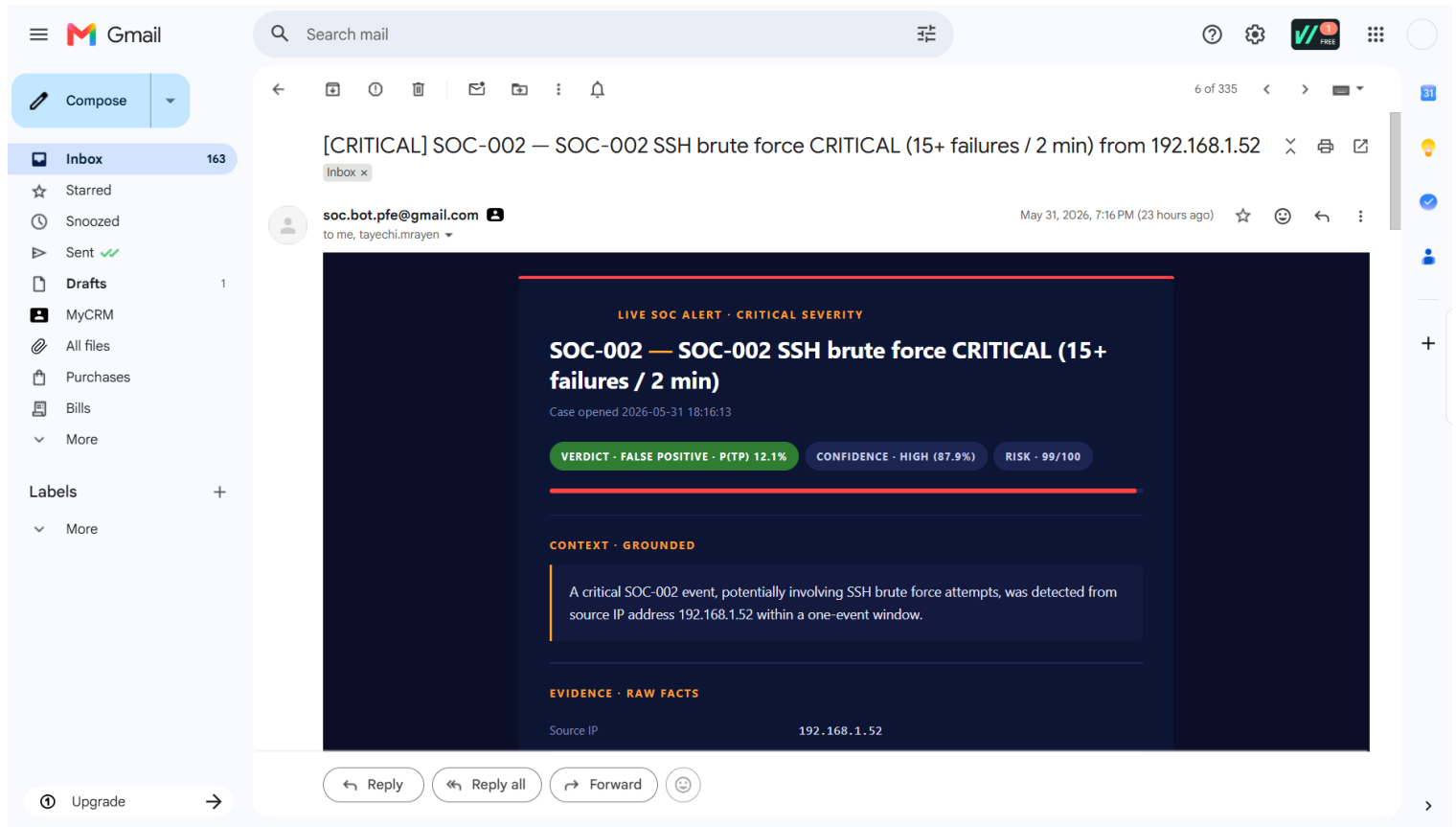


Figure 4.18: The Tier-1 alert email

The email then sets out the raw facts (as shown in Figure 4.19), the source and destination addresses, the victim host, the MITRE technique (T1110.001 Password Guessing, tactic TA0006), the 58 failed attempts, the TheHive case number and a link to the Kibana results, and places the alert on the kill chain at the Credential Access stage.

It closes with the recommended immediate and short-term actions (as shown in Figure 4.20), a false-positive checklist, and a containment note stating that no automated containment was performed because an auto-block requires an explicit Tier-1 click. The analyst acknowledges the alert with the “I’m on this case” control, which auto-tags the case in TheHive, and can open the alert directly in Kibana or the case in TheHive from the same email.

EVIDENCE · RAW FACTS

Source IP	192.168.1.52
Destination IP	192.168.1.53
Victim host	192.168.1.53 →
MITRE ATT&CK	T1110.001 Password Guessing / TA0006 Credential Access
Failed Attempts	58
TheHive case	#135
Kibana results	open →

KILL CHAIN POSITION

Detected stage: **Credential Access**

Reconnaissance
Initial Access
Execution
Persistence
Priv. Esc.
Defense Evasion
Credential Access
Discovery
Lateral Movement
Collection
C2
Exfiltration
Impact

IMMEDIATE ACTIONS · NEXT 5 MIN

- 1 Confirm the alert pattern in Kibana (link below).
- 2 Pivot on the source IP and the user across the last 24h.

Figure 4.19: The alert email (evidence and facts)

SHORT-TERM ACTIONS · NEXT 30 MIN

- 1 Cross-check the source against MISP attribute tags (misp:known | misp:new).
- 2 Review the attached IoCs (source IP, destination, victim host, file hash, URL).
- 3 If the rule is a recurring false positive, tune or suppress it.

FALSE-POSITIVE CHECK

- > Is the source IP a known internal scanner / pentester?
- > Is the activity inside a planned maintenance window?
- > Did the same rule fire many times from this IP in the last hour (likely scanner)?

CONTAINMENT NOTES

✓ No automated containment performed — autoblock requires an L1 click (Phase 20).

AUTO-CONTAINED

✓ I'm on this case

Click to acknowledge · auto-tags the case in TheHive

Open in Kibana

Open TheHive case →

Figure 4.20: The alert email (recommended actions and links)

Clicking the "I'm on this case" button records the acknowledgement. The analyst is shown the confirmation in Figure 4.21: the SOC is notified that the case is being handled, the case reference and timestamp are displayed, and the entry is signed by the bot. The acknowledgement is written back to TheHive, where the case receives an `email-acked` tag (as shown in Figure 4.22) and a corresponding entry on the case timeline, so the ownership and timing of the first human touch are on the record.

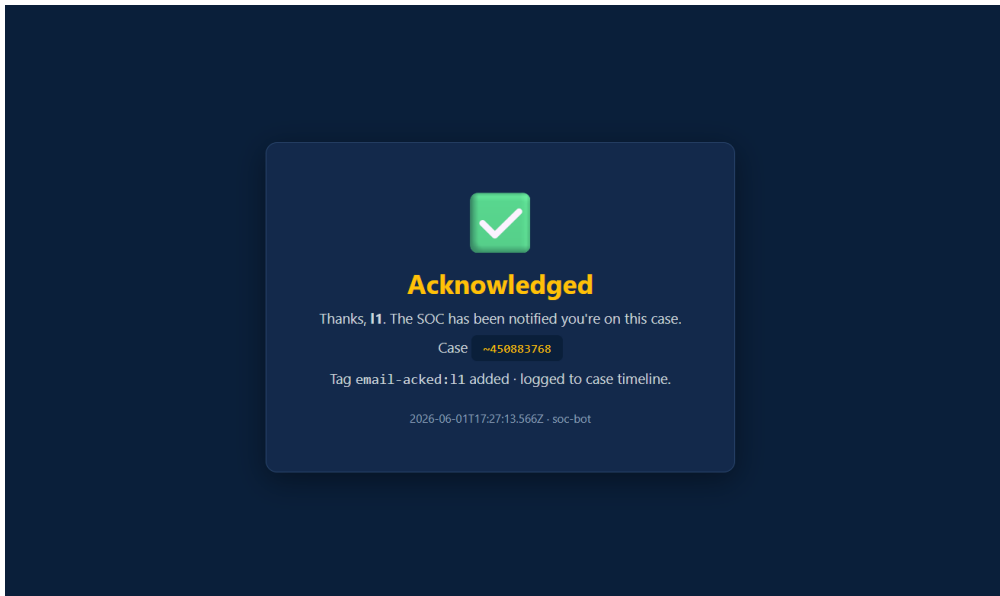


Figure 4.21: Acknowledgement confirmation

`email-acked:kchaou.habib67@gmail.com`

Figure 4.22: `email-acked` tag written to the case on acknowledgement.

Responder-driven decision: The analyst acts on the case through the Cortex responder menu (as shown in Figure 4.23), which exposes the eight responders of Section 3.6.3. Because the triage model had returned a false positive on the brute force (Section 4.2.2), the analyst selected three responders and launched them together: `SOC_Confirm_TP` to overturn the verdict, `SOC_Escalate_L2` to escalate the case, and `SOC_Autoblock_Isolate` to contain the source. Each responder posts to WF9 (the verdict and the escalation) or WF10 (the containment) and writes an audit comment back to the case (as shown in Figure 4.24): the verdict is recorded as overturned from the model's false positive to `TruePositive`, the escalation as a hand-off to the Tier-2 owner for a critical credential-access incident, and the containment as an `ssh_autoblock` on `192.168.1.52` via WF10 with the status `initiated`. Each comment is sourced to `cortex_responder`, which keeps the machine-and-human boundary explicit, and the case is tagged with the corresponding `responder:` tags (as shown in

Figure 4.25).

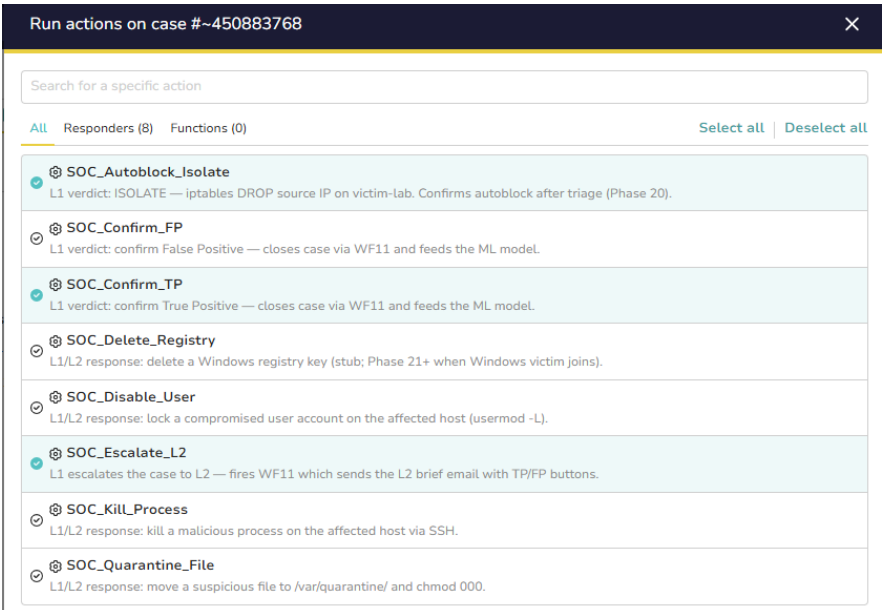


Figure 4.23: The Cortex responder menu

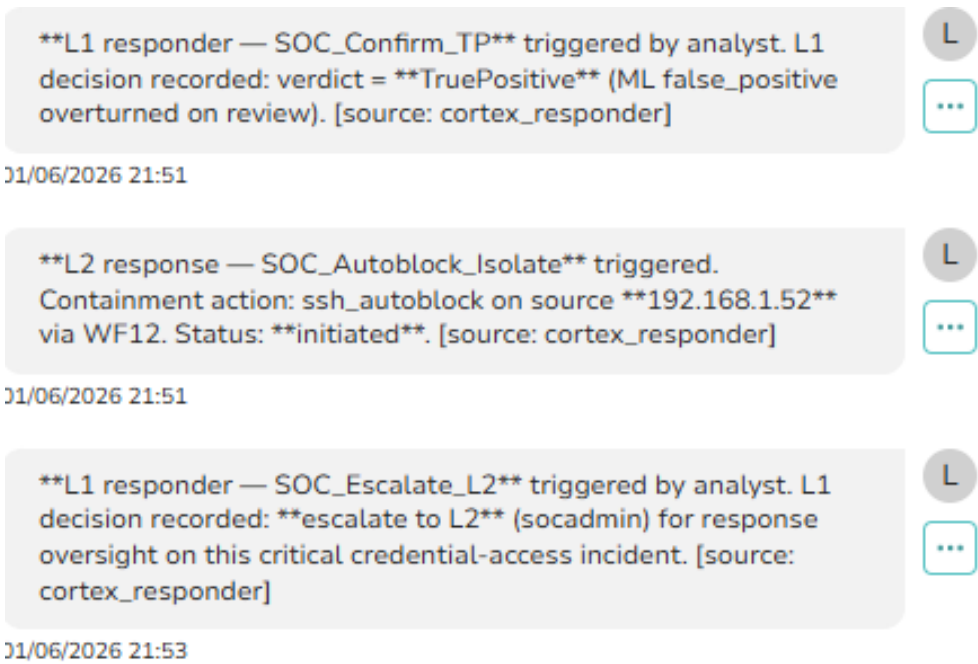


Figure 4.24: The audit comments



Figure 4.25: The responder:* tags added to the case.

Escalation email. The SOC_Escalate_L2 responder fires WF9, which generates the Tier-2 brief and sends it by email from soc.bot.pfe@gmail.com. The brief (as shown in Figures 4.26–4.27)

opens with the initial alert summary and the asset, then sets out the Tier-2 investigation: the MITRE mapping, a process analysis of the 58 failed attempts, a threat-intelligence lookup on the source, and the SIEM and firewall observations. It lists the remediation actions and their status, the recommended next steps, and the raw facts, and ends with one-click *Confirm True Positive* and *Confirm False Positive* controls that close the case in TheHive and feed the result back to the model.

1. INITIAL ALERT SUMMARY (L1 ESCALATION)

Trigger Time	2026-06-01 22:14 UTC
Asset	lab-victim-host (User: root)
Description	TheHive case #135 identified a critical SSH brute force attack against victim-lab originating from IP address 192.168.1.52, resulting in 15+ failed login attempts within a 2-minute timeframe using the 'testuser1' account.

2. L2 INVESTIGATION DETAILS

MITRE ATT&CK

TA0006 • T1110.001 (Brute Force)

Reconnaissance Initial Access Execution Persistence Priv Esc Defense Evasion Credential Access Discovery Lateral Movement Collection C2 Exfiltration Impact

Process Analysis

Analysis revealed 58 failed SSH login attempts from 192.168.1.52 targeting the victim-lab host as the 'testuser1' account. The attacker utilized a brute-force methodology, repeatedly attempting credentials. The attack occurred between 14:35 and 14:37 UTC.

Threat Intel Lookup

Threat intelligence suggests 192.168.1.52 is associated with known malicious activity, including previous attempts at credential stuffing and vulnerability exploitation. AbusePDB flags this IP as exhibiting suspicious behavior, specifically related to brute-force attacks and potential botnet activity. VT reports similar IPs associated with credential theft campaigns.

Network Logs (SIEM / Firewall)

SIEM logs show a high volume of SSH traffic originating from 192.168.1.52 to victim-lab port 22. Firewall logs indicate blocked connections from this source IP. No outbound traffic was observed from victim-lab during the event.

3. ACTIONS TAKEN (REMEDIATION)

- ✓ **Network Isolation.** 192.168.1.52 was immediately isolated from the network to prevent further access.
COMPLETED
- ⚠ **Account Action.** The 'testuser1' account on victim-lab was immediately disabled and password reset.
PENDING ANALYST
- ✓ **Malware Deletion or Process Termination.** No malware was detected on victim-lab, and all processes associated with the SSH connection were terminated.
COMPLETED

Figure 4.26: The Tier-2 brief email (top)

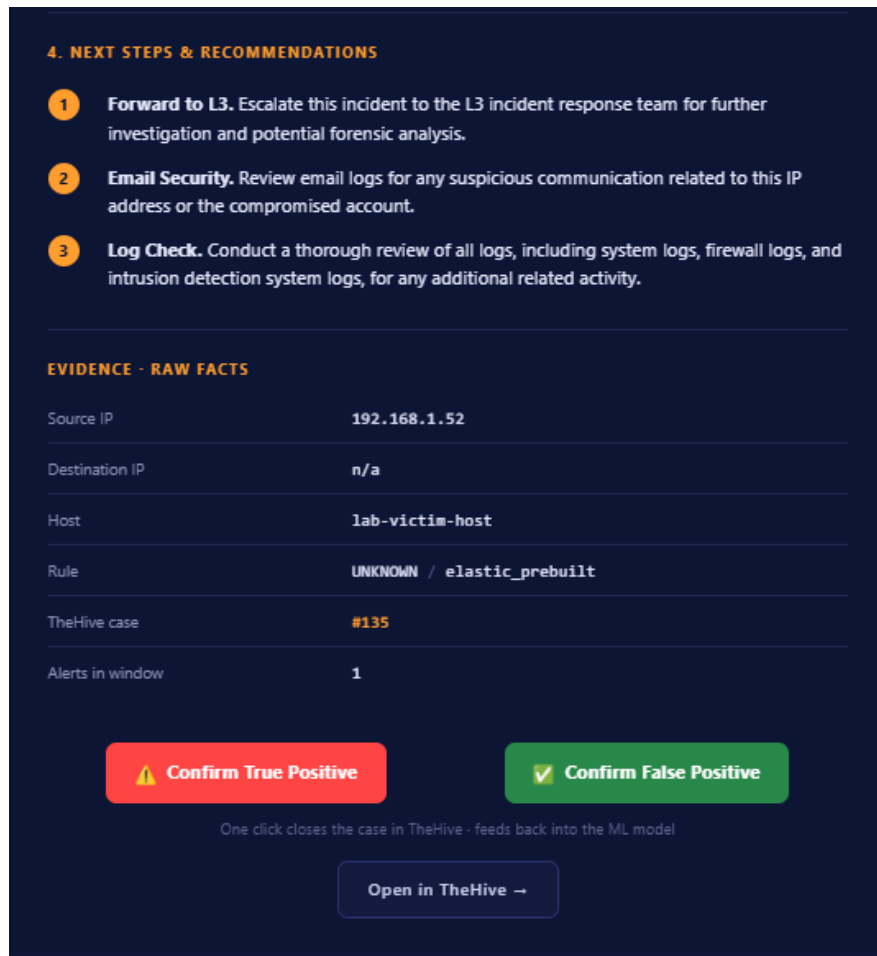


Figure 4.27: The Tier-2 brief email (bottom)

Confirmed containment. The containment launched with the verdict applies the `ssh_autoblock` recorded above against 192.168.1.52 on the victim host through WF10. The action is gated behind the explicit responder launch rather than fired automatically, and the response runner carries a guard that refuses to firewall the management plane (its `ssh_autoblock` script is reproduced in Figure C.5). On the victim host the resulting rule sits at the top of the INPUT chain (as shown in Figure 4.28): a DROP target for all traffic from 192.168.1.52, so any subsequent connection from the attacker is discarded at the host.

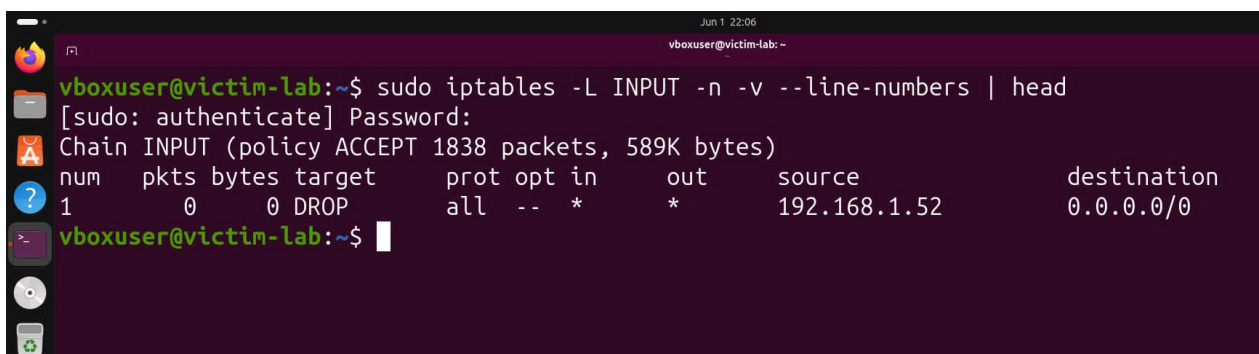


Figure 4.28: iptables DROP rule on the victim host

Resolution. With the verdict confirmed and the source contained, the case is resolved as **TruePositive** under the Tier-2 owner (as shown in Figure 4.29). The case carries its own time metrics: a detection latency under one second, while the triage and acknowledgement times reflect the analyst’s availability in the laboratory rather than any system latency.

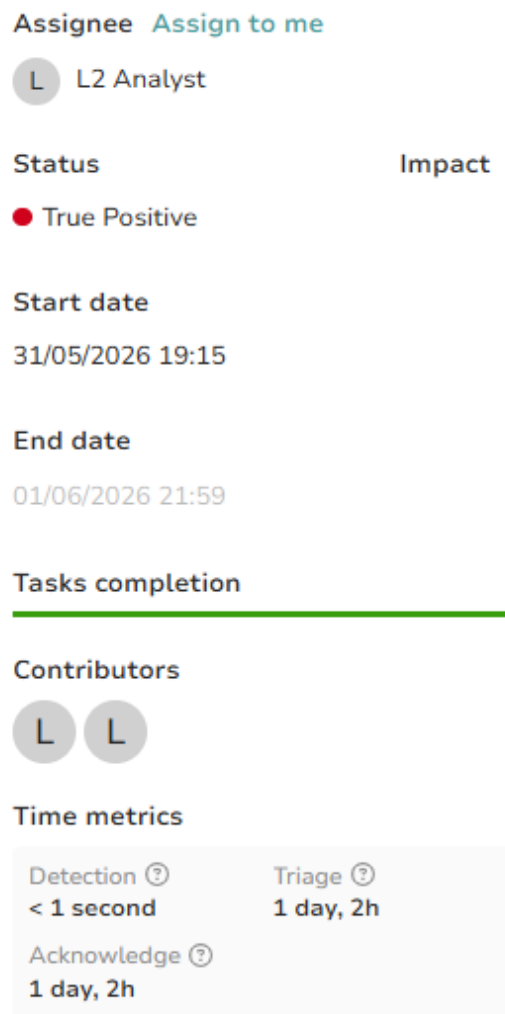


Figure 4.29: The resolved case

Investigation report. On resolution, WF5 generates the investigation report and posts it as a case page (as shown in Figure 4.30). The report is structured into an executive summary, a timeline, the MITRE mapping with its mitigations, the threat intelligence, the response and resolution, and the model feedback, and it records that the resolved label is returned to `ml-api /feedback` for the continuous-learning loop.

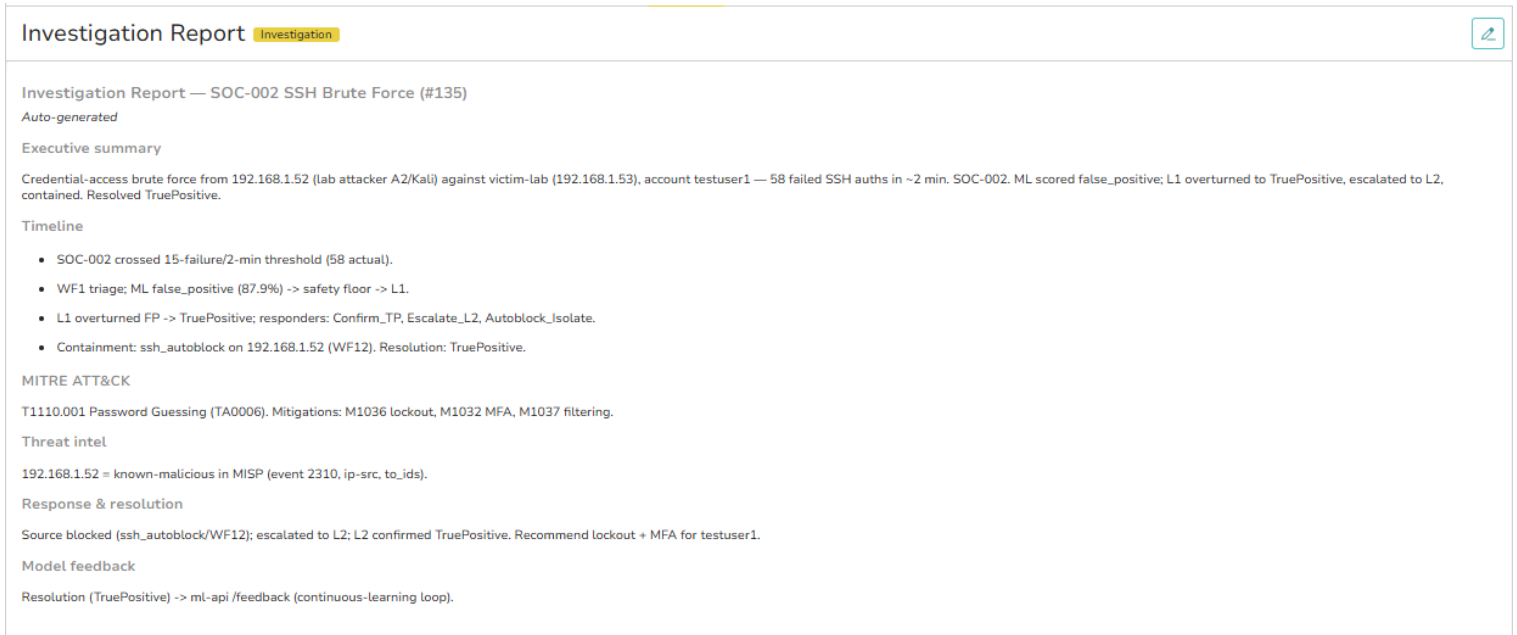


Figure 4.30: Investigation report

4.3 Autonomous capabilities and AI model performance

4.3.1 Threat intelligence to detection rule

Beyond reacting to alerts, the platform turns threat intelligence into new detection logic through WF3 (as shown in Figure 4.31). When an analyst publishes an event in MISP, the webhook hands it to a chain that extracts the indicators, prompts the language model on the inference host to generate a rule, validates and assembles the output as NDJSON, and then branches on whether the generation is valid: a valid rule is imported into Kibana and logged in TheHive, while an invalid or failed generation is routed to a failure log and imported nowhere.

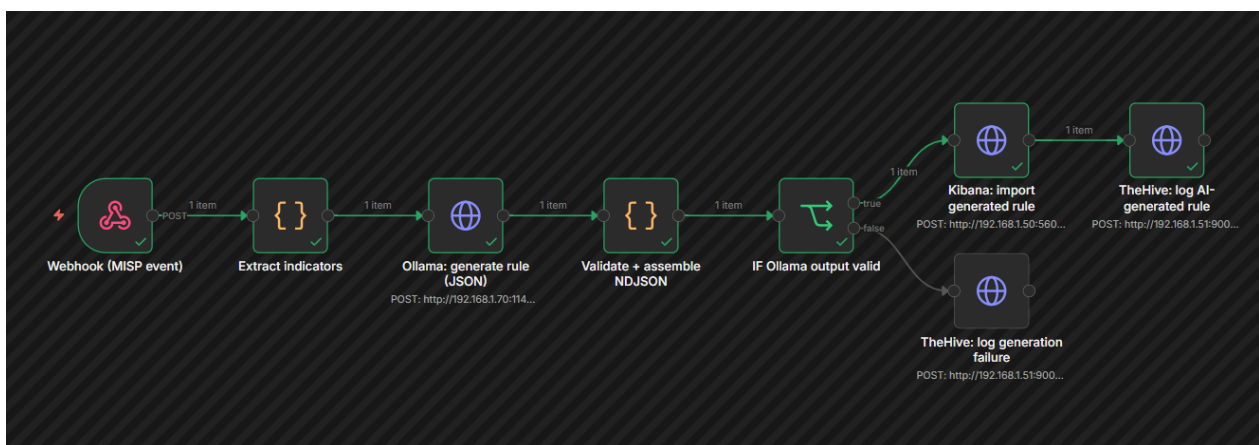


Figure 4.31: The WF3 pipeline

Rule generation: Publishing a MISP IP indicator produced a valid rule, “[AUTO] Malicious IP Detection” (as shown in Figure 4.32). The rule carries a KQL query (`logs AND (source.ip="198.51.100.77")`), a critical severity, a risk score of 95, a MITRE mapping (TA0005 Defense Evasion, T1074 — an off-tactic pairing that itself illustrates the model’s generation drift, Section 4.4.3), and the provenance tag `provenance:misp-ai-generated`. It is imported disabled by default, so a human reviews and enables it before it can fire, the same human-in-the-loop principle that governs the response side. The deployment is logged in TheHive as an alert from the source `n8n-WF3` (as shown in Figure 4.33).

The screenshot displays the TheHive web interface for a rule named "[AUTO] Malicious IP Detection". At the top, there's a search bar with the text "Filter your data using KQL syntax" and buttons for "Today" and "Refresh". Below the title, it shows the rule's creation and update timestamps, a status toggle set to "Disable", and an "Edit rule settings" button. A notification bell icon indicates "Last response" and "Notify when alerts generated".

The main content is divided into two columns:

- About:**
 - Description:** Detects connections from the known malicious IP address 198.51.100.77.
 - Severity:** Critical (indicated by a red dot).
 - Risk score:** 95.
 - MITRE ATT&CK™:** Defense Evasion (TA0005) and Data Staged (T1074).
 - Max alerts per run:** 100.
 - Tags:** project:pfe, env:lab, provenance:misp-ai-generated, mitre:TA0005, mitre:T1074, malware, ip, threat_detection, misp.
- Definition:**
 - Index patterns:** logs-*, filebeat-*, .alerts-security.alerts-default.
 - Custom query:** logs-* AND (source.ip="198.51.100.77").
 - Custom query language:** KQL.
 - Rule type:** Query.
 - Timeline template:** None.

A "Schedule" section is visible at the bottom right.

Figure 4.32: The rule generated by WF3

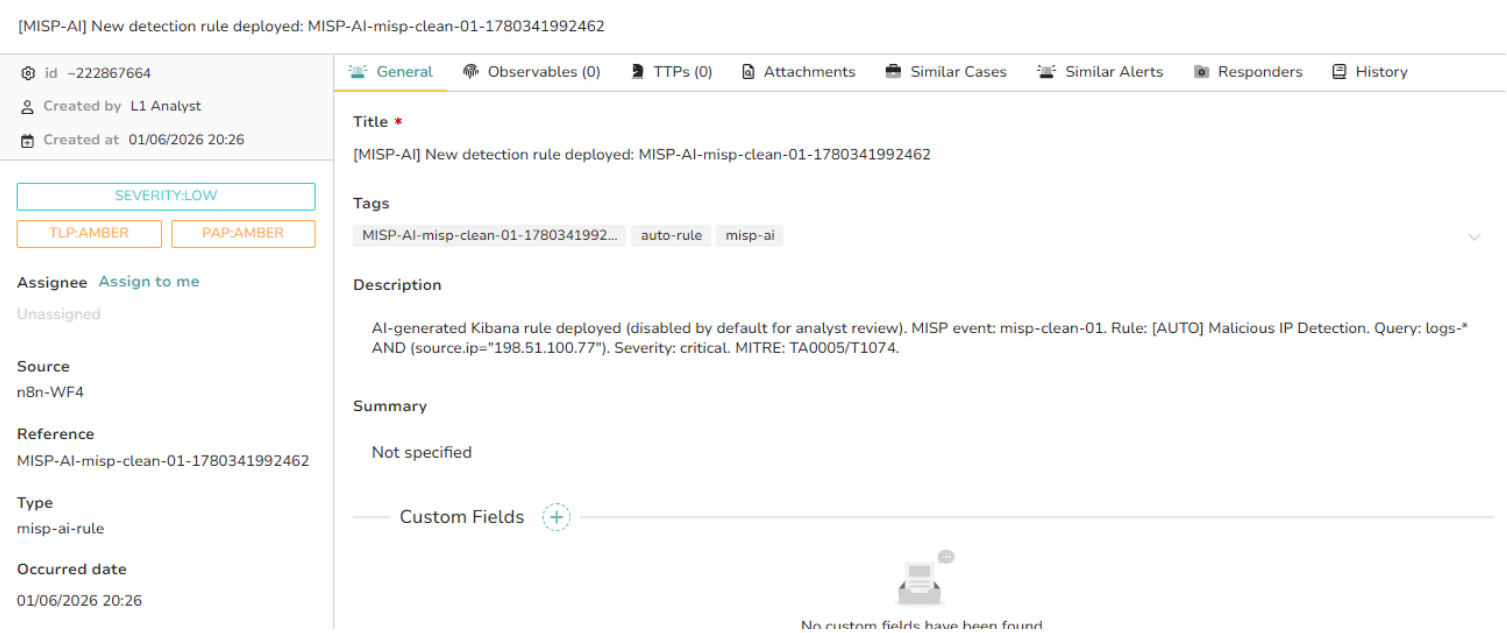


Figure 4.33: TheHive alert logging the AI-generated rule

4.3.2 Self-maintenance and continuous learning

The platform maintains and improves itself on a schedule. WF4 runs every Monday at 04:00; it retrains the triage model on the last thirty days of labels resolved by analysts in TheHive, reloads the models, checks the health of Elasticsearch and `ml-api`, counts the recent alert volume, and files the result as a maintenance case for audit (as shown in Figure 4.34). A manual run produced the case shown: the thirty-day retrain completed and advanced the model’s `retrained_at` timestamp to 2026-06-01 19:55, reloading both the Random Forest and the XGBoost models, and the alert volume over the previous seven days was 1,215. The retrain metrics recorded on the case are the same figures reported in Section 4.3.3.

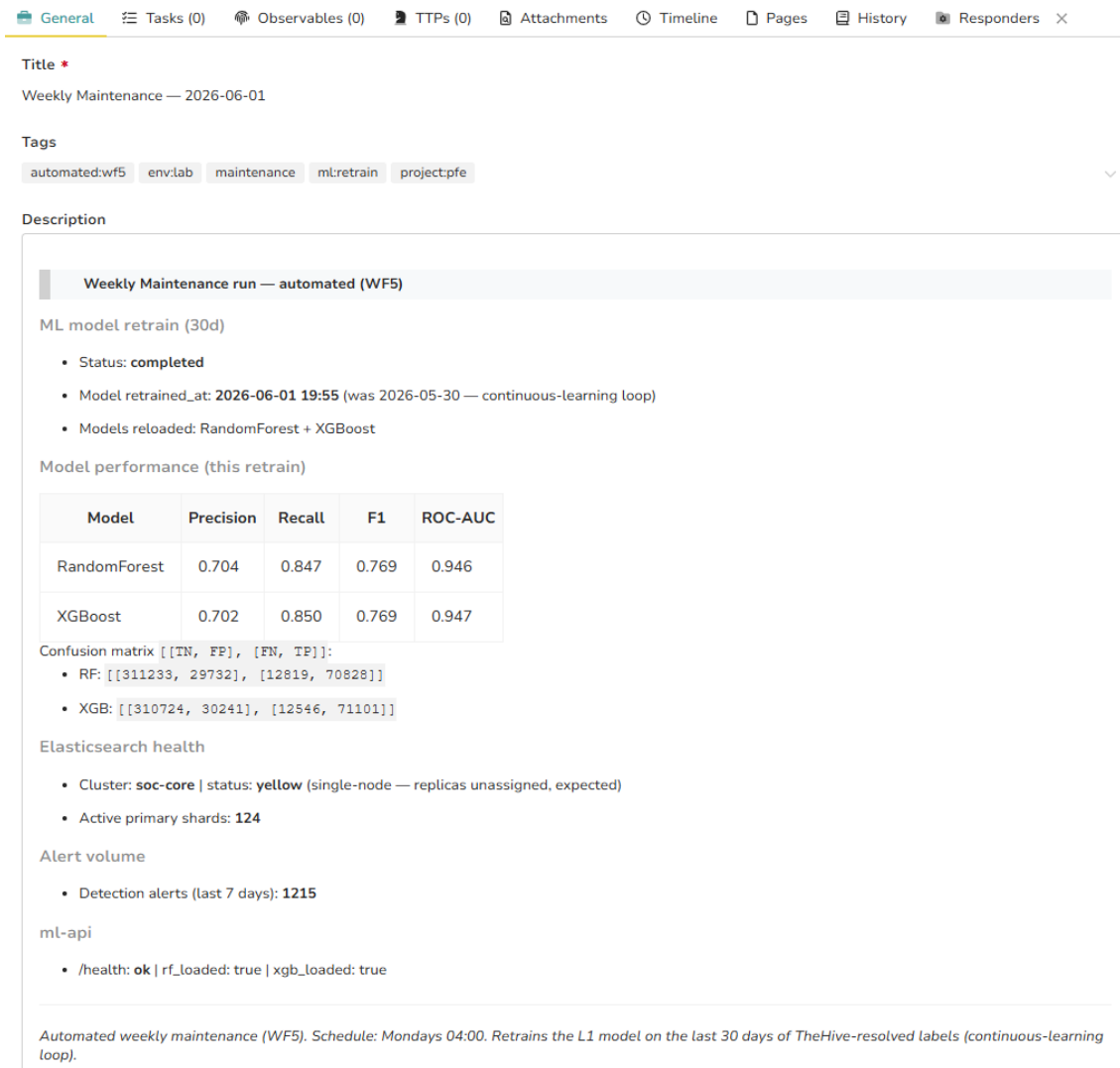


Figure 4.34: The weekly-maintenance case filed by WF4

4.3.3 ML performance

Two evaluations are reported: the benchmark obtained at initial training, and the model after the thirty-day continuous-learning retrain of Section 4.3.2. The two algorithms perform almost identically. At initial training both reach a precision of 0.906 and an F1 around 0.875; after the retrain the operating point shifts toward recall, with precision falling and recall holding, while the area under the ROC curve stays at roughly 0.946 throughout. Table 4.2 reports the figures.

Table 4.2: Triage model performance

Model	Evaluation	Precision	Recall	F1	ROC-AUC
Random Forest	Initial training	0.906	0.846	0.875	0.945
XGBoost	Initial training	0.906	0.847	0.876	0.946
Random Forest	After 30-day retrain	0.704	0.847	0.769	0.946
XGBoost	After 30-day retrain	0.702	0.850	0.769	0.947

The ROC curves of the two models are almost coincident (as shown in Figure 4.35), with areas under the curve of 0.946 and 0.947, confirming that they discriminate benign from malicious alerts equally well. The confusion matrices for the initial benchmark are shown in Figure 4.36, over the held-out test set of 90,000 samples. The Random Forest correctly identified 41,051 benign and 38,083 attack alerts, against 3,949 benign alerts misclassified as attacks and 6,917 attacks misclassified as benign; XGBoost is within a few dozen samples on every cell. The false-negative cell, genuine attacks scored as benign, is the failure mode that the case #135 miss falls into (Section 4.4.3). The matrices recorded after the retrain are carried on the maintenance case of Figure 4.34. The Random Forest is retained as the default because its feature importances are directly interpretable (Section 4.3.4).

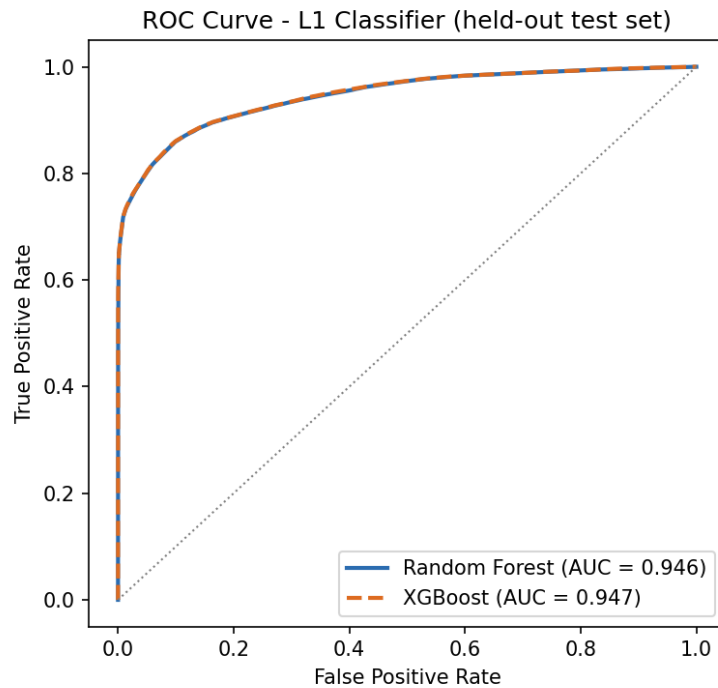


Figure 4.35: ROC curves for the Random Forest and XGBoost triage models on the held-out test set.

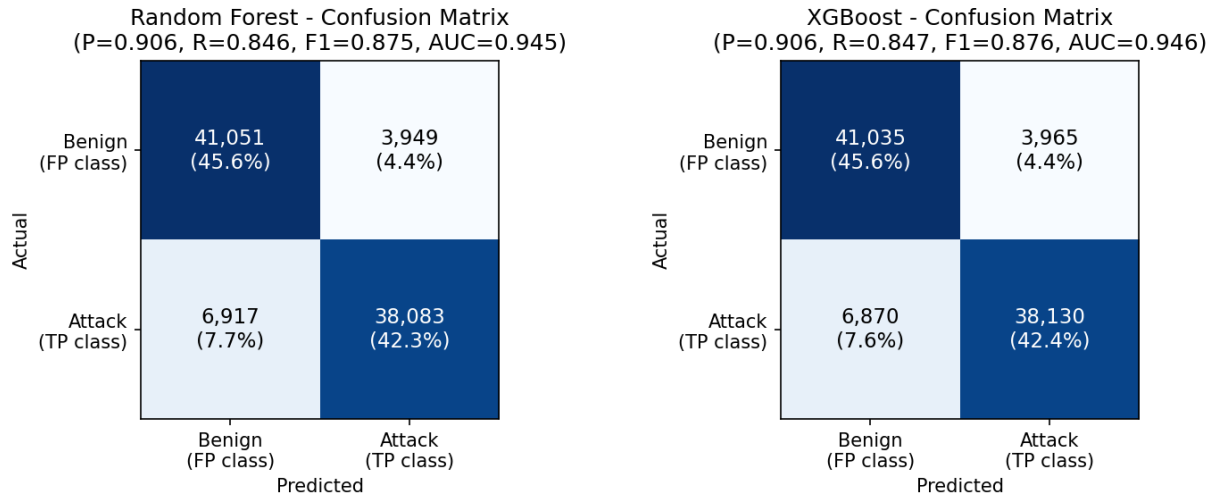


Figure 4.36: Confusion matrices: Random Forest (left) and XGBoost (right).

4.3.4 Feature importance

The Random Forest importances are shown in Figure 4.37. The decision is dominated by a single feature, the count of alerts in the correlation window, which carries seventy per cent of the model's weight; the internal-source flag, the risk score and the ordinal severity account for most of the remainder, and the temporal features are negligible.

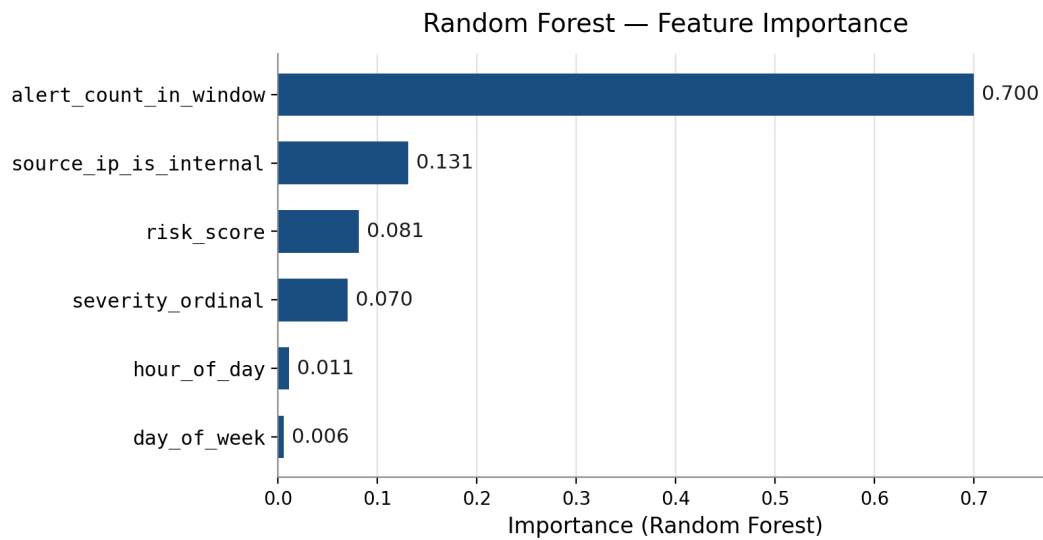


Figure 4.37: Random Forest feature importances

4.4 Discussion

4.4.1 Operational impact and alert reduction

The pipeline reduces analyst load at three points. The severity gate prevents low- and medium-severity events from raising a case; deduplication folds repeated alerts for the same source and rule into a single case rather than many; and the auto-close of confident, low-risk false positives removes them from the queue entirely. Over the seven days preceding the evaluation the platform recorded 1,215 detection alerts (as shown in Figure 4.34); the ratio of that volume to the cases actually raised is the reduction the system delivers, measured at approximately 89.3%.

4.4.2 Latency

The detection-to-case latency is low: case #135 recorded a detection time under one second (as shown in Figure 4.29). The triage and acknowledgement times on the same case, of the order of a day, reflect the analyst’s availability in the laboratory rather than any processing delay, since the case waited for a human to open the alert email. The only intrinsically slow stages are the language-model generations of WF3, the investigation report and the escalation brief, which run on the dedicated inference host; the classifier itself returns a verdict in well under a second.

4.4.3 Limitations

The validation surfaced limitations across the system:

The most consequential concerns the breadth of detection. The system’s detection rests on a fixed set of signature- and threshold-based rules, so it recognises only the techniques those rules describe; an attack that matches no rule produces no alert, and therefore no case, and the platform has no measure of its false-negative rate against techniques outside this set. The validation exercised a handful of representative attacks rather than a broad or evolving attack surface, so the detection claim is bounded by the rules in place.

A second limitation is the basis for the reported metrics. The CICIDS2017 features correlate strongly with the label, so the figures of Section 4.3.3 are optimistic proxies; the system’s real discrimination accrues through the resolved-label feedback loop rather than from the public dataset.

A third concerns the generated content. The language model is small (`gemma3:4b`), and its briefs and rules occasionally drift from the deterministic facts of a case: a generated rule may carry an unexpected tactic, or an escalation brief may place the kill chain at a different stage than the

alert. The design contains this rather than trusting it, generated rules are imported disabled for human review, and the briefs supplement the grounded fields rather than replacing them, but the drift limits how far the narrative content can be relied upon unsupervised.

The remaining limitations are architectural and operational. Elasticsearch runs as a single node, so the cluster sits in a yellow state by design; the laboratory has a single Linux victim, so the Windows-oriented responders remain stubs and the response runner is exercised only through `ssh_autoblock`; the retraining runs synchronously in a single `ml-api` worker and loads the whole dataset in process, which contends with Elasticsearch on the constrained host; and a transient reactive-streams race on the incident-management host occasionally returned write errors that succeeded on retry. The validation itself covered a handful of attack types in a controlled laboratory, not a production-scale, broadly varied attack surface.

4.4.4 Future improvements

Several improvements follow directly from the limitations:

Broadening the detection coverage: The rule set should be extended beyond the techniques validated here, and the threat-intelligence-to-rule path of Section 4.3.1 is the natural vehicle for this, turning new intelligence into detection logic automatically as it arrives. The false-negative rate should then be measured against a broader and evolving attack surface rather than the representative sample used in this validation, so that the coverage claim rests on evidence.

Fine-tuning the language model. The drift in the generated briefs and rules stems in part from a small, general-purpose model prompted on an unfamiliar domain; fine-tuning it on the platform's own corpus — the resolved cases, their deterministic fields, the MITRE mappings and the analyst-corrected outputs that accumulate over time — would align its language and structure with the SOC's conventions and reduce its divergence from the case facts. The disabled-by-default import and the human-review gate would still bound the result, but a domain-adapted model would need that supervision less often.

Strengthening the generated content. Tighter schema validation of the model's output, stricter grounding against the case's deterministic fields, and a larger or better-instructed model would reduce the drift, while keeping the disabled-by-default import and the human-review gate that already contain it.

Hardening the deployment: Elasticsearch should move to a multi-node cluster to leave the single-node yellow state; a Windows endpoint should be added so that the full response set, disabling

a user, killing a process, quarantining a file, deleting a registry key, can be exercised rather than stubbed.

Conclusion

This chapter presented a holistic assessment of the autonomous SOC, examining its detection, triage and response in the context of an emulated port scan and an SSH brute-force intrusion. These assessments confirmed that the platform reliably detected the attacks with its Elastic detection rules, triaged them with the supervised classifier, assembled and enriched complete cases in TheHive through Cortex and MISP, generated the briefs, investigation reports and threat-intelligence rules with the integrated gemma3:4b model, and carried each incident through to a confirmed containment via the Cortex responders and n8n. The assessment also uncovered limitations, chiefly the detection breadth bounded by the fixed rule set and the lab-scale constraints of the deployment; however, it is clear that automating the detection-to-response loop significantly improves threat-management efficiency within the SOC context and provides a strong basis for future advancement.

General Conclusion

The goal of this end-of-studies project was to improve how a Security Operations Center works by automating the two steps current practice still leaves to people: responding to a confirmed threat, and writing each incident up as a report and a ticket. Detection and case management are already well served by mature tools, but these two steps are not, so we set out to automate them with AI, driven by an automated triage engine, and to build the result as a complete, self-contained SOC that could be integrated and tested from end to end.

The stack is entirely open-source and runs at no recurring cost on the organisation's own hardware: Suricata and the Elastic Stack for detection, MISP for threat intelligence, TheHive and Cortex for case management and enrichment, and n8n for orchestration, with the two pieces the bundle lacked added on top, a supervised classifier that triages every alert and a local language model that writes the analyst's brief. The report followed the work: Chapter 1 framed the problem and the proposed solution, Chapter 2 surveyed the techniques and the tools, Chapter 3 detailed the design and implementation, and Chapter 4 evaluated the system against emulated attacks.

The evaluation confirmed that the pipeline works from end to end: each attack was detected, triaged by the classifier, assembled into an enriched case in TheHive, written up as an AI-drafted brief, report and ticket, and carried through to a human-confirmed containment, with the resolved verdict returning to the model as fresh training data. The system cut the alert volume reaching an analyst by roughly 89%, the triage model reached a precision above 0.90 at initial benchmarking and a ROC-AUC near 0.95 throughout, and the detection-to-case latency stayed under a second, all under one principle: the machine triages, but the human decides.

The work also has clear boundaries: detection is limited to the rules in place, the headline metrics rest partly on a public dataset, and the deployment was tested at laboratory scale. These set the directions for what follows: broadening the detection coverage through the threat-intelligence-to-rule path, fine-tuning the language model on the platform's own resolved cases, and hardening the deployment toward production. Being open, local, and free to run, the system is also within reach of the small and mid-sized organisations the commercial market prices out. In closing, this project shows that open-source tools and a carefully bounded use of artificial intelligence can automate a SOC's most repetitive work, the response and the reporting, while leaving every consequential decision in human hands.

Webography

- [1] Next Step. *Next Step: Presentation*. URL: <https://www.nextstep-it.com/> (visited on 05/19/2026).
- [2] Tanitjobs. *Next Step IT — Company Profile and Certifications*. URL: <https://www.tanitjobs.com/company/10715/Next-Step-IT> (visited on 05/19/2026).
- [3] Next Step. *Next Step: Services*. URL: <https://www.nextstep-it.com/solutions-services/> (visited on 05/19/2026).
- [4] IBM Security. *Cost of a Data Breach Report 2024*. 2024. URL: <https://www.ibm.com/reports/data-breach> (visited on 05/19/2026).
- [5] ISC2. *Cybersecurity Workforce Study 2024*. 2024. URL: <https://www.isc2.org/Insights/2024/10/Cybersecurity-Workforce-Study> (visited on 05/19/2026).
- [6] Cloud Security Alliance and Dropzone AI. *Beyond the Hype: A Benchmark Study of AI in the SOC*. 2025. URL: <https://cloudsecurityalliance.org/artifacts/a-benchmark-study-of-ai-agents-in-the-soc> (visited on 05/20/2026).
- [7] Blake E. Strom et al. *MITRE ATT&CK: Design and Philosophy*. Tech. rep. The MITRE Corporation, 2020. URL: https://attack.mitre.org/docs/ATTACK_Design_and_Philosophy_March_2020.pdf (visited on 05/19/2026).
- [8] Chris Johnson et al. *Guide to Cyber Threat Information Sharing*. Tech. rep. NIST Special Publication 800-150. National Institute of Standards and Technology, 2016. DOI: 10.6028/NIST.SP.800-150. URL: <https://csrc.nist.gov/pubs/sp/800/150/final> (visited on 05/19/2026).
- [9] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.
- [10] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.
- [11] Ziwei Ji et al. “Survey of Hallucination in Natural Language Generation”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38. DOI: 10.1145/3571730.

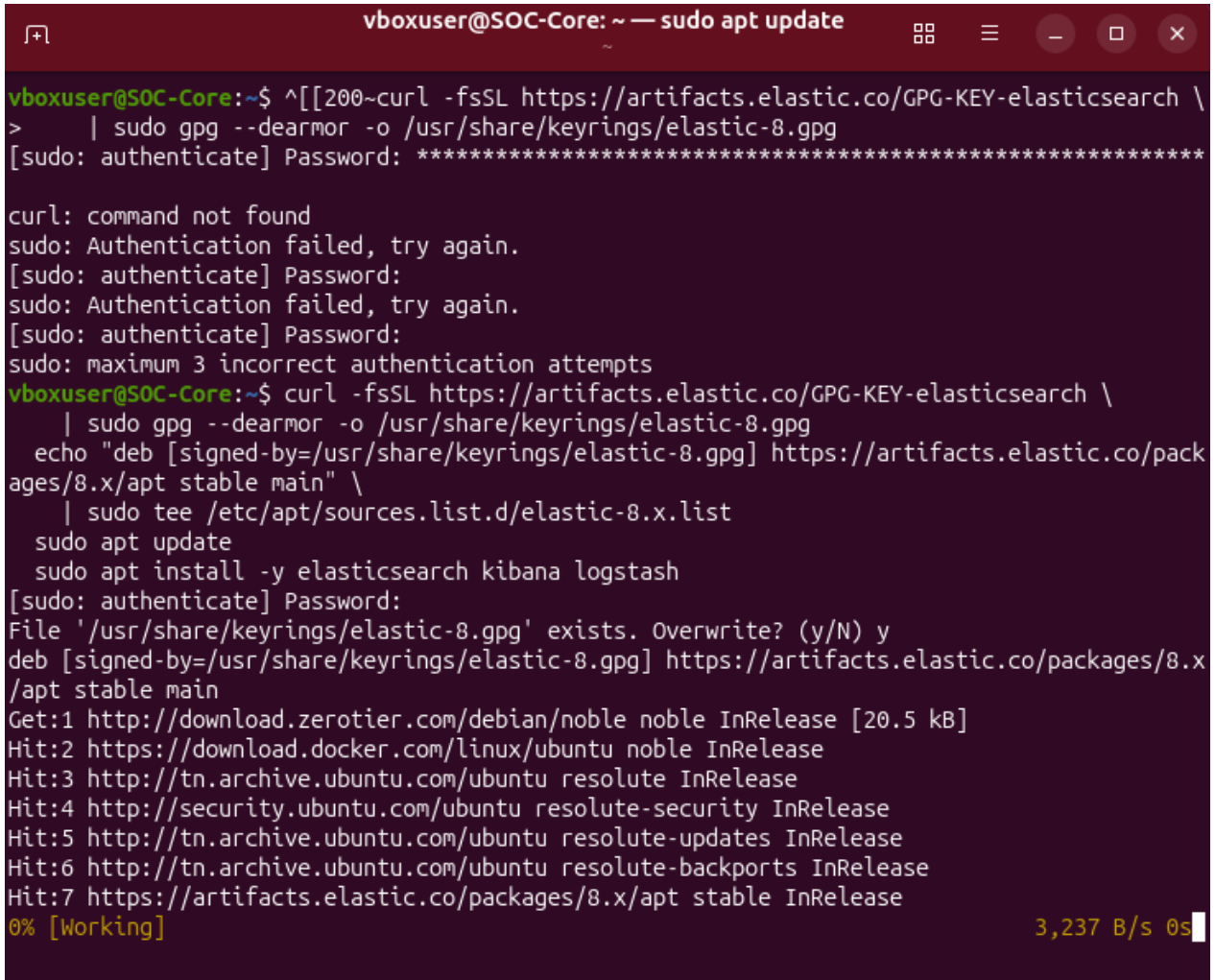
- [12] Open Information Security Foundation. *Suricata: Open Source IDS / IPS / NSM Engine*. URL: <https://suricata.io/> (visited on 05/21/2026).
- [13] Elastic. *Elastic Stack: Elasticsearch and Kibana*. URL: <https://www.elastic.co/elastic-stack> (visited on 05/21/2026).
- [14] MISP Project. *MISP: Open Source Threat Intelligence Platform*. URL: <https://www.misp-project.org/> (visited on 05/21/2026).
- [15] StrangeBee. *TheHive: Open Source Security Incident Response*. URL: <https://strangebee.com/thehive/> (visited on 05/21/2026).
- [16] TheHive Project. *Cortex: Observable Analysis and Active Response Engine*. URL: <https://github.com/TheHive-Project/Cortex> (visited on 05/21/2026).
- [17] n8n. *n8n: Workflow Automation Platform*. URL: <https://n8n.io/> (visited on 05/21/2026).

Appendix A

Platform deployment and threat-intelligence automation

This appendix documents how the platform of Chapter 3 is provisioned across its two operational hosts and kept fed with threat intelligence. The first group of figures covers the SOC-Core host — the SIEM, the Fleet-managed agent and the orchestrator; the second covers the incident-management host — the case-management and threat-intelligence stack — together with the cron job that refreshes the MISP feeds. The platform’s running behaviour is shown by the functional tests of the evaluation chapter. The third-party packages are integrated rather than authored; the `cron-feeds.sh` script is part of the project’s own glue.

A.1 SOC-Core host: SIEM, Fleet and orchestration

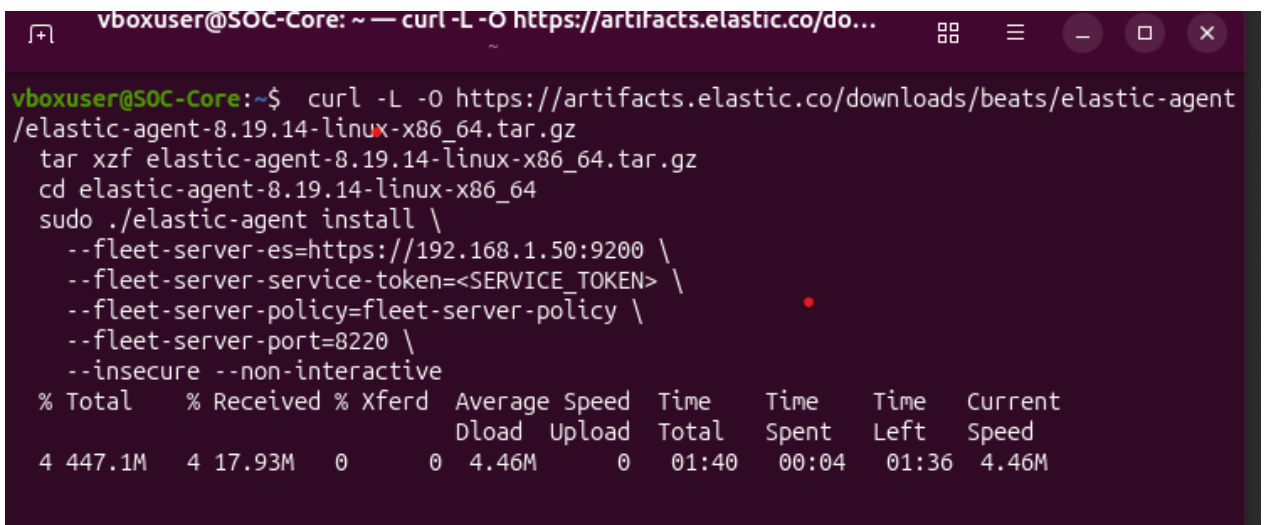


```
vboxuser@SOC-Core: ~ — sudo apt update

vboxuser@SOC-Core:~$ ^[[200~curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch \
> | sudo gpg --dearmor -o /usr/share/keyrings/elastic-8.gpg
[sudo: authenticate] Password: *****

curl: command not found
sudo: Authentication failed, try again.
[sudo: authenticate] Password:
sudo: Authentication failed, try again.
[sudo: authenticate] Password:
sudo: maximum 3 incorrect authentication attempts
vboxuser@SOC-Core:~$ curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch \
| sudo gpg --dearmor -o /usr/share/keyrings/elastic-8.gpg
echo "deb [signed-by=/usr/share/keyrings/elastic-8.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" \
| sudo tee /etc/apt/sources.list.d/elastic-8.x.list
sudo apt update
sudo apt install -y elasticsearch kibana logstash
[sudo: authenticate] Password:
File '/usr/share/keyrings/elastic-8.gpg' exists. Overwrite? (y/N) y
deb [signed-by=/usr/share/keyrings/elastic-8.gpg] https://artifacts.elastic.co/packages/8.x
/apt stable main
Get:1 http://download.zerotier.com/debian/noble noble InRelease [20.5 kB]
Hit:2 https://download.docker.com/linux/ubuntu noble InRelease
Hit:3 http://tn.archive.ubuntu.com/ubuntu resolute InRelease
Hit:4 http://security.ubuntu.com/ubuntu resolute-security InRelease
Hit:5 http://tn.archive.ubuntu.com/ubuntu resolute-updates InRelease
Hit:6 http://tn.archive.ubuntu.com/ubuntu resolute-backports InRelease
Hit:7 https://artifacts.elastic.co/packages/8.x/apt stable InRelease
0% [Working] 3,237 B/s 0s
```

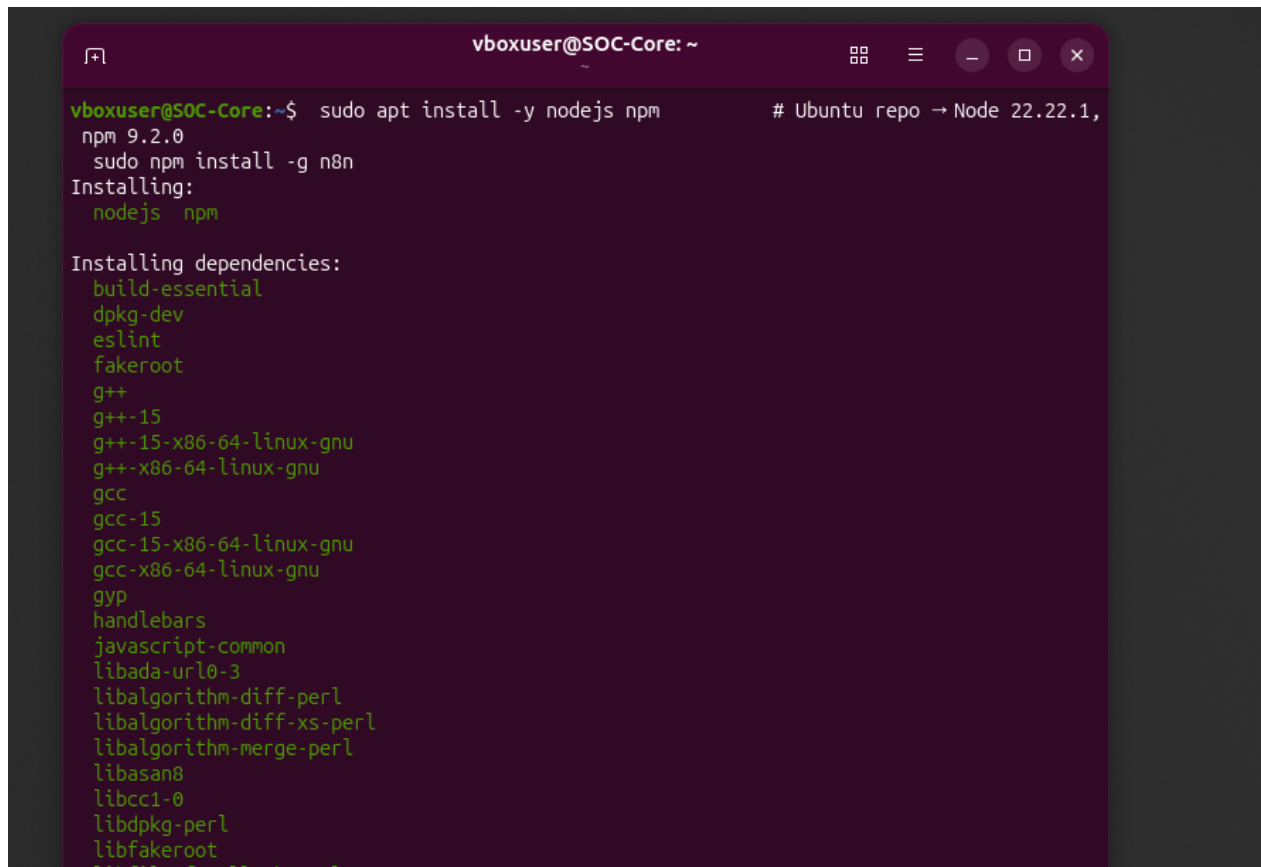
Figure A.1: Installing the Elastic Stack



```
vboxuser@SOC-Core: ~ — curl -L -O https://artifacts.elastic.co/do...

vboxuser@SOC-Core:~$ curl -L -O https://artifacts.elastic.co/downloads/beats/elastic-agent
/elastic-agent-8.19.14-linux-x86_64.tar.gz
tar xzf elastic-agent-8.19.14-linux-x86_64.tar.gz
cd elastic-agent-8.19.14-linux-x86_64
sudo ./elastic-agent install \
--fleet-server-es=https://192.168.1.50:9200 \
--fleet-server-service-token=<SERVICE_TOKEN> \
--fleet-server-policy=fleet-server-policy \
--fleet-server-port=8220 \
--insecure --non-interactive
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             %    Dload  Upload   Total   Spent    Left   Speed
 4 447.1M    4 17.93M    0     0  4.46M      0  01:40    00:04    01:36  4.46M
```

Figure A.2: Installing the Elastic Agent as Fleet Server on SOC-Core

A terminal window titled 'vboxuser@SOC-Core: ~' with standard window controls. The terminal shows the following commands and output:

```
vboxuser@SOC-Core:~$ sudo apt install -y nodejs npm          # Ubuntu repo → Node 22.22.1,
npm 9.2.0
sudo npm install -g n8n
Installing:
nodejs npm

Installing dependencies:
build-essential
dpkg-dev
eslint
fakeroot
g++
g++-15
g++-15-x86-64-linux-gnu
g++-x86-64-linux-gnu
gcc
gcc-15
gcc-15-x86-64-linux-gnu
gcc-x86-64-linux-gnu
gyp
handlebars
javascript-common
libada-url0-3
libalgorithm-diff-perl
libalgorithm-diff-xs-perl
libalgorithm-merge-perl
libasan8
libcc1-0
libdpkg-perl
libfakeroot
```

Figure A.3: Installing Node.js and n8n

A.2 Incident-management host: case management and threat intelligence

```
vboxuser@incident-mgmt:~$ curl -fsSL https://artifacts.elastic.co/GPG-KEY-elasticsearch \
| sudo gpg --dearmor -o /usr/share/keyrings/elastic-8.gpg
echo "deb [signed-by=/usr/share/keyrings/elastic-8.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" \
| sudo tee /etc/apt/sources.list.d/elastic-8.list
sudo apt update
sudo apt install -y elasticsearch
deb [signed-by=/usr/share/keyrings/elastic-8.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main
Get:1 https://artifacts.elastic.co/packages/8.x/apt stable InRelease [3,249 B]
Get:2 http://download.zerotier.com/debian/noble noble InRelease [20.5 kB]
Get:3 https://artifacts.elastic.co/packages/8.x/apt stable/main amd64 Packages [107 kB]
Hit:4 http://tn.archive.ubuntu.com/ubuntu resolute InRelease
Hit:5 http://tn.archive.ubuntu.com/ubuntu resolute-updates InRelease
Hit:6 http://security.ubuntu.com/ubuntu resolute-security InRelease
Hit:7 http://tn.archive.ubuntu.com/ubuntu resolute-backports InRelease
0% [Connecting to apache.jfrog.io]
```

Figure A.4: Installing Elasticsearch on the incident-management host

```

vboxuser@incident-mgmt:~$ curl -fsSL https://downloads.apache.org/cassandra/KEYS \
| sudo gpg --dearmor -o /usr/share/keyrings/apache-cassandra.gpg
echo "deb [signed-by=/usr/share/keyrings/apache-cassandra.gpg] https://deb.debian.org/debian/apache.org 41x main" \
| sudo tee /etc/apt/sources.list.d/cassandra.sources.list
sudo apt update
sudo apt install -y cassandra
[sudo: authenticate] Password:
deb [signed-by=/usr/share/keyrings/apache-cassandra.gpg] https://deb.debian.org/debian/apache.org 41x main
Get:1 http://download.zerotier.com/debian/noble noble InRelease [20.5 kB]
Hit:2 http://tn.archive.ubuntu.com/ubuntu resolute InRelease
Get:3 http://tn.archive.ubuntu.com/ubuntu resolute-updates InRelease [136 kB]
Get:4 http://security.ubuntu.com/ubuntu resolute-security InRelease [136 kB]
Hit:5 http://tn.archive.ubuntu.com/ubuntu resolute-backports InRelease
Get:6 http://tn.archive.ubuntu.com/ubuntu resolute-updates/main amd64 Packages [194 kB]
Get:8 http://tn.archive.ubuntu.com/ubuntu resolute-updates/main Translation-en [50.8 kB]
Get:9 http://tn.archive.ubuntu.com/ubuntu resolute-updates/main amd64 Components [29.6 kB]
Get:10 http://tn.archive.ubuntu.com/ubuntu resolute-updates/universe amd64 Packages [93.1 kB]
Get:11 http://tn.archive.ubuntu.com/ubuntu resolute-updates/universe Translation-en [27.9 kB]
Get:12 http://tn.archive.ubuntu.com/ubuntu resolute-updates/universe amd64 Components [46.0 kB]
Get:13 http://security.ubuntu.com/ubuntu resolute-security/main amd64 Packages [147 kB]
Get:14 http://security.ubuntu.com/ubuntu resolute-security/main Translation-en [43.0 kB]
Get:15 http://security.ubuntu.com/ubuntu resolute-security/main amd64 Components [23.6 kB]
Get:16 http://security.ubuntu.com/ubuntu resolute-security/universe amd64 Packages [89.4 kB]
Get:17 http://security.ubuntu.com/ubuntu resolute-security/universe Translation-en [26.8 kB]
Get:18 http://security.ubuntu.com/ubuntu resolute-security/universe amd64 Components [39.7 kB]
0% [Connecting to apache.jfrog.io] 178 kB/s 0s

```

Figure A.5: Installing Apache Cassandra, TheHive's primary datastore

```
vboxuser@incident-mgmt:~$ sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.
asc
echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com
/linux/ubuntu noble stable" \
| sudo tee /etc/apt/sources.list.d/docker.list
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-com
pose-plugin
deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/u
buntu noble stable
Get:1 http://download.zerotier.com/debian/noble noble InRelease [20.5 kB]
Get:2 https://download.docker.com/linux/ubuntu noble InRelease [48.5 kB]
Hit:3 https://artifacts.elastic.co/packages/8.x/apt stable InRelease
Get:4 https://download.docker.com/linux/ubuntu noble/stable amd64 Packages [56.1 kB]
Hit:5 http://tn.archive.ubuntu.com/ubuntu resolute InRelease
Hit:6 http://security.ubuntu.com/ubuntu resolute-security InRelease
Hit:7 http://tn.archive.ubuntu.com/ubuntu resolute-updates InRelease
Hit:8 http://tn.archive.ubuntu.com/ubuntu resolute-backports InRelease
Hit:9 https://apache.jfrog.io/artifactory/cassandra-deb 41x InRelease
Fetched 125 kB in 21s (5,846 B/s)
39 packages can be upgraded. Run 'apt list --upgradable' to see them.
Installing:
  containerd.io  docker-buildx-plugin  docker-ce  docker-ce-cli  docker-compose-plugin

Installing dependencies:
  docker-ce-rootless-extras  pigz
```

Figure A.6: Installing the Docker engine, the runtime for the TheHive, Cortex and MISP containers.

```
vboxuser@incident-mgmt:~$ cd /soc-project/thehive-cortex
sudo docker compose build
sudo docker compose up -d
```

Figure A.7: Building and starting the TheHive and Cortex containers with Docker Compose.

```
vboxuser@incident-mgmt:~/soc-project$ cd ..
vboxuser@incident-mgmt:~$ cd ~/soc-project/misp
sudo docker compose build
sudo docker compose up -d
```

Figure A.8: Building and starting the MISP container with Docker Compose.

```
#!/bin/bash
# MISP feed auto-fetch – runs every 6h via host crontab.
# fetch+cache for all enabled feeds. Exit codes ignored so a transient
# feed failure (e.g. provider down) doesn't trip cron mail spam.

set +e
LOG=/home/vboxuser/soc-project/misp/cron-feeds.log
{
  echo "=== $(date -Is) fetchFeed all ==="
  docker exec misp-misp-core-1 sudo -u www-data \
    /var/www/MISP/app/Console/cake Server fetchFeed 1 all 2>&1 | tail -5
  echo "=== $(date -Is) cacheFeed all ==="
  docker exec misp-misp-core-1 sudo -u www-data \
    /var/www/MISP/app/Console/cake Server cacheFeed 1 all 2>&1 | tail -5
  echo ""
} >> "$LOG" 2>&1

# rotate log if > 5MB
if [ -f "$LOG" ] && [ "$(stat -c%s "$LOG" 2>/dev/null || echo 0)" -gt 5242880 ]; then
  mv "$LOG" "${LOG}.1"
fi
```

Figure A.9: cron-feeds.sh: the host cron job

Appendix B

The AI decision layer

This appendix collects the source behind the two AI components of Section 3.5: the supervised triage classifier (Section 3.5.1) and the local language model (Section 3.5.2). The first three figures are the classifier’s feature schema, its inference path and its deterministic safety floor; the last three are the prompt templates that drive the escalation writing. The prompts run as a chain — correlation first, then the escalation brief that consumes it, then the remediation recommendation — and each returns strict JSON that populates advisory fields for the analyst, never the verdict, severity or response, which stay deterministic.

```
FEATURE_NAMES: list[str] = [  
    "risk_score",  
    "severity_ordinal",  
    "rule_source_idx",  
    "rule_category_idx",  
    "rule_fp_rate_30d",  
    "rule_volume_30d_log",  
    "mitre_tactic_idx",  
    "mitre_technique_hash",  
    "source_ip_is_internal",  
    "source_ip_is_management",  
    "hour_of_day",  
    "day_of_week",  
    "is_business_hours",  
    "auto_block_flag",  
    "alert_count_in_window",  
]
```

Figure B.1: The fifteen features the triage classifier consumes

```
def _classify_alert(alert: dict, model_kind: str | None) -> dict:
    model, kind = _pick_model(model_kind)
    if model is None:
        return {
            "error": "no model loaded",
            "predicted_label": "unknown",
            "confidence": 0.0,
            "recommended_action": "send_to_analyst",
        }
    vec = alert_to_feature_vector(alert, _rule_stats).reshape(1, -1)
    proba = model.predict_proba(vec)[0]
    pred = int(np.argmax(proba))
    confidence = float(proba[pred])
    label_str = "true_positive" if pred == 1 else "false_positive"
    risk = float(alert.get("risk_score") or 0)
    alert_count = float(alert.get("alert_count_in_window") or alert.get("alert_count") or 1)
    auto_block = bool(alert.get("auto_block_flag") or alert.get("auto_block"))
    action = _recommended_action(pred, confidence, risk, alert_count, auto_block)
    importances = getattr(model, "feature_importances_", None)
    top3 = []
    if importances is not None:
        idx = np.argsort(importances)[::-1][:3]
        top3 = [FEATURE_NAMES[i] for i in idx]
    return {
        "predicted_label": label_str,
        "confidence": round(confidence, 4),
        "recommended_action": action,
        "model": "RandomForest_v2" if kind == "rf" else "XGBoost_v2",
        "model_kind": kind,
        "trained_at": _meta.get("trained_at"),
        "dataset_source": _meta.get("dataset_source", "unknown"),
        "feature_importance_top3": top3,
    }
```

Figure B.2: The classifier inference path

```
def _recommended_action(
    label: int,
    confidence: float,
    risk_score: float,
    alert_count: float = 1.0,
    auto_block: bool = False,
) -> str:
    # Deterministic SAFETY FLOOR. The ML verdict is advisory – it is NEVER the
    # sole authority for the one irreversible automated action (auto-close FP).
    # A confident-FP is only allowed to auto-close when the alert is genuinely
    # low-stakes: low/medium risk, a single (non-bursty) firing, and the rule
    # did not request auto-block. A real high/critical, bursty, or block-flagged
    # alert the model *thinks* is FP is sent to an analyst instead of silently
    # closed – this is what stops a noisy CICIDS-trained model from auto-closing
    # a real web shell / reverse shell. The model sharpens over time via
    # /feedback; the floor is the human-trust guarantee.
    if confidence < CONFIDENCE_LOW:
        return "send_to_analyst"
    if label == 0 and confidence >= CONFIDENCE_HIGH:
        if risk_score < RISK_HIGH and alert_count <= 3 and not auto_block:
            return "auto_close_fp"
        return "send_to_analyst"
    if label == 1 and confidence >= CONFIDENCE_HIGH and risk_score >= RISK_HIGH:
        return "auto_escalate"
    if label == 1 and confidence >= CONFIDENCE_HIGH:
        return "send_to_analyst"
    return "queue_low_priority"
```

Figure B.3: The deterministic safety floor

```
You are a senior SOC analyst clustering related security alerts into likely
attack campaigns. You will be given a current case plus a list of other
alerts that fired around the same time from the same source IP or against
the same destination IP.

Your job: group these alerts into 1-4 likely "narrative threads" (could be
a single campaign or unrelated noise), each tagged with a kill-chain stage.
Return STRICT JSON only - no prose outside the JSON object.

# Current case
case_id: {{case_id}}
title: {{case_title}}
severity: {{case_severity}}
mitre_tactic: {{case_mitre_tactic}}
mitre_technique: {{case_mitre_technique}}
source_ip: {{case_source_ip}}
destination_ip: {{case_destination_ip}}
timestamp: {{case_timestamp}}

# Related alerts found in {{window_minutes}}-minute window
{{related_alerts_json}}

# Required output schema
{
  "clusters": [
    {
      "label": "<short human label, e.g. 'SSH brute force followed by lateral move'>",
      "kill_chain_stage": "<one of: reconnaissance | initial-access | execution | persistence | privilege-escalation | defense-evasion | credential-access | discovery>",
      "confidence": "<one of: low | medium | high>",
      "alert_ids": [<"<alert id from input>", "...>"],
      "narrative": "<one sentence explaining what likely happened in this cluster>"
    }
  ],
  "overall_assessment": "<one sentence: are these alerts likely a single campaign, multiple campaigns, or noise?>",
  "kill_chain_progression_detected": true
}

Rules:
- The current case alert is implicitly in the first cluster. List its case_id as the first alert_id.
- If an alert is clearly unrelated noise, put it in a separate cluster with confidence "low".
- If you can identify NO clusters beyond the current case alone, return one cluster containing only the case.
- Output the JSON object and NOTHING else.
```

Figure B.4: The alert-correlation prompt

```

You are a senior SOC analyst writing a concise L2 escalation brief for an incident that has just been escalated to second-tier review. The brief must be readable by an analyst in under 60 seconds and must surface exactly what L2 needs to decide whether to contain, investigate further, or close.

Return STRICT JSON only - no prose outside the JSON object.

# Case under review
case_id: {{case_id}}
title: {{case_title}}
severity: {{case_severity}} (1=low, 4=critical)
tags: {{case_tags}}
escalation_reason: {{escalation_reason}}

# Observables attached to this case
{{observables_json}}

# Last 5 comments on this case
{{recent_comments}}

# Correlation result from earlier prompt
{{cluster_summary}}

# Required output schema
{
  "situation": "<2-3 sentence plain-English description of what is happening>",
  "kill_chain": [
    {"stage": "<kill chain stage>", "evidence": "<one-sentence evidence from observables/alerts>"}
  ],
  "impact": {
    "affected_assets": ["<hostname or IP>", "..."],
    "data_at_risk": "<what could be exfiltrated/alterd, or 'none currently'>",
    "blast_radius": "<one of: single-host | subnet | crown-jewel | unknown>"
  },
  "urgency_score": 0.0,
  "key_indicators": [
    "<short bullet, max 12 words>"
  ],
  "open_questions_for_l2": [
    "<short question the L2 analyst should answer first>"
  ]
}

Rules:
- urgency_score >= 0.8 only if blast_radius is crown-jewel OR data exfiltration evidence is present.
- urgency_score is a float between 0.0 and 1.0.
- key_indicators has 3-5 entries, ordered by importance.
- open_questions_for_l2 has 2-4 entries.
- Output the JSON object and NOTHING else.

```

Figure B.5: The L2 escalation-brief prompt

```

You are a senior SOC analyst recommending concrete remediation actions for
an incident that has been escalated to L2. Your recommendations must be
tied to the MITRE ATT&CK Mitigations catalogue (M1037, M1041, etc.) and
include a concrete command or playbook step where possible.

Return STRICT JSON only - no prose outside the JSON object.

# Incident context
mitre_technique_id: {{mitre_technique_id}}
mitre_tactic_id: {{mitre_tactic_id}}
asset_criticality: {{asset_criticality}}
affected_assets: {{affected_assets}}

# Situation summary (from previous step)
{{summary_situation}}

# Required output schema
{
  "actions": [
    {
      "priority": "<P1|P2|P3>",
      "action": "<short imperative sentence, e.g. 'Isolate the affected host from internal VLAN'>",
      "mitre_mitigation_id": "<M1037|M1041|M1042|M1018|M1026|M1027|M1030|M1031|M1036|M1043|M1044|M1045|M1046|M1047|M1048|M1049|M1050|M1051 etc.>",
      "mitre_mitigation_name": "<canonical name, e.g. 'Filter Network Traffic'>",
      "command": "<concrete shell command, API call, or playbook ref - or empty string if not applicable>",
      "owner_role": "<one of: ll-soc | l2-soc | network-admin | sysadmin | dev-team | legal>",
      "estimated_time_minutes": 0
    }
  ],
  "containment_window_minutes": 0,
  "post_action_checks": [
    "<short verification step the analyst should run after applying the actions>"
  ]
}

Rules:
- Provide 3 to 5 actions, ordered by priority (P1 first).
- At least one action MUST be P1.
- Map every action to a real ATT&CK mitigation ID (M-prefix).
- For containment_window_minutes use a realistic positive integer (e.g. 15 for crown-jewel, 60 for subnet, 240 for single-host).
- post_action_checks has 2-3 entries.
- Output the JSON object and NOTHING else.

```

Figure B.6: The remediation-recommendation prompt

Appendix C

The automated response library

This appendix shows the response path of Section 3.6.3: the Cortex responders an analyst clicks, the orchestrator they trigger, and the vetted action scripts it dispatches. The responders are thin shims that only post to an n8n webhook; all logic lives in the workflow and the scripts. Every action script defaults to a dry run and executes for real only when explicitly confirmed, and each carries guards that refuse the laboratory control plane.

```
import requests
from cortexutils.responder import Responder

class SOCConfirmFP(Responder):
    def __init__(self):
        Responder.__init__(self)

    def run(self):
        Responder.run(self)
        data = self.get_data() or {}
        obj = data.get("object", data) if isinstance(data, dict) else {}
        case_id = obj.get("_id") or obj.get("id") or ""
        if not case_id:
            self.error("No case_id in input data.")
            return
        n8n_url = self.get_param("config.n8n_webhook_url",
                                "http://192.168.1.50:5678/webhook/l1-decision")
        l1_user = (self.get_param("parameters.user", None)
                  or obj.get("updatedBy") or obj.get("createdBy")
                  or "soc-bot@thehive.local")
        body = {"case_id": case_id, "decision": "fp", "l1_user": l1_user, "source": "cortex_responder"}
        try:
            r = requests.post(n8n_url, json=body, timeout=20)
            r.raise_for_status()
            self.report({"message": "Case closed as FP via WF11", "http_status": r.status_code,
                        "case_id": case_id, "l1_user": l1_user})
        except Exception as e:
            self.error(f"WF11 webhook POST failed: {e}")

if __name__ == "__main__":
    SOCConfirmFP().run()
```

Figure C.1: The SOC_Confirm_FP Cortex responder

```
vboxuser@incident-mgmt:~/soc-project/thehive-cortex/cortex/Cortex-Analyzers/responders/SOC$ cat SOC_Confirm_FP.json
{
  "name": "SOC_Confirm_FP",
  "version": "1.0",
  "author": "SOC-LAB (PFE)",
  "url": "https://soc-lab.local",
  "license": "AGPL-V3",
  "description": "L1 verdict: confirm False Positive - closes case via WF11 and feeds the ML model.",
  "dataTypeList": ["thehive:case"],
  "command": "SOC/SOC_Confirm_FP.py",
  "baseConfig": "SOC",
  "configurationItems": [
    {
      "name": "n8n_webhook_url",
      "description": "n8n WF11 L1 decision webhook URL",
      "type": "string",
      "multi": false,
      "required": true,
      "defaultValue": "http://192.168.1.50:5678/webhook/l1-decision"
    }
  ]
}
```

Figure C.2: The Cortex responder manifest for SOC_Confirm_FP

```
import requests
from cortexutils.responder import Responder

class SOCAutoblockIsolate(Responder):
    def __init__(self):
        Responder.__init__(self)

    def run(self):
        Responder.run(self)
        data = self.get_data() or {}
        obj = data.get("object", data) if isinstance(data, dict) else {}
        case_id = obj.get("_id") or obj.get("id") or ""
        if not case_id:
            self.error("No case _id in input data.")
            return
        n8n_url = self.get_param("config.n8n_webhook_url",
                                "http://192.168.1.50:5678/webhook/response-action")
        l1_user = (self.get_param("parameters.user", None)
                  or obj.get("updatedBy") or obj.get("createdBy")
                  or "soc-bot@thehive.local")
        body = {"case_id": case_id, "action": "ssh_autoblock", "confirm": True,
               "l1_user": l1_user, "source": "cortex_responder"}
        try:
            r = requests.post(n8n_url, json=body, timeout=20)
            r.raise_for_status()
            self.report({"message": "Autoblock requested via WF12", "http_status": r.status_code,
                       "case_id": case_id, "l1_user": l1_user})
        except Exception as e:
            self.error(f"WF12 webhook POST failed: {e}")

if __name__ == "__main__":
    SOCAutoblockIsolate().run()
```

Figure C.3: The SOC_Autoblock_Isolate Cortex responder

```

def main() -> int:

    if args.action == "auto":
        mapped = _resolve_action_from_map(args.technique_id)
        action = mapped["action"]
        params = mapped["params_from_case"]
    else:
        action = args.action
        defaults_by_action = {
            "ssh_autoblock": ["source_ip"],
            "kill_process": ["host_ip", "process_name"],
            "disable_user": ["host_ip", "username"],
            "quarantine_file": ["host_ip", "file_path"],
            "delete_registry": ["host_ip", "reg_key"],
        }
        params = defaults_by_action.get(action, [])

    script = SCRIPT_BY_ACTION.get(action)
    if not script or not script.exists():
        result = {"action": action, "status": "error",
                  "details": f"unknown action '{action}' or script not found",
                  "case_id": args.case_id}
        print(json.dumps(result))
        _log(f"{dt.datetime.now(dt.UTC).isoformat()} {args.requester} {args.case_id} {action} ERROR_NO_SCRIPT")
        return 2

    cli_args = _build_script_args(action, params, observables, overrides)
    if args.confirm:
        cli_args.append("--confirm")
    try:
        cp = subprocess.run(
            [str(script), *cli_args],
            capture_output=True, text=True, timeout=60,
            env={**os.environ, "SOC_RESPONSE_CONFIRMED": "1" if args.confirm else "0"},
        )
        try:
            result = json.loads(cp.stdout.strip().splitlines()[-1])
        except Exception:
            result = {"action": action, "status": "error",
                      "details": f"non-JSON: stdout={cp.stdout[-300:]} stderr={cp.stderr[-300:]}",
                      "rc": cp.returncode}
    except subprocess.TimeoutExpired:
        result = {"action": action, "status": "timeout", "details": "60s timeout"}
    except Exception as e:
        result = {"action": action, "status": "error", "details": f"runner exception: {e!r}"}
    result["case_id"] = args.case_id
    result["ts"] = dt.datetime.now(dt.UTC).isoformat()
    result["requester"] = args.requester
    result["dry_run"] = not args.confirm
    print(json.dumps(result))
    _log(f"{result['ts']} {args.requester} {args.case_id} {action} {result.get('status','?')}")
    return 0

```

Figure C.4: The response orchestrator invoked by the response-action workflow

```
set -euo pipefail

TARGET_IP="${1:-}"
CONFIRM_FLAG="${2:-}"
VM_B2="192.168.1.53"
SSH_USER="soc-response"
SSH_KEY="/home/vboxuser/.ssh/soc_response"

LAB_IPS="^(192\,168\,1\,(50|51|52|53)|127\,10\,0\,2\,)"

if [[ -z "$TARGET_IP" ]]; then
    echo '{"action":"ssh_autoblock","status":"error","details":"missing target_ip arg"}'
    exit 2
fi

if [[ "$TARGET_IP" =~ $LAB_IPS ]]; then
    echo '{"action":"ssh_autoblock","target_ip":"$TARGET_IP","status":"refused","details":"target is in lab control plane"}'
    exit 3
fi

CONFIRMED=0
if [[ "$CONFIRM_FLAG" == "--confirm" ]] || [[ "${SOC_RESPONSE_CONFIRMED:-0}" == "1" ]]; then
    CONFIRMED=1
fi

if [[ "$CONFIRMED" -eq 0 ]]; then
    echo '{"action":"ssh_autoblock","target_ip":"$TARGET_IP","status":"dry_run","details":"would: ssh ${SSH_USER}@${VM_B2} sudo /sbin/iptables -I INPUT'}
    exit 0
fi

OUTPUT=$(ssh -o BatchMode=yes -o StrictHostKeyChecking=accept-new -i "$SSH_KEY" \
    "${SSH_USER}@${VM_B2}" \
    "sudo /sbin/iptables -I INPUT -s ${TARGET_IP} -j DROP && sudo /sbin/iptables -nL INPUT | grep ${TARGET_IP} | head -1" 2>&1) || RC=$?
RC=${RC:-0}

ESC_OUTPUT=$(printf '%s' "$OUTPUT" | python3 -c "import sys,json; print(json.dumps(sys.stdin.read()))")
echo '{"action":"ssh_autoblock","target_ip":"$TARGET_IP","status":${( [[ $RC -eq 0 ]] && echo "ok" || echo "failed" )},"host":"$VM_B2","rc":$RC,\`
exit $RC
```

Figure C.5: The ssh_autoblock action script

```
set -euo pipefail

TARGET_HOST="${1:-}"
PID_OR_NAME="${2:-}"
CONFIRM_FLAG="${3:-}"
SSH_USER="soc-response"
SSH_KEY="/home/vboxuser/.ssh/soc_response"
LAB_HOSTS="^(192\,168\,1\,(50|51|52))$"

if [[ -z "$TARGET_HOST" || -z "$PID_OR_NAME" ]]; then
    echo '{"action":"kill_process","status":"error","details":"usage: kill_process.sh <host_ip> <pid_or_name> [--confirm]"}'
    exit 2
fi

if [[ "$TARGET_HOST" =~ $LAB_HOSTS ]]; then
    echo '{"action":"kill_process","target_host":"$TARGET_HOST","status":"refused","details":"refuse to kill on lab control plane"}'
    exit 3
fi

CONFIRMED=0
if [[ "$CONFIRM_FLAG" == "--confirm" ]] || [[ "${SOC_RESPONSE_CONFIRMED:-0}" == "1" ]]; then
    CONFIRMED=1
fi

if [[ "$PID_OR_NAME" =~ ^[0-9]+$ ]]; then
    CMD="sudo kill -9 ${PID_OR_NAME} && echo killed_pid_${PID_OR_NAME}"
else
    SAFE_NAME=$(printf '%s' "${PID_OR_NAME}" | tr -cd 'a-zA-Z0-9._-')
    CMD="sudo pkill -9 -x ${SAFE_NAME} && echo killed_name_${SAFE_NAME}"
fi

if [[ "$CONFIRMED" -eq 0 ]]; then
    echo '{"action":"kill_process","target_host":"$TARGET_HOST","target":"$PID_OR_NAME","status":"dry_run","details":"would: ssh ${SSH_USER}@${TARGET_HOST} '$CMD'"
    exit 0
fi

OUTPUT=$(ssh -o BatchMode=yes -o StrictHostKeyChecking=accept-new -i "$SSH_KEY" \
    "${SSH_USER}@${TARGET_HOST}" "$CMD" 2>&1) || RC=$?
RC=${RC:-0}

ESC_OUTPUT=$(printf '%s' "$OUTPUT" | python3 -c "import sys,json; print(json.dumps(sys.stdin.read()))")
echo '{"action":"kill_process","target_host":"$TARGET_HOST","target":"$PID_OR_NAME","status":${( [[ $RC -eq 0 ]] && echo "ok" || echo "failed" )},"rc":$RC,\`
exit $RC
```

Figure C.6: The kill_process action script

```
TARGET_HOST="${1:-}"
USERNAME="${2:-}"
CONFIRM_FLAG="${3:-}"
SSH_USER="soc-response"
SSH_KEY="/home/vboxuser/.ssh/soc_response"
LAB_HOSTS=""(192\.\168\.\1\.(50|51|52))$
PROTECTED_USERS=""(root|soc-response|vboxuser|kali|admin)$

if [[ -z "$TARGET_HOST" || -z "$USERNAME" ]]; then
    echo '{"action":"disable_user","status":"error","details":"usage: disable_user.sh <host_ip> <username> [--confirm]"}'
    exit 2
fi

if [[ "$TARGET_HOST" == $LAB_HOSTS ]]; then
    echo '{"action":"disable_user","target_host":"$TARGET_HOST","status":"refused","details":"refuse on lab control plane"}'
    exit 3
fi

if [[ "$USERNAME" == $PROTECTED_USERS ]]; then
    echo '{"action":"disable_user","username":"$USERNAME","status":"refused","details":"refuse to lock protected account"}'
    exit 3
fi

CONFIRMED=0
if [[ "$CONFIRM_FLAG" == "--confirm" ]] || [[ "${SOC_RESPONSE_CONFIRMED:-0}" == "1" ]]; then
    CONFIRMED=1
fi

SAFE_USER=$(printf '%s' "$USERNAME" | tr -cd 'a-zA-Z0-9_-' )
CMD="sudo usermod -L ${SAFE_USER} && sudo passwd -l ${SAFE_USER} 2>&1 | tail -1"

if [[ "$CONFIRMED" -eq 0 ]]; then
    echo '{"action":"disable_user","target_host":"$TARGET_HOST","username":"$SAFE_USER","status":"dry_run","details":"would: ssh ${SSH_USER}@${TARGET_HOST} "$CMD"}'
    exit 0
fi

OUTPUT=$(ssh -o BatchMode=yes -o StrictHostKeyChecking=accept-new -i "$SSH_KEY" \
    "${SSH_USER}@${TARGET_HOST}" "$CMD" 2>&1) || RC=$?
RC=${RC:-0}

ESC_OUTPUT=$(printf '%s' "$OUTPUT" | python3 -c "import sys,json; print(json.dumps(sys.stdin.read()))")
echo '{"action":"disable_user","target_host":"$TARGET_HOST","username":"$SAFE_USER","status":${ [[ $RC -eq 0 ]] && echo "ok" || echo "failed" },\nc\n":$RC,\ndeta
exit $RC
```

Figure C.7: The disable_user action script

```
TARGET_HOST="${1:-}"
FILE_PATH="${2:-}"
CONFIRM_FLAG="${3:-}"
SSH_USER="soc-response"
SSH_KEY="/home/vboxuser/.ssh/soc_response"
LAB_HOSTS=""(192\.\168\.\1\.(50|51|52))$

if [[ -z "$TARGET_HOST" || -z "$FILE_PATH" ]]; then
    echo '{"action":"quarantine_file","status":"error","details":"usage: quarantine_file.sh <host_ip> <abs_path> [--confirm]"}'
    exit 2
fi

if [[ "${FILE_PATH:0:1}" != "/" ]]; then
    echo '{"action":"quarantine_file","status":"error","details":"file path must be absolute"}'
    exit 2
fi

if [[ "$TARGET_HOST" == $LAB_HOSTS ]]; then
    echo '{"action":"quarantine_file","target_host":"$TARGET_HOST","status":"refused","details":"refuse on lab control plane"}'
    exit 3
fi

case "$FILE_PATH" in
    /bin/*|sbin/*|usr/bin/*|usr/sbin/*|lib/*|usr/lib/*|etc/*|boot/*|proc/*|sys/*)
        echo '{"action":"quarantine_file","file_path":"$FILE_PATH","status":"refused","details":"refuse to quarantine system path"}'
        exit 3
    ;;
esac

CONFIRMED=0
if [[ "$CONFIRM_FLAG" == "--confirm" ]] || [[ "${SOC_RESPONSE_CONFIRMED:-0}" == "1" ]]; then
    CONFIRMED=1
fi

SAFE_PATH=$(printf '%s' "$FILE_PATH" | tr -d '\n\r')
QPATH="/var/quarantine/${echo "$SAFE_PATH" | tr '/' '_'}.$(date +%s)"
CMD="sudo mkdir -p /var/quarantine && sudo mv '${SAFE_PATH}' '${QPATH}' && sudo chmod 000 '${QPATH}' && echo quarantined:${QPATH}"

if [[ "$CONFIRMED" -eq 0 ]]; then
    echo '{"action":"quarantine_file","target_host":"$TARGET_HOST","file_path":"$SAFE_PATH","status":"dry_run","details":"would: ssh ${SSH_USER}@${TARGET_HOST}
    exit 0
fi

OUTPUT=$(ssh -o BatchMode=yes -o StrictHostKeyChecking=accept-new -i "$SSH_KEY" \
    "${SSH_USER}@${TARGET_HOST}" "$CMD" 2>&1) || RC=$?
RC=${RC:-0}

ESC_OUTPUT=$(printf '%s' "$OUTPUT" | python3 -c "import sys,json; print(json.dumps(sys.stdin.read()))")
echo '{"action":"quarantine_file","target_host":"$TARGET_HOST","file_path":"$SAFE_PATH","quarantine_path":"$QPATH","status":${ [[ $RC -eq 0 ]] && echo "ok"
exit $RC
```

Figure C.8: The quarantine_file action script

```
set -euo pipefail

TARGET_HOST="${1:-}"
REG_KEY="${2:-}"

if [[ -z "$TARGET_HOST" || -z "$REG_KEY" ]]; then
    echo '{"action":"delete_registry","status":"error","details":"usage: delete_registry.sh <host_ip> <reg_key> [--confirm]"}'
    exit 2
fi

echo '{"action":"delete_registry","target_host":"$TARGET_HOST","reg_key":"$REG_KEY","status":"not_implemented","details":'
exit 0
```

Figure C.9: The delete_registry action script

Abstract

This report presents an autonomous, human-in-the-loop Security Operations Center (SOC) that automates incident response and reporting with artificial intelligence. Modern SOC's detect and manage threats well, but two steps stay manual: responding to a confirmed threat and writing each incident up as a report and a ticket. The proposed SOC is built entirely from open-source tools at no recurring cost (Suricata, the Elastic Stack, MISP, TheHive, Cortex, and n8n). A supervised classifier triages every alert, while a locally hosted language model drafts the analyst's brief without taking any decision. Every consequential action, above all containment, stays behind a human confirmation. Tested end to end against emulated attacks, the system confirmed reliable detection, triage, AI-generated reporting, and human-confirmed response.

Keywords: SOC automation, alert triage, machine learning, incident response, AI generated reporting, open source.

Résumé

Ce rapport présente un Security Operations Center (SOC) autonome, gardant l'humain dans la boucle, qui automatise la réponse aux incidents et leur reporting à l'aide de l'intelligence artificielle. Les SOC modernes détectent et gèrent bien les menaces, mais deux étapes restent manuelles : la réponse à une menace confirmée et la rédaction de chaque incident sous forme de rapport et de ticket. La solution est construite entièrement à partir d'outils open-source et sans coût récurrent (Suricata, la suite Elastic, MISP, TheHive, Cortex et n8n). Un classifieur supervisé trie chaque alerte, tandis qu'un modèle de langage hébergé localement rédige la note de l'analyste sans prendre aucune décision. Toute action conséquente, en premier lieu le confinement, reste soumise à une confirmation humaine. Testé de bout en bout face à des attaques émulées, le système a confirmé la fiabilité de la détection, du tri, de la génération de rapports par IA et de la réponse validée par l'humain.

Mots clés: Automatisation du SOC, tri des alertes, apprentissage automatique, réponse aux incidents, rapports par IA, open source.