Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп’ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра

на тему: «Проект SIEM-системи на базі відкритого рішення Drain3»

Виконала: студентка 4-го курсу групи КБ-22-3
спеціальності 125 «Кіберберзпека»
_____ Марія МОСКАЛЕНКО
Керівник: доцент кафедри КІ та КБ, к.т.н.
_____ Олексій ШЕЛУХА
Рецензент: Доцент кафедри КІ та КБ, к.пед.н., доцент
_____ Олена ГОЛОВНЯ

Житомир – 2026

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ
Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
Андрій Єфіменко
2 лютого 2026 р.

ЗАВДАННЯ
на випускну кваліфікаційну роботу бакалавра

Студент: Москаленко Марія Романівна
Тема роботи: «Проект SIEM-системи на базі відкритого рішення Drain3»
затверджена Наказом університету від 2 березня 2026 р. №58/с
Термін здачі студентом закінченої роботи 8 червня 2026 р.

Вихідні дані роботи:

- процеси виявлення кіберзагроз на основі аналізу потоків log-файлів операційних систем, веб-серверів та мережевого обладнання.
- методи та алгоритми гібридної аномалійної детекції в потоках системних логів із застосуванням алгоритму template mining Drain3 та ансамблевого методу IsolationForest.

Обладнання та програмне забезпечення:

1. Середовище Python 3.11, бібліотеки та фреймворки: FastAPI, SQLAlchemy, drain3, scikit-learn, statsmodels, anthropic, weasyprint;
2. СУБД SQLite ;
3. Docker Desktop для розгортання мультиконтейнерної конфігурації;
4. Anthropic Claude API для реалізації AI-компонентів системи (пояснення алертів, forensic звіти, прогнозний наратив).
5. Середовище розробки Visual Studio Code та Claude Code.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Олександра ПОКОТИЛО	02.02.2026	02.02.2026
2	Олександра ПОКОТИЛО	02.02.2026	02.02.2026
3	Олександра ПОКОТИЛО	02.02.2026	02.02.2026

Студент _____ Марія МОСКАЛЕНКО
Керівник _____ Олексій ШЕЛУХА

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	02.02.2026	Виконано
2	Аналіз літературних джерел та сучасного стану теми	15.02.2026	Виконано
3	Формування структури пояснювальної записки	28.02.2026	Виконано
4	Проектування архітектури SIEM-системи	15.03.2026	Виконано
5	Оформлення 1 розділу пояснювальної записки	01.04.2026	Виконано
6	Реалізація детекційного ядра (rules engine, IsolationForest, Drain3)	20.04.2026	Виконано
7	Розробка веб-інтерфейсу та REST API	30.04.2026	Виконано
8	Розробка модуля forensic-розслідування та інтеграція Claude API	06.05.2026	Виконано
9	Оформлення 2 розділу пояснювальної записки	10.05.2026	Виконано
10	Тестування та збір результатів evaluation	20.05.2026	Виконано
11	Оформлення 3 розділу пояснювальної записки	28.05.2026	Виконано
12	Остаточне формування пояснювальної записки.	08.06.2026	Виконано

Студент _____ Марія МОСКАЛЕНКО

Керівник _____ Олексій ШЕЛУХА

АНОТАЦІЯ

Пояснювальна записка: 91 сторінок, 53 рисунків. Список використаних джерел налічує 36 найменування та розміщений на 3 сторінках.

Метою кваліфікаційної роботи є проектування та реалізація прототипу SIEM-системи з гібридним детекційним ядром, що поєднує rule-based та ML-based підходи на базі алгоритмів Drain3 та IsolationForest, із забезпеченням автоматизованого forensic-документування інцидентів засобами великих мовних моделей.

Проаналізовано сучасні підходи до забезпечення інформаційної безпеки організації та досліджено існуючі SIEM-рішення. Визначено систему функціональних вимог до розроблюваного прототипу та обґрунтовано вибір гібридного підходу до детекції загроз. Спроектовано модульну архітектуру системи з восьми компонентів з асинхронною комунікацією. Розроблено двоетапний pipeline нормалізації логів на основі Grok та Drain3, детекційне ядро з 13 YAML-правилами та IsolationForest, кореляційний рушій з п'ятьма kill chain ланцюжками та модуль forensic-розслідування з інтеграцією Claude API. Наведено результати порівняльного тестування: гібридний підхід виявив у 2.6 рази більше атак порівняно з rule-based (Recall 0.358 проти 0.137), F1-score зріс на 97%, AUC-ROC склав 0.863.

Ключові слова: SIEM, ІНФОРМАЦІЙНА БЕЗПЕКА, ВИЯВЛЕННЯ АНОМАЛІЙ, МАШИННЕ НАВЧАННЯ, ISOLATIONFOREST, DRAIN3, КІБЕРЗАГРОЗИ, FORENSIC-АНАЛІЗ, ВЕЛИКІ МОВНІ МОДЕЛІ, MITRE ATT&CK.

ANNOTATION

The explanatory note to the bachelor's qualification work contains 91 pages, 53 figures.. The list of references includes 36 items and is placed on 3 pages.

The purpose of the qualification work is to design and implement a prototype SIEM system with a hybrid detection engine combining rule-based and ML-based approaches using Drain3 and IsolationForest algorithms, with automated forensic incident documentation powered by large language models.

Modern approaches to organizational information security have been analyzed and existing SIEM solutions have been examined. A system of functional requirements for the prototype has been defined and the choice of a hybrid threat detection approach has been justified. A modular eight-component system architecture with asynchronous communication has been designed. A two-stage log normalization pipeline based on Grok and Drain3, a detection engine with 13 YAML rules and IsolationForest, a correlation engine with five kill chain sequences, and a forensic investigation module with Claude API integration have been developed. Comparative testing results are presented: the hybrid approach detected 2.6 times more attacks than the rule-based approach (Recall 0.358 vs 0.137), F1-score increased by 97%, AUC-ROC reached 0.863.

Keywords: SIEM, INFORMATION SECURITY, ANOMALY DETECTION, MACHINE LEARNING, ISOLATIONFOREST, DRAIN3, CYBER THREATS, FORENSIC ANALYSIS, LARGE LANGUAGE MODELS, MITRE ATT&CK.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10
1.1 Аналіз підходів до забезпечення інформаційної безпеки організації 10
1.2 Аналіз функціонування існуючих SIEM-систем 13
1.3 Аналіз вимог до розробки SIEM-систем 20
1.4 Постановка технічого завдання 25
Висновки до розділу 26
РОЗДІЛ 2 ПРОЕКТУВАННЯ SIEM-СИСТЕМИ 28
2.1 Проектування загальної архітектури системи 28
2.2 Проектування підсистеми збору та обробки даних 31
2.3 Проектування аналітичного ядра 38
2.4 Проектування інтерфейсної підсистеми 48
Висновки до розділу 55
РОЗДІЛ 3 НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ SIEM-СИСТЕМИ 56
3.1 Розгортання та налаштування 56
3.2 Перевірка функціоналу 59
3.3 Оцінка ефективності 72
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 78
ДОДАТКИ 81

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

ІБ – інформаційна безпека

ОС – операційна система

ПЗ – програмне забезпечення

API – Application Programming Interface (інтерфейс програмування застосунків)

CIA – Confidentiality, Integrity, Availability (конфіденційність, цілісність, доступність)

CSV – Comma-Separated Values (значення, розділені комами)

IDS – Intrusion Detection System (система виявлення вторгнень)

IOC – Indicator of Compromise (індикатор компрометації)

IP – Internet Protocol (інтернет-протокол)

LLM – Large Language Model (велика мовна модель)

ML – Machine Learning (машинне навчання)

ORM – Object-Relational Mapping (об'єктно-реляційне відображення)

PDF – Portable Document Format (портативний формат документів)

REST – Representational State Transfer (архітектурний стиль веб-сервісів)

SIEM – Security Information and Event Management (система управління інформацією та подіями безпеки)

SOC – Security Operations Center (центр операцій з безпеки)

SSE – Server-Sent Events (події, надіслані сервером)

TTPs – Tactics, Techniques and Procedures (тактики, техніки та процедури)

UI – User Interface (користувацький інтерфейс)

URL – Uniform Resource Locator (уніфікований локатор ресурсів)

UTC – Coordinated Universal Time (всесвітній координований час)

YAML – Yet Another Markup Language (мова розмітки даних)

ВСТУП

Актуальність теми. Стрімкий розвиток цифрової інфраструктури організацій супроводжується пропорційним зростанням кіберзагроз. Існуючі комерційні SIEM-рішення є фінансово недоступними для більшості організацій, тоді як безкоштовні альтернативи мають обмежену ML-функціональність та відсутність інструментів автоматизованого документування інцидентів. Актуальність роботи визначається потребою у доступному дослідницькому прототипі SIEM-системи, що поєднує rule-based та ML-based підходи до детекції загроз з автоматизованим forensic-документуванням засобами великих мовних моделей.

Метою кваліфікаційної роботи є проектування та реалізація прототипу SIEM-системи з гібридним детекційним ядром, що поєднує rule-based та ML-based підходи на базі алгоритмів Drain3 та IsolationForest, із забезпеченням автоматизованого forensic-документування інцидентів засобами великих мовних моделей (LLM).

Визначена мета обумовлює наступні завдання:

1. Дослідити та систематизувати сучасні підходи до детекції кіберзагроз у SIEM-системах та провести порівняльний аналіз існуючих рішень.
2. Спроектувати модульну архітектуру прототипу SIEM-системи на мові Python із забезпеченням розширюваності та ізоляції компонентів.
3. Реалізувати гібридне детекційне ядро: rule-based engine (YAML-правила) та ML-based anomaly detector (Drain3 + IsolationForest).
4. Розробити механізм кореляції подій у часовому вікні для виявлення багатоетапних атак типу kill chain.
5. Реалізувати веб-інтерфейс моніторингу з підтримкою real-time оновлення та прогнозування загроз.
6. Розробити модуль forensic-розслідування з AI-генерацією PDF-звітів на базі Claude API.
7. Провести кількісну оцінку якості детекції та порівняти ефективність rule-based та гібридного підходів.

Об'єктом дослідження є процеси виявлення кіберзагроз на основі аналізу потоків log-файлів операційних систем, веб-серверів та мережевого обладнання в реальному часі.

Предметом дослідження є методи та алгоритми гібридної аномалійної детекції в потоках системних логів із застосуванням алгоритму template mining Drain3 та ансамблевого методу IsolationForest.

У процесі виконання кваліфікаційної роботи розроблено прототип SIEM-системи PySIEM — програмний комплекс для централізованого збору, нормалізації, аналізу та кореляції подій безпеки з автоматизованим виявленням аномалій та forensic-документуванням інцидентів засобами штучного інтелекту.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз підходів до забезпечення інформаційної безпеки організації

Стрімкий розвиток цифрової інфраструктури організацій супроводжується пропорційним зростанням кіберзагроз. За даними щорічного звіту IBM Cost of a Data Breach Report 2023, середня вартість витоку даних у світі досягла 4,45 млн доларів США рекордний показник за всі роки досліджень [1]. Кількість зафіксованих кібератак зросла на 38% у 2022 році порівняно з попереднім роком, а середній час виявлення зловмисника в інфраструктурі жертви становить 197 днів [2]. В таких умовах забезпечення інформаційної безпеки (ІБ) перетворюється з суто технічної задачі на стратегічний пріоритет організації.

Інформаційна безпека організації визначається як стан захищеності інформаційних активів, який забезпечує збереження їх ключових властивостей. Відповідно до міжнародного стандарту ISO/IEC 27000:2018, фундаментальною моделлю ІБ є тріада CIA (Confidentiality, Integrity, Availability) — конфіденційність, цілісність та доступність [3].



Рисунок 1.1 – Тріада CIA

Порушення будь-якої складової тріади CIA спричиняє збитки організації, фінансові, репутаційні або операційні. Сучасна практика ІБ розширює класичну тріаду додатковими властивостями: автентичністю (Authenticity), невідмовністю

(Non-repudiation) та підзвітністю (Accountability), що особливо актуально в контексті розслідування кіберінцидентів [4].

Для систематичного забезпечення ІБ організації застосовують концептуальні моделі та фреймворки.Знизу наведено найбільш поширені:

Фреймворк NIST Cybersecurity Framework (CSF) 2.0 — розроблений Національним інститутом стандартів і технологій США, визначає шість функціональних доменів: Govern (управління), Identify (ідентифікація), Protect (захист), Detect (виявлення), Respond (реагування), Recover (відновлення) [5]. SIEM-системи є центральним інструментом доменів Detect та Respond. Фреймворк широко застосовується американськими організаціями та набуває поширення у світі.

Стандарт ISO/IEC 27001:2022 — міжнародний стандарт, що визначає вимоги до систем менеджменту інформаційної безпеки (ISMS Information Security Management System) [6]. Стандарт базується на процесному підході та циклі PDCA (Plan-Do-Check-Act). Додаток А стандарту містить 93 заходи безпеки, згруповані у чотири категорії: організаційні, людські, фізичні та технологічні.

Модель Zero Trust — концепція, що базується на принципі "не довіряй нікому, перевіряй все" (Never Trust, Always Verify). На відміну від традиційного периметрального захисту, Zero Trust передбачає постійну автентифікацію та авторизацію кожного запиту незалежно від його місця походження [7]. Впровадження Zero Trust генерує великі обсяги телеметричних даних, ефективна обробка яких вимагає застосування SIEM-рішень.

Концепція Defense in Depth — стратегія багатошарового захисту, що передбачає застосування декількох незалежних захисних механізмів. Навіть якщо зловмисник подолає один шар захисту, він зіткнеться з наступним. SIEM-системи у цій концепції є компонентом шару виявлення та реагування на загрози.

Сучасні підходи до забезпечення ІБ класифікують за часовим горизонтом реагування на три категорії:

1. Реактивний підхід орієнтований на реагування після факту інциденту. Включає forensic-аналіз, відновлення компрометованих систем, усунення

вразливостей та вдосконалення захисних заходів на основі отриманого досвіду. Digital forensics — науковий напрямок, що займається збором та аналізом цифрових доказів для розслідування кіберінцидентів. Незважаючи на очевидне обмеження (реагування вже після завданої шкоди), реактивний підхід залишається невід'ємною складовою будь-якої стратегії ІБ.

2. Проактивний підхід спрямований на виявлення загроз до їх матеріалізації або на ранніх стадіях розвитку атаки. Реалізується через: Threat Intelligence (збір та аналіз даних про актуальні загрози), Penetration Testing (симульовані атаки на власну інфраструктуру), Vulnerability Management (систематичне виявлення та усунення вразливостей) та Security Monitoring (безперервний моніторинг подій). Саме до цього класу належать SIEM-системи, що забезпечують проактивне виявлення аномалій та атак у реальному часі.

3. Predictive підхід використовує методи машинного навчання та статистичного аналізу для прогнозування майбутніх загроз на основі поточних патернів поведінки. Є найсучаснішим напрямком розвитку галузі та знаходиться у стані активного дослідження. Запропонований у даній роботі прототип SIEM-системи реалізує елементи predictive підходу через прогнозування обсягу подій та ймовірності атак на горизонті 24 годин.

Ключовою організаційною структурою, що реалізує проактивний підхід до ІБ, є Security Operations Center (SOC) — підрозділ, відповідальний за безперервний (24/7) моніторинг, виявлення та реагування на кіберзагрози [8]. Ефективність SOC визначається трьома факторами: людським капіталом (кваліфікованими аналітиками), процесами (методологіями та плейбуками) та технологіями (інструментами моніторингу та аналізу). SIEM-система є центральним технологічним інструментом SOC.

Особливої уваги заслуговує фреймворк MITRE ATT&CK (Adversarial Tactics, Techniques and Common Knowledge) — структурована база знань про тактики, техніки та процедури (TTPs) реальних кіберзловмисників [9]. Фреймворк організований у вигляді матриці, де рядки представляють тактики (загальні цілі

зловмисника), а стовпці техніки досягнення цих цілей. У табл. 1.1 наведено приклади технік MITRE ATT&CK.

Таблиця 1.1

Приклади технік MITRE ATT&CK

Тактика	**ID техніки**	**Назва техніки**	**Опис**
Initial Access	T1078	Valid Accounts	Використання легітимних облікових даних для входу
Credential Access	T1110.001	Password Guessing	Перебір паролів методом brute-force або dictionary attack
Privilege Escalation	T1548.003	Sudo and Sudo Caching	Підвищення привілеїв через зловживання sudo
Lateral Movement	T1021.004	SSH	Переміщення між хостами через SSH-з'єднання
Persistence	T1543.003	Windows Service	Встановлення шкідливого сервісу для закріплення
Discovery	T1046	Network Service Scan	Сканування мережі для виявлення активних сервісів

Таким чином, аналіз підходів до забезпечення ІБ організації демонструє, що SIEM-системи займають центральне місце в сучасній архітектурі кіберзахисту. Вони реалізують проактивний підхід до виявлення загроз, інтегруються з провідними фреймворками (NIST CSF, MITRE ATT&CK) та є невід'ємним компонентом SOC будь-якого масштабу.

1.2 Аналіз функціонування існуючих SIEM-систем

SIEM-системи (Security Information and Event Management) — клас програмних рішень, що забезпечують централізований збір, зберігання, кореляцію та аналіз подій безпеки з різнорідних джерел інфраструктури організації. Акронім SIEM об'єднує дві концепції: SIM (Security Information Management) — управління

інформацією безпеки та довгострокове зберігання логів, та SEM (Security Event Management) — реальночасовий моніторинг та кореляція подій. Термін було введено аналітиками Gartner у 2005 році [10].

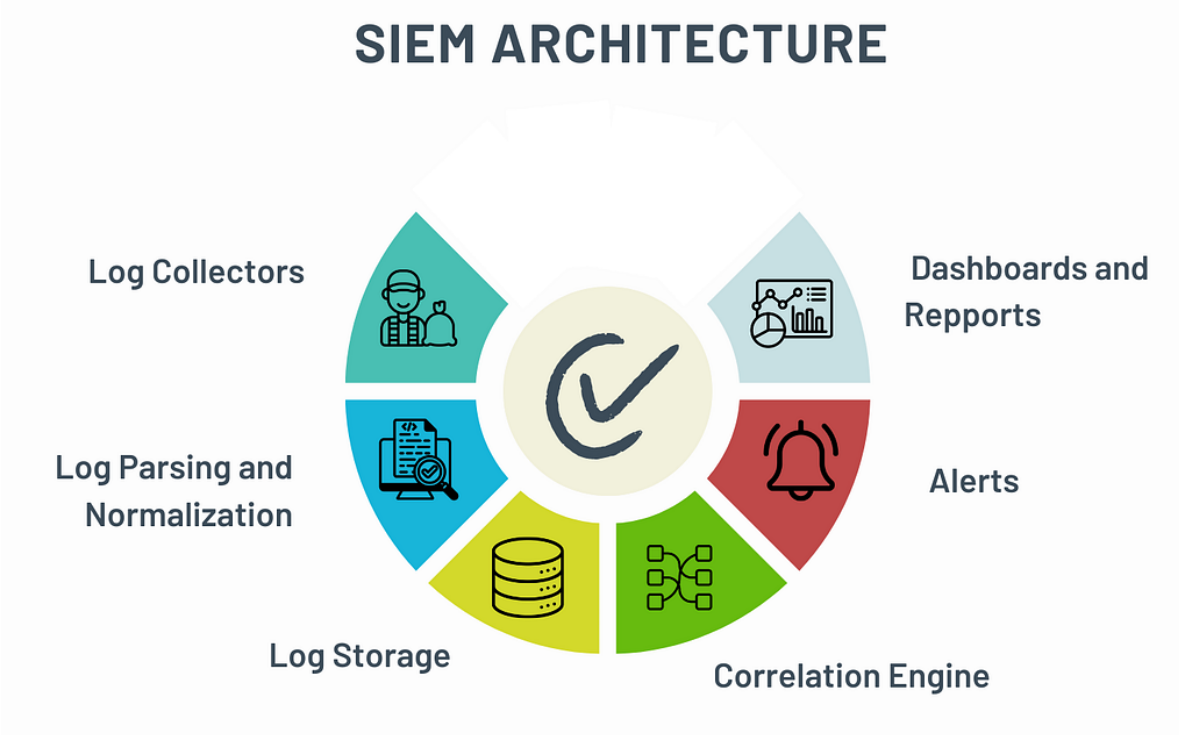


Рисунок 1.2 – Архітектура SIEM

Функціональна архітектура SIEM-системи включає кілька ключових підсистем, які взаємодіють у єдиному pipeline обробки даних:

1. Модуль збору та нормалізації даних (Log Ingestion & Parsing). Відповідає за отримання логів з різнорідних джерел через різні протоколи (Syslog UDP/TCP, SNMP Trap, WMI, API) та їх нормалізацію до єдиної внутрішньої схеми. Якість парсингу безпосередньо впливає на точність детекції.
2. Сховище подій (Event Storage). Забезпечує довгострокове зберігання нормалізованих подій з підтримкою повнотекстового пошуку та швидкого доступу до даних. Традиційно реалізується на базі Elasticsearch або власних колоночних БД.
3. Кореляційний рушій (Correlation Engine). Є серцем SIEM-системи — аналізує потік подій у реальному часі, виявляючи складні патерни загроз, що розподілені у часі та між різними джерелами. Реалізує sliding window correlation та rule-based detection.

4. Підсистема управління алертами та інцидентами. Забезпечує агрегацію, дедуплікацію та пріоритизацію алертів, управління їх життєвим циклом (відкрито/підтверджено/закрито) та ескалацію до відповідальних осіб.
5. Аналітичний інтерфейс (Dashboard). Надає аналітикам SOC зручний доступ до даних через веб-інтерфейс з підтримкою реального часу, фільтрації, пошуку та побудови звітів.

Центральним викликом для SIEM-систем є обробка різнорідних форматів логів. Кожен постачальник програмного та апаратного забезпечення має власний формат журналювання, що ускладнює уніфікацію та кореляцію даних. У табл. 1.2 наведено аналіз основних форматів логів.

Таблиця 1.2

Порівняльний аналіз форматів логів

Формат логів	**Джерело**	**Приклад рядка**	**Складність парсингу**
Syslog (RFC 5424)	Linux/Unix системи	<34>1 2024-01-15T10:00:00Z host sshd - - Failed password for root	Середня стандартний заголовок, змінне тіло
Windows Event Log (XML)	Windows Server, AD	<Event><EventID> 4625</EventID> ...</Event>	Низька структурований XML
Apache/Nginx CLF	Веб-сервери	192.168.1.1 - - [15/Jan/2024] "GET /admin" 401	Низька фіксований формат
JSON structured logs	Мікросервіси, Docker	{"ts":"2024-01-15","level":"ERROR", "msg":"auth failed"}	Дуже низька машиночитабельний

Продовження таблиці 1.2

Довільний текст	Legacy-системи	Jan 15 10:00:01 FAILED LOGIN attempt #5 user=admin	Висока потребує ML-парсингу

Особливої уваги заслуговує проблема парсингу логів довільного формату — legacy-систем, власних додатків та нестандартних конфігурацій. Традиційний підхід базується на написанні Grok-виразів (регулярних виразів) вручну для кожного нового формату. Проте це потребує значних зусиль та погано масштабується. Альтернативою є застосування алгоритму Drain3 для unsupervised log parsing, що автоматично виявляє шаблони у потоках логів [11]. Саме цей підхід обрано в якості основи детекційного ядра розробленого прототипу.

На ринку SIEM-рішень присутні як комерційні платформи, так і рішення з відкритим вихідним кодом. Проведено детальний порівняльний аналіз восьми систем, результати якого наведено в табл. 1.3.

Таблиця 1.3

Порівняльний аналіз існуючих SIEM-систем

Система	**Відкр. код**	**ML-детекція**	**Хмарна підтримка**	**Вартість**	**Ліцензія**
Splunk Enterprise	Ні	Так	Так	>150K$/рік	Комерційна
IBM QRadar	Ні	Частково	Так	>100K$/рік	Комерційна
MS Sentinel	Ні	Так	Azure only	Pay-per-use	Комерційна
Elastic SIEM	Частково	Частково	Так	Freemium	SSPL/ELv2

Продовження таблиці 1.3

Wazuh	Так	Частково	Так	Безкошт.	Apache 2.0
Graylog	Частково	Ні	Так	Freemium	SSPL
AlienVault OSSIM	Так	Ні	Ні	Безкошт.	GPLv2
PySIEM (розробл.)	Так	Так (Drain3+IF)	Так	Безкошт.	Відкрита

Splunk Enterprise є беззаперечним лідером ринку комерційних SIEM-рішень. Система забезпечує обробку петабайтів даних на добу, підтримує понад 2800 додатків та інтеграцій та надає потужні можливості для побудови аналітики через власну мову пошукових запитів SPL (Search Processing Language). Машинне навчання реалізовано через Splunk Machine Learning Toolkit та MLTK. Проте висока вартість ліцензування (від 150 000 USD на рік для середньої організації) робить Splunk недоступним для переважної більшості компаній [12].

IBM QRadar — ентерпрайз-рішення з розвиненими можливостями кореляції та вбудованим Threat Intelligence. QRadar використовує власну мову запитів AQL (Ariel Query Language) та підтримує розгортання як on-premise, так і в хмарі (QRadar on Cloud). Для ML-детекції застосовується модуль User Behavior Analytics (UBA). Основними недоліками є складність налаштування (вимагає сертифікованих спеціалістів) та висока вартість консалтингу з впровадження.

Microsoft Sentinel — хмарна SIEM+SOAR платформа, нативно інтегрована з екосистемою Microsoft Azure. Відрізняється підтримкою мови KQL (Kusto Query Language), автоматизацією реагування через Azure Logic Apps та вбудованими ML-моделями для поведінкового аналізу. Ключовим обмеженням є прив'язка до інфраструктури Azure для організацій, що використовують мультихмарний підхід, це може бути критичним недоліком.

Elastic SIEM (компонент Elastic Security) базується на технологічному стеку ELK (Elasticsearch, Logstash, Kibana). Переваги: висока продуктивність, підтримка

Detection Rules у форматі Sigma та наявність безкоштовного рівня. Машинне навчання для anomaly detection доступне лише у платних тарифних планах (Platinum та вище). Зміна ліцензії з Apache 2.0 на SSPL/ELv2 у 2021 році ускладнила використання у певних сценаріях.

Wazuh є найбільш функціональним рішенням з відкритим вихідним кодом. Базується на OSSEC та розширює його можливостями threat detection, File Integrity Monitoring (FIM), compliance monitoring та vulnerability assessment. Підтримує правила у форматі XML та інтеграцію з MITRE ATT&CK. ML-детекція реалізована частково через статистичний аналіз аномалій. Потребує значних ресурсів для розгортання та підтримки (рекомендований кластер Elasticsearch з 3+ нод).

AlienVault OSSIM — найстаріше рішення з відкритим кодом, що надає базові SIEM-можливості: збір логів, кореляцію, управління вразливостями. Суттєво поступається сучасним аналогам за функціональністю та масштабованістю. Розробник (AT&T Cybersecurity) переорієнтувався на комерційний продукт USM Anywhere.

Аналіз розглянутих систем дозволяє виявити спільні обмеження, що обумовлюють актуальність даного дослідження:

- **Недостатня інтеграція методів unsupervised ML для anomaly detection.** Більшість розглянутих систем реалізують переважно rule-based підхід або надають ML-функціональність лише у преміум-тарифах (Elastic Platinum, Splunk MLTK). Фундаментальна проблема rule-based підходу полягає у тому, що він виявляє виключно відомі атаки, для яких написані правила. За даними дослідження Verizon DBIR 2023, 74% успішних зламів використовують нові або модифіковані техніки, що не охоплюються існуючими сигнатурами [19]. Таким чином, SIEM-система без ML-компоненту апріорі не здатна виявити значну частину реальних загроз.
- **Відсутність пояснюваності виявлених аномалій (Explainable AI).** Аналітик SOC отримує алерт «аномальна активність» без пояснення, які саме ознаки спричинили спрацювання. Це призводить

до двох проблем: по-перше, аналітику доводиться витрачати значний час на ручне дослідження контексту події; по-друге, без пояснення неможливо відрізнити справжню загрозу від хибного спрацювання. Дослідження показують, що аналітики SOC витрачають до 40% робочого часу на обробку хибних спрацювань [20]. Інтеграція LLM для генерації природномовних пояснень аномалій є перспективним напрямком вирішення цієї проблеми.

- **Висока складність налаштування та впровадження.** Навіть рішення з відкритим кодом (Wazuh, Elastic SIEM) потребують значних зусиль для розгортання: конфігурування кластеру Elasticsearch, написання Grok-виразів для кожного нового формату логів, налаштування правил кореляції. За оцінками Gartner, організації витрачають від 6 до 18 місяців на повноцінне впровадження SIEM-рішення [10]. Висока складність призводить до того, що малі та середні організації або відмовляються від SIEM взагалі, або використовують лише базовий функціонал.
- **Відсутність вбудованих інструментів forensic-документування.** Жодне з розглянутих open-source рішень не надає вбудованого workflow для документування розслідувань інцидентів безпосередньо в інтерфейсі SIEM. Аналітики змушені використовувати окремі інструменти (Jira, Confluence, Word) для ведення нотаток, збору доказів та генерації звітів. Це порушує єдність контексту розслідування та збільшує час реагування на інцидент (MTTR Mean Time To Respond). Автоматизована генерація forensic-звітів засобами LLM може скоротити час документування з годин до хвилин.
- **Слабка підтримка прогнозування загроз.** Існуючі SIEM-системи функціонують переважно у реактивному режимі — виявляють загрози, що вже відбуваються або відбулись. Predictive analytics, що дозволяє прогнозувати ймовірність атак на основі поточних патернів

поведінки (time-series forecasting, seasonal decomposition), практично відсутня у більшості рішень. Проактивне сповіщення аналітиків про підвищений ризик у наступні години дозволило б суттєво скоротити час між початком атаки та її виявленням (MTTD Mean Time To Detect).

Зазначені обмеження обумовлюють наукову та практичну актуальність розробки дослідницького прототипу SIEM-системи з гібридним детекційним ядром, що поєднує rule-based та ML-based підходи, доповнені можливостями пояснюваного AI та автоматизованого forensic-документування.

1.3 Аналіз вимог до розробки SIEM-систем

Проектування SIEM-системи потребує чіткого визначення вимог, що враховують як загальноприйняті стандарти галузі, так і специфіку конкретного застосування. Вимоги формуються на основі аналізу існуючих рішень (розд. 1.2), міжнародних стандартів (ISO/IEC 27001, NIST CSF) та академічних досліджень у галузі anomaly detection.

Одним з ключових технічних викликів при розробці SIEM-систем є вибір методу детекції аномалій. Кожен підхід має власні характеристики за метриками Precision та Recall, що суттєво впливає на операційну ефективність системи. У табл. 1.4 наведено порівняльний аналіз основних методів.

Таблиця 1.4

Порівняльний аналіз методів детекції аномалій для SIEM-систем

Метод	**Precision**	**Recall**	**Застосування у SIEM**
Rule-based (сигнатури)	Висока (>0.95)	Низький (відомі атаки)	Детекція відомих атак: brute-force, SQL-ін'єкція
Statistical threshold	Середня	Середній	Аномалії об'єму трафіку, кількості запитів
IsolationForest (ML)	Середня (0.7–0.9)	Вищий (невід. загрози)	Unsupervised anomaly detection у log-потоках

Продовження таблиці 1.4

LSTM / Deep Learning	Висока	Високий	Sequence-based anomaly detection, потребує GPU
Гібридний (rule+ML)	Висока (>0.88)	Вищий (0.85+)	Оптимальний баланс для прототипу SIEM

Rule-based детекція забезпечує високу Precision (мінімум хибних спрацювань) для відомих атак, проте має суттєве обмеження за Recall: виявляє лише ті загрози, для яких написані правила. Нові, невідомі атаки (zero-day) та складні багатоетапні кампанії можуть залишатись непоміченими. Це є фундаментальним обмеженням сигнатурного підходу [13].

Алгоритм IsolationForest (Isolation Forest), запропонований Liu et al. у 2008 році, є одним з найефективніших unsupervised алгоритмів для виявлення аномалій [14]. Принцип роботи базується на властивості аномальних точок: вони ізолюються значно швидше (за менше кроків бінарного розбиття простору ознак), ніж нормальні спостереження. Алгоритм не потребує нормального розподілу даних та ефективно масштабується для великих наборів даних. В контексті аналізу логів ознаки включають: час доби, день тижня, довжину повідомлення, кількість токенів, ідентифікатор шаблону Drain3 та поведінкові характеристики користувача.

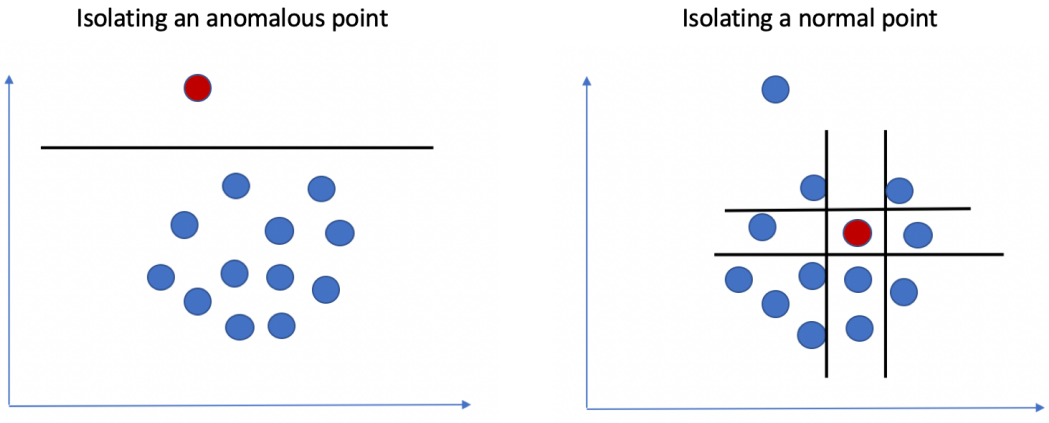


Рисунок 1.4 - Алгоритм IsolationForest

Алгоритм Drain3 (Deep Learning Algorithm for Log Parsing) — метод online log parsing, що автоматично виявляє шаблони (templates) у потоках текстових логів [11]. Алгоритм організовує шаблони у вигляді дерева глибини D, де внутрішні

вузли розбивають повідомлення за довжиною та першим токеном, а листові вузли містять групи схожих повідомлень. Drain3 відрізняється від оригінального Drain підтримкою персистентності стану між перезапусками, що критично для продуктивного застосування. Threshold подібності sim_th (за замовчуванням 0.5) визначає, наскільки схожими мають бути два повідомлення для об'єднання в один шаблон.

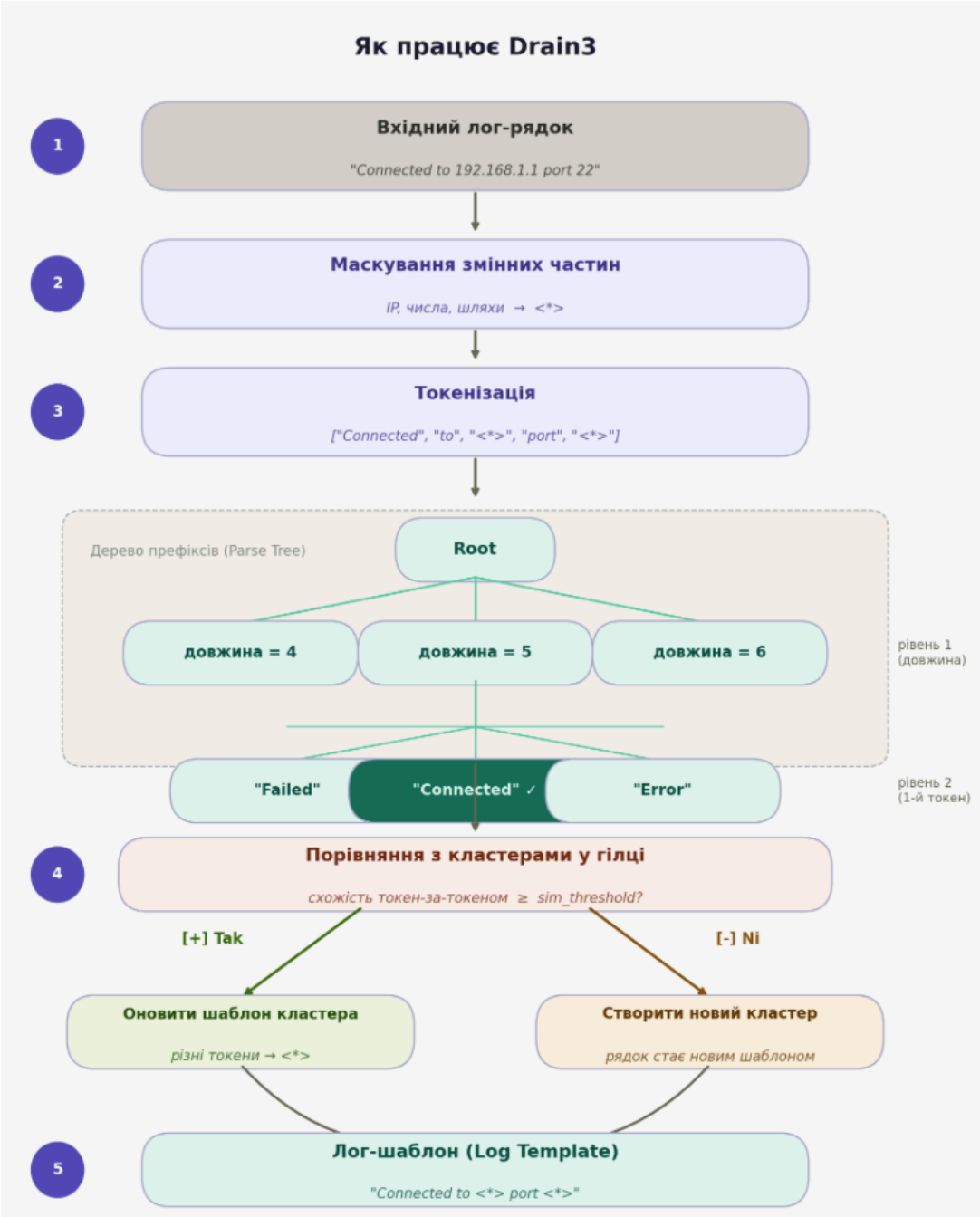


Рисунок 1.5 – Робота алгоритму Drain3

Гібридний підхід, що поєднує rule-based та ML-based детекцію, забезпечує найкращий баланс між Precision та Recall. Правила детектують відомі атаки з

мінімальними хибними спрацюваннями, тоді як ML-компонент виявляє статистично аномальні події, які не підпадають під жодне правило. Саме цей підхід обрано для реалізації у прототипі PySIEM. Теоретичне обґрунтування ефективності гібридного підходу підтверджується результатами попередніх досліджень [15], де показано зростання F1-метрики на 15-25% порівняно з чисто rule-based рішеннями.

Важливим аспектом проектування SIEM є вибір механізму кореляції подій. Кореляція дозволяє виявляти складні багатоетапні атаки (kill chain), що складаються з окремих подій, кожна з яких сама по собі не є підозрілою. Для прототипу обрано підхід sliding window correlation, що базується на механізмі ковзного часового вікна: система відстежує послідовності подій у визначеному інтервалі часу для визначеного набору атрибутів (IP-адреса, ім'я користувача тощо).

Нормативно-правова база для розробки SIEM-систем формується міжнародними та національними стандартами. Стандарт NIST SP 800-92 «Guide to Computer Security Log Management» [16] визначає рекомендації щодо організації збору, зберігання та аналізу логів безпеки. Відповідно до стандарту, організації мають зберігати логи щонайменше 90 днів в онлайн-режимі та 1 рік в архіві. Стандарт NIST SP 800-137 «Information Security Continuous Monitoring» [17] визначає вимоги до безперервного моніторингу інформаційної безпеки, на основі яких формується функціональний мінімум SIEM-рішення.

На підставі проведеного аналізу сформовано систему вимог до розроблюваного прототипу. Вимоги систематизовано за функціональними категоріями у табл. 1.5 та класифіковано за пріоритетом: обов'язкові (MVP), бажані та додаткові.

Таблиця 1.5

Система вимог до прототипу SIEM-системи PySIEM

№	Категорія	Вимога	Пріоритет
1	Збір даних	Підтримка Syslog UDP/TCP, file tail, HTTP webhook, Docker-контейнерів	Обов'язкова

Продовження таблиці 1.5

2	Парсинг	Нормалізація до єдиної схеми (ECS-подібна); підтримка Grok + Drain3	Обов'язкова
3	Детекція	Rule-based (YAML-правила, Sigma-подібний формат) з підтримкою threshold та sequence	Обов'язкова
4	ML-детекція	Unsupervised anomaly detection (IsolationForest) з автоматичним перенавчанням	Обов'язкова
5	Кореляція	Sliding window correlation для виявлення багатоетапних атак (kill chain)	Обов'язкова
6	Risk scoring	Оцінка ризику алерту на основі severity, anomaly score, asset criticality	Обов'язкова
7	Інтерфейс	Веб-дашборд з real-time оновленням, фільтрацією та сортуванням подій	Обов'язкова
8	Forensic	Модуль розслідування інцидентів з управлінням evidence та коментарями	Бажана
9	Звітність	AI-генерація forensic PDF-звітів через LLM API з можливістю редагування	Бажана
10	Прогнозування	ML-прогноз обсягу подій та ймовірності атак на наступні 24 години	Додаткова
11	MITRE ATT&CK	Автоматичне маппування алертів до технік MITRE ATT&CK	Бажана
12	GeoIP	Збагачення подій геолокаційними даними та ASN для зовнішніх IP-адрес	Додаткова

Обов'язкові вимоги формують мінімальний функціональний набір (MVP Minimum Viable Product), без якого система не може вважатись SIEM-рішенням. Збір логів з різнорідних джерел є базовою вимогою, без консолідації даних неможлива кореляція між різними компонентами інфраструктури. Нормалізація до єдиної схеми забезпечує семантичну уніфікацію полів: незалежно від джерела, поле «IP-адреса атакуючого» завжди матиме однакову назву та формат у нормалізованій події.

Бажані вимоги суттєво підвищують цінність системи та відрізняють PySIEM від простих агрегаторів логів. Модуль forensic-розслідування реалізує повний work

flow: від виявлення алерту до генерації PDF-звіту для керівництва функціональність, що відсутня у більшості open-source рішень. Інтеграція MITRE ATT&CK підвищує аналітичну цінність алертів, надаючи тактичний контекст кожній виявленій загрозі.

Нефункціональні вимоги включають: продуктивність (обробка не менше 500 подій/с в режимі реального часу та 5000 подій/с в batch-режимі), надійність (збереження всіх подій навіть при збоях детекційного ядра через ізоляцію компонентів), зручність використання (веб-інтерфейс без встановлення додаткового ПЗ на стороні аналітика) та розширюваність (модульна архітектура, що дозволяє додавати нові джерела та алгоритми детекції без модифікації ядра системи).

Вимоги до метрик якості детекції сформовано на основі аналізу академічних досліджень у галузі anomaly detection для системних логів. Цільові значення Precision $\geq$ 0.85 та Recall $\geq$ 0.75 відповідають прийнятному рівню для дослідницького прототипу. Для порівняння, продуктивні SIEM-системи зазвичай налаштовуються на Precision > 0.95 за рахунок нижчого Recall, що мінімізує навантаження на аналітиків SOC від хибних спрацювань [18].

1.4 Постановка технічого завдання

На підставі проведеного аналізу предметної області (розд. 1.1), огляду існуючих рішень (розд. 1.2) та визначених вимог (розд. 1.3) сформовано технічне завдання на розробку прототипу SIEM-системи PySIEM з AI-driven детекційним ядром. Параметри технічного завдання наведено в табл. 1.6.

Таблиця 1.6

Технічне завдання на розробку прототипу SIEM-системи PySIEM

Параметр	**Значення / опис**
Назва системи	PySIEM прототип SIEM-системи з AI-driven детекцією на базі Drain3
Мова реалізації	Python 3.11+ (основна логіка), JavaScript (веб-інтерфейс)

Продовження таблиці 1.6

Log parsing	Drain3 (алгоритм template mining для структурування довільних логів)
Anomaly detection	sklearn IsolationForest з time-aware features (sin/cos encoding, user behavioral profile)
Rule engine	YAML-правила з підтримкою threshold, sequence та group_by умов
Кореляція	Sliding window correlator на defaultdict + deque; підтримка kill chain sequences
Сховище даних	SQLite (розробка/тестування) з міграцією на PostgreSQL (продуктивне середовище)
Веб-фреймворк	FastAPI + Uvicorn; фронтенд: Chart.js, HTMX, TailwindCSS (CDN)
AI-компонент	Anthropic Claude API (claude-sonnet-4-6): пояснення алертів, forensic звіти, прогнози
Джерела логів	Linux syslog, Windows Event Log (симульований), Apache access/error log, Docker webhook
Розгортання	Docker Compose (multi-container: siem + windows-sim + apache-sim + linux-attacker)
Цільові метрики	Precision ≥ 0.85, Recall ≥ 0.75, F1 ≥ 0.80 на синтетичному датасеті
Обсяг датасету	≥ 1000 подій (≈77% нормальних, ≈23% атак) з ground truth мітками

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної області інформаційної безпеки, досліджено існуючі SIEM-рішення та сформовано технічне завдання на розробку прототипу PySIEM.Встановлено, що SIEM-системи займають центральне місце в сучасній архітектурі кіберзахисту та органічно інтегруються у провідні фреймворки NIST CSF 2.0 і MITRE ATT&CK. Ефективна стратегія інформаційної безпеки має поєднувати реактивний, проактивний та predictive підходи, однак більшість існуючих рішень орієнтовані переважно на перші два.

Порівняльний аналіз восьми SIEM-систем виявив принципові обмеження наявних рішень. Комерційні платформи Splunk, QRadar та Microsoft Sentinel забезпечують

розвинену ML-детекцію та високу масштабованість, проте є фінансово недоступними для більшості організацій через високу вартість ліцензування. Рішення з відкритим кодом Wazuh та AlienVault OSSIM усувають проблему вартості, однак мають обмежену ML-функціональність та значну складність розгортання. Спільним недоліком усіх розглянутих систем є відсутність пояснюваності виявлених аномалій, вбудованих інструментів forensic-документування та прогнозування загроз.

Аналіз методів детекції аномалій обґрунтував вибір гібридного підходу, що поєднує rule-based детекцію для відомих атак та unsupervised IsolationForest для виявлення невідомих загроз. Алгоритм Drain3 обрано для автоматичного парсингу логів довільного формату як альтернативу ручному написанню Grok-виразів, що суттєво знижує поріг впровадження системи. На підставі виявлених обмежень існуючих рішень сформовано технічне завдання на розробку прототипу PySIEM з гібридним детекційним ядром, AI-компонентом для пояснення алертів і генерації forensic-звітів та модулем predictive-аналітики.

РОЗДІЛ 2 ПРОЕКТУВАННЯ SIEM-СИСТЕМИ

2.1 Проектування загальної архітектури системи

Проектування архітектури є фундаментальним етапом розробки будь-якої програмної системи. Правильно обрана архітектура визначає масштабованість, надійність та зручність подальшої розробки і підтримки системи. Для прототипу SIEM-системи PySIEM ключовими архітектурними вимогами є: модульність (кожен компонент виконує одну чітко визначену функцію), розширюваність (додавання нових джерел та алгоритмів без модифікації ядра), ізоляція збоїв (відмова одного компоненту не призводить до зупинки системи) та мінімальна складність розгортання.

При проектуванні архітектури розглянуто три основні підходи: монолітна архітектура, мікросервісна архітектура та модульний моноліт. Порівняльний аналіз наведено в табл. 2.1.

Таблиця 2.1

Порівняльний аналіз архітектурних підходів для прототипу SIEM-системи

Критерій	**Моноліт**	**Мікросервіси**	**Модульний моноліт**
Кількість процесів при розгортанні	1	5–10+	1
Мережева латентність між компонентами	0 мс	1–10 мс	0 мс
Кількість технологій в стеку	1	3–5+	1–2
Час розгортання (хв)	<5	30–60	<5

На підставі порівняльного аналізу для прототипу PySIEM обрано підхід модульного моноліту (Modular Monolith). Цей підхід поєднує переваги обох крайнощів: простоту розгортання та низьку латентність монолітної архітектури з чіткою межею відповідальності модулів, характерною для мікросервісів. Комунікація між модулями здійснюється через asyncio.Queue — асинхронну чергу Python, що забезпечує розв'язання виробника і споживача без мережевої затримки.

Важливим архітектурним рішенням є застосування асинхронної моделі виконання на базі asyncio. Це дозволяє паралельно обробляти потоки подій з кількох джерел без блокування головного потоку, що критично для забезпечення low-latency детекції в реальному часі. Кожне джерело логів запускається як окрема asyncio.Task, а нормалізовані події передаються у спільну asyncio.Queue з максимальним розміром 10 000 елементів для запобігання переповненню пам'яті.

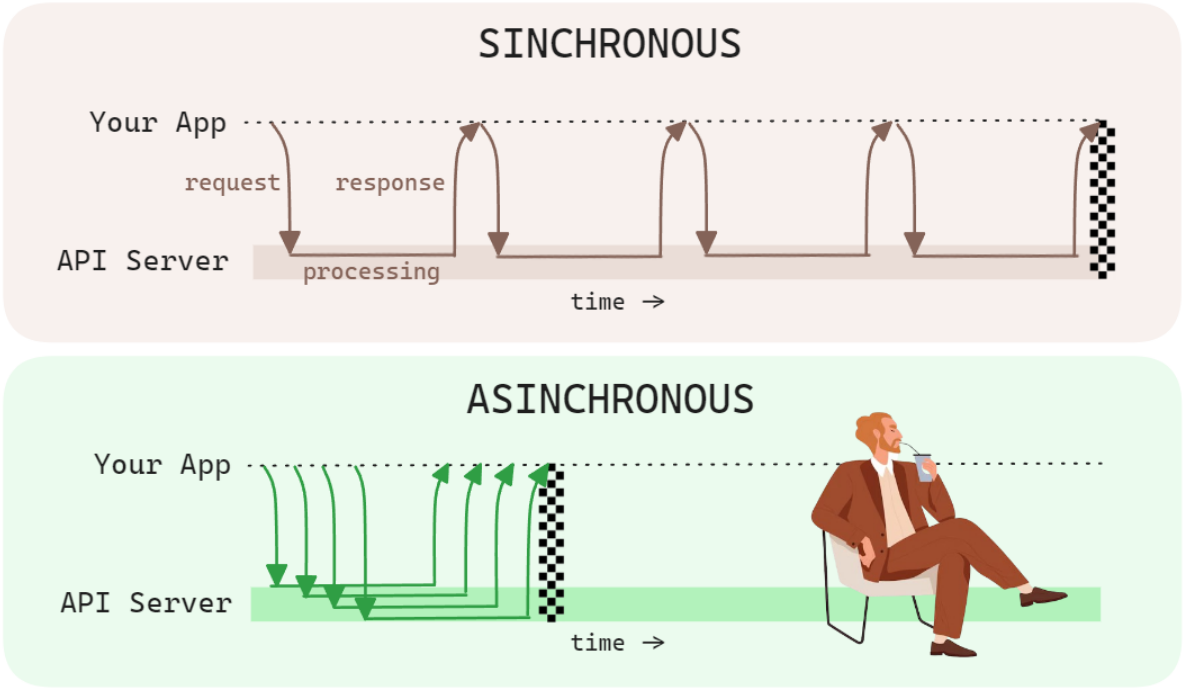


Рисунок 2.1 - Порівняння синхронного та асинхронного методу

Архітектура PySIEM організована у вигляді pipeline обробки даних, де кожен компонент отримує дані від попереднього та передає результат наступному. Система складається з восьми основних модулів:

1. collector.py модуль збору логів. Реалізує підтримку чотирьох транспортних механізмів: Syslog UDP/TCP (socketserver), file tail (watchdog + inotify), HTTP webhook (FastAPI endpoint) та Kafka consumer (aiokafka). Продукує RawLogEntry об'єкти до вхідної черги.

2. parser.py модуль нормалізації. Отримує RawLogEntry, застосовує Grok-pattern matching для відомих форматів та Drain3 template mining для невідомих. Продукує NormalizedEvent об'єкти зі стандартизованими полями та template_id для подальшого ML-аналізу.

3. storage.py рівень зберігання даних. Реалізує ORM-моделі через SQLAlchemy та забезпечує персистентність подій, алертів, інцидентів та розслідувань. Підтримує SQLite (розробка) та PostgreSQL (продуктив) без змін коду завдяки абстракції SQLAlchemy.

4. anomaly_detector.py ML-компонент детекції. Інкапсулює IsolationForest модель sklearn та логіку формування feature vector. Реалізує режими навчання (fit) та інференсу (predict), управління поведінковими профілями користувачів та автоматичне перенавчання.

5. rules_engine.py рушій правил. Завантажує YAML-правила при старті та оцінює кожну нормалізовану подію відносно всіх активних правил. Реалізує threshold та sequence умови через sliding window counter на defaultdict + deque.

6. correlation_engine.py кореляційний рушій. Агрегує алерти від rules engine та ML detector, виявляє kill chain послідовності (brute-force → privilege escalation → lateral movement) та генерує Incident об'єкти.

7. alert_manager.py управління алертами. Забезпечує дедуплікацію (пригнічення повторних алертів), застосування risk scorer та збереження алертів у БД. Опціонально надсилає webhook-сповіщення у зовнішні системи.

8. dashboard/ веб-інтерфейс. FastAPI додаток, що надає REST API для frontend та серверний рендеринг HTML-шаблону. Реалізує SSE (Server-Sent Events) для стримінгу AI-відповідей та інтеграцію з Claude API для пояснення алертів та генерації forensic-звітів.

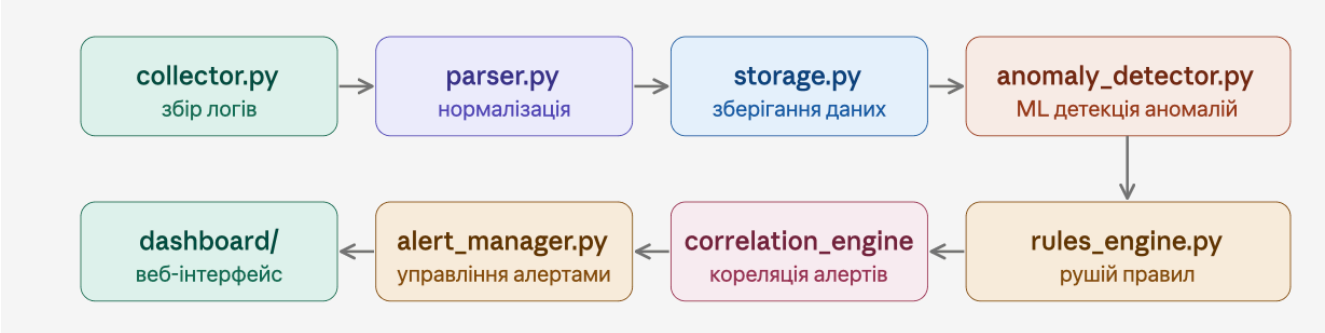


Рисунок 2.2 – Загальна архітектура системи

Потік даних між компонентами організовано наступним чином: collector → parser (через asyncio.Queue) → [storage, anomaly_detector, rules_engine]

(паралельно) → correlation_engine → alert_manager → storage. Цей однонаправлений потік унеможливлює циклічні залежності між компонентами та спрощує відлагодження.

Система розгортається у вигляді Docker Compose конфігурації, що включає чотири контейнери: основний SIEM-процес (python main.py), симулятор Windows Event Log (Python скрипт, що генерує події EventID 4624/4625/4672/4688), симулятор Apache (генерує CLF access log), симулятор Linux атакуючого (генерує brute-force та privilege escalation послідовності). Всі симулятори надсилають логи до основного контейнера через POST /api/ingest/webhook, що дозволяє демонструвати мультиджерельну кореляцію в реальному часі.

Для збереження стану між перезапусками контейнерів використовуються Docker volumes: data/siem.db (SQLite база даних), models/ (серіалізовані ML-моделі у форматі pickle), rules/ (YAML-файли правил, що можна редагувати без перебудови образу).

2.2 Проектування підсистеми збору та обробки даних

2.2.1 Модуль збору та нормалізації логів

Модуль збору та нормалізації логів є першою ланкою pipeline обробки даних у PySIEM. Від якості реалізації цього модуля залежить повнота та точність подальшої детекції, якщо подія не зібрана або нормалізована з помилками, вона не буде виявлена жодним детекційним алгоритмом. Модуль реалізовано у двох файлах: collector.py (збір) та parser.py (нормалізація).

Collector реалізує патерн Producer у архітектурі Producer-Consumer. Кожне джерело логів є незалежним Producer, що асинхронно генерує RawLogEntry об'єкти до спільної asyncio.Queue. Такий підхід забезпечує природний backpressure: якщо черга заповнюється (досягає maxsize=10000), нові записи блокуються до звільнення місця, запобігаючи надмірному споживанню пам'яті.

Підтримувані транспортні механізми наведено в табл. 2.2. Для кожного джерела реалізовано окремий клас-нащадок BaseSource з методом stream(queue: asyncio.Queue), що запускається як asyncio.Task.

Таблиця 2.2

Транспортні механізми збору логів у PySIEM

Транспорт	**Протокол**	**Порт за замовч.**	**Застосування**
Syslog UDP	UDP	514	Linux/Unix системи, мережеве обладнання (Cisco, Juniper)
Syslog TCP	TCP	514 / 601	Надійна доставка з підтвердженням отримання
File tail	inotify / polling	—	Локальні лог-файли: /var/log/auth.log, /var/log/syslog
HTTP webhook	HTTP POST	8083	Симулятори Windows/Apache, Docker-контейнери, CI/CD
Kafka consumer	TCP (Kafka)	9092	High-throughput pipeline для великих обсягів логів

RawLogEntry — dataclass, що інкапсулює мінімально необхідні метадані для подальшого парсингу. Структура об'єкту включає: raw_text (оригінальний рядок логу), source (ідентифікатор джерела: linux/windows/apache), transport (тип транспорту: syslog/file/webhook), received_at (UTC timestamp отримання). Навмисно мінімальна структура гарантує, що collector не виконує жодної семантичної обробки, а виключно транспортну функцію.

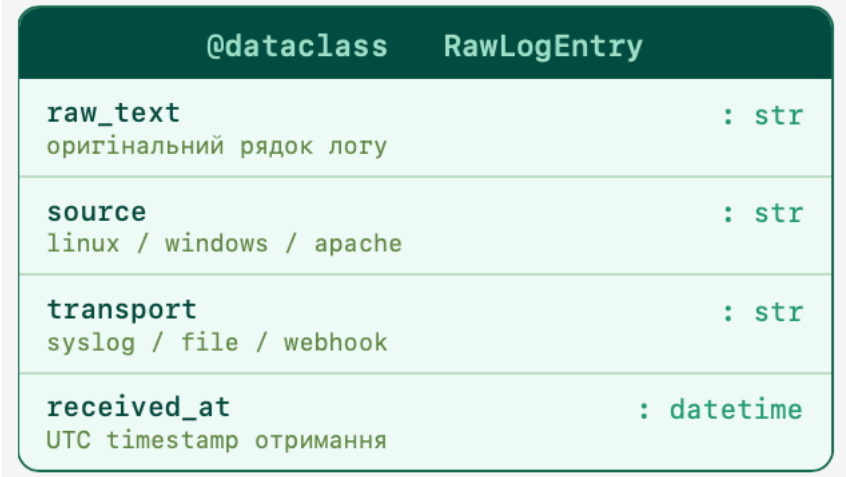


Рисунок 2.3 – Мінімальна структура об’єкту

Для file tail джерела застосовано бібліотеку watchdog, що використовує inotify (Linux) або FSEvents (macOS) для отримання сповіщень від ОС про зміни у файлі. Це значно ефективніше за polling-підхід (перевірку розміру файлу з інтервалом), оскільки не споживає CPU у стані очікування. При старті системи collector зчитує останні 1000 рядків файлу як «холодний старт» для заповнення baseline ML-моделі.

Нормалізація є процесом перетворення різнорідних форматів логів до єдиної внутрішньої схеми NormalizedEvent. Схема розроблена на базі Elastic Common Schema (ECS) з адаптаціями для потреб прототипу. Повний перелік полів схеми наведено в додатку А(таблиця А.1).

Таблиця 2.3

Схема нормалізованої події NormalizedEvent

Поле	**Тип даних**	**Опис**
id	UUID	Унікальний ідентифікатор події
occurred_at	TIMESTAMPTZ	Час події (UTC)
event_type	TEXT	Тип: auth.failure, sudo.exec тощо
severity	SMALLINT	Критичність 0–5
source_ip / dest_ip	TEXT	Мережевий контекст
template_id	TEXT	Хеш шаблону Drain3
anomaly_score	FLOAT	Оцінка IsolationForest [0–1]
raw_message	TEXT	Оригінальний рядок логу

Ключовою особливістю схеми є поле template_id, це хеш шаблону Drain3, що автоматично генерується при парсингу. Це поле є унікальним для підходу PySIEM та відсутнє у стандартних SIEM-схемах. Template_id дозволяє IsolationForest виявляти нові невідомі типи повідомлень: якщо подія містить template_id, що рідко зустрічався у training set, це є ознакою аномалії.

Поле fields типу JSONB (або JSON для SQLite) зберігає додаткові атрибути, специфічні для конкретного джерела. Наприклад, для Windows Event Log: EventID, Channel, провайдер; для Apache : HTTP-метод, URI, HTTP-статус, розмір відповіді. Це забезпечує гнучкість схеми без необхідності додавати нові стовпці до таблиці при кожному новому джерелі.

Parser реалізує стратегію каскадного парсингу: спочатку застосовуються детерміновані Grok-патерни для відомих форматів, і лише якщо жоден не підходить активується Drain3 для невідомих. Такий підхід забезпечує оптимальний баланс: швидкий та точний парсинг для відомих джерел та автоматичну адаптацію до нових.

Реалізовано наступні Grok-патерни для відомих форматів: syslog (RFC 5424), SSH auth.failure/auth.success/invalid_user, sudo (COMMAND, session), su (session.open, session.close, failure), Apache CLF (access log), Apache error log. Бібліотека pygrok дозволяє описувати патерни декларативно з іменованими групами захоплення, що автоматично маппуються до полів NormalizedEvent.

Порівняльний аналіз підходів Grok та Drain3 наведено в табл. 2.4. Обидва підходи застосовуються у парсері PySIEM, Grok для структурованих відомих форматів, Drain3 для довільного тексту.

Таблиця 2.4

Порівняльний аналіз підходів до парсингу логів

Критерій	**Grok (регулярні вирази)**	**Drain3 (ML template mining)**
Необхідність налаштування	Висока (потрібно написати pattern для кожного формату)	Мінімальна (алгоритм навчається автоматично)
Підтримка нових форматів	Потребує ручного додавання нових patterns	Автоматично адаптується до нових форматів
Швидкість парсингу	Висока (regex оптимізовано)	Середня (tree traversal)

Продовження таблиці 2.4

Виявлення шаблонів	Лише заздалегідь визначені	Автоматичне кластерування схожих повідомлень
Template ID для ML	Відсутній	Генерується автоматично(ключова ознака для IsolationForest)
Персистентність стану	Не потребує	Так, шаблони зберігаються між перезапусками
Застосування у PySIEM	Відомі формати (SSH, sudo, Apache)	Невідомі формати + генерація template_id для ML

Алгоритм Drain3 організовує шаблони у вигляді дерева фіксованої глибини D. Кожен вхідний рядок проходить такі кроки обробки: (1) попередня обробка: видалення IP-адрес, чисел, хешів регулярними виразами; (2) вибір кореня дерева за довжиною токенів; (3) вибір внутрішнього вузла за першим токеном; (4) пошук найбільш схожого шаблону серед листових вузлів за метрикою cosine similarity токенних наборів; (5) оновлення шаблону або створення нового. Параметр sim_th (порогове значення схожості, за замовчуванням 0.5) визначає гранулярність кластеризації: нижче значення = менше шаблонів, вище = більше.

Алгоритм Drain3 — кроки обробки рядка

Крок	Дія	Деталі
0	Вхідний рядок	`"Failed password for root from 192.168.1.5 port 22"`
1	Попередня обробка	IP, числа, хеші замінюються на <*> регулярними виразами Результат: `"Failed password for root from <*> port <*>"`
2	Вибір кореня дерева	Корінь обирається за кількістю токенів у рядку Тут: 9 токенів → гілка depth=9
3	Вибір внутрішнього вузла	Внутрішній вузол обирається за першим токеном Тут: `"Failed"` → відповідна гілка дерева
4	Пошук схожого шаблону	Серед листових вузлів обчислюється cosine similarity токенних наборів. Порівнюється з порогом `sim_th = 0.5`
5	Оновлення або створення	Схожість ≥ sim_th → оновити шаблон (різні токени → <*>) Схожість < sim_th → створити новий кластер

Параметр sim_th: низьке значення → менше шаблонів (груба кластеризація) · високе значення → більше шаблонів (точна кластеризація)

Рисунок 2.4 – Кроки обробки рядка

Важливим аспектом реалізації parser.py є обробка часових міток. Логи різних систем використовують різні формати: syslog «Jan 15 10:00:01», Windows Event Log ISO 8601, Apache «15/Jan/2024:10:00:01 +0000». Усі часові мітки нормалізуються до UTC. Для syslog реалізовано спеціальну функцію _parse_syslog_ts(), що вирішує проблему відсутності року у стандартному форматі через порівняння з поточним часом — якщо результат більше ніж на 1 годину у майбутньому, рік зменшується на 1 (обробка переходу рік/рік).

2.2.2 Сховище даних

Сховище даних є центральним компонентом SIEM-системи, що забезпечує персистентність усіх подій, алертів, інцидентів та розслідувань. Правильний вибір технології зберігання та проектування схеми БД суттєво впливають на продуктивність системи та зручність аналітичних запитів.

Для вибору оптимального рішення для зберігання даних проведено порівняльний аналіз трьох альтернатив: SQLite, PostgreSQL та OpenSearch.

Результати аналізу наведено в табл. 2.5.

Таблиця 2.5

Порівняльний аналіз технологій зберігання даних для SIEM

Критерій	**SQLite**	**PostgreSQL**	**OpenSearch / Elasticsearch**
Складність розгортання	Мінімальна (файл)	Середня (сервер)	Висока (кластер)
Продуктивність запитів	До 1M подій/добу	До 100M подій/добу	Мільярди подій
Full-text search	FTS5 (обмежено)	pg_trgm, tsvector	Нативний (Lucene)
JSONB підтримка	JSON (без індексів)	JSONB + GIN індекси	Нативний (dynamic mapping)
Транзакції ACID	Так	Так	Ні (eventual consistency)

Продовження таблиці 2.5

Вартість	Безкоштовно	Безкоштовно	Безкоштовно (OSS)
Придатність для прототипу	Оптимально	Рекомендовано для продуктиву	Надмірно складно

На підставі порівняльного аналізу прийнято рішення використовувати SQLite як основне сховище для прототипу з підтримкою міграції на PostgreSQL для продуктивного середовища. SQLite забезпечує нульову складність розгортання (файл-база даних без необхідності окремого серверного процесу), підтримує ACID-транзакції та надає достатню продуктивність для обсягів прототипу. Абстракція SQLAlchemy ORM дозволяє перемикатись між SQLite та PostgreSQL зміною одного рядка конфігурації (connection string) без модифікації коду запитів.

OpenSearch відхилено через надмірну складність розгортання для прототипу: мінімальна конфігурація потребує 3 вузли та 6+ GB RAM, що неприйнятно для дослідницького проєкту. Крім того, відсутність повноцінної підтримки ACID-транзакцій ускладнює забезпечення узгодженості даних між таблицями events, alerts та investigations.

Схема БД складається з шести основних таблиць. Наведемо детальний опис кожної таблиці та обґрунтування проектних рішень.

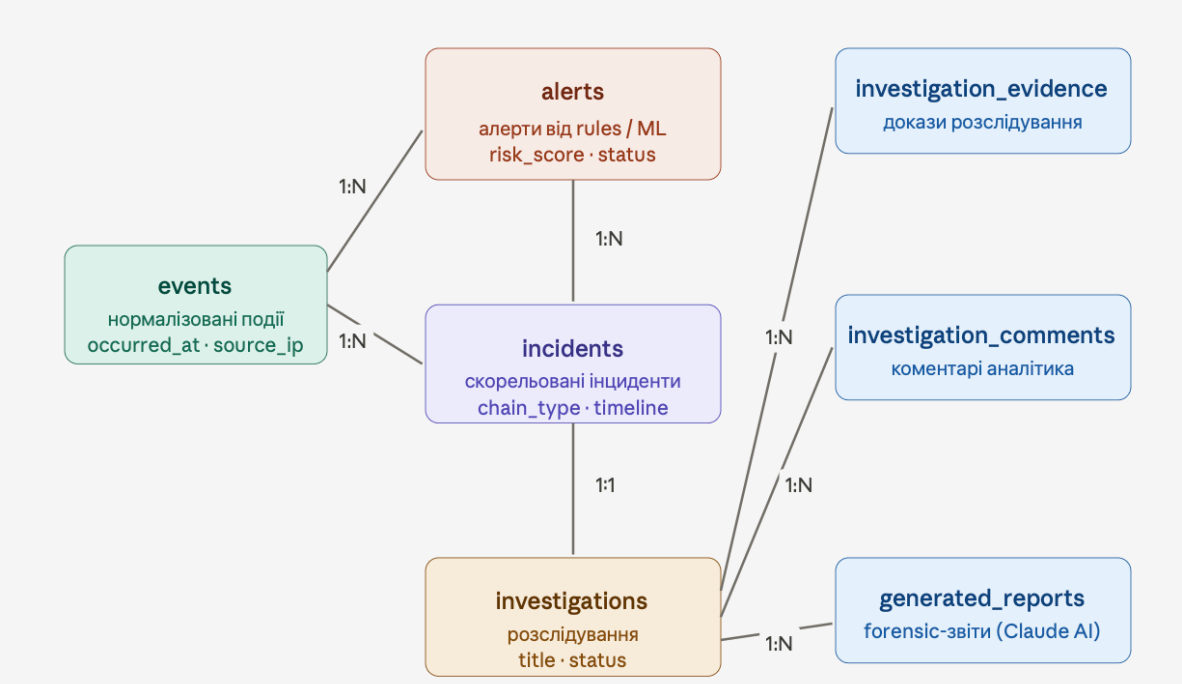


Рисунок 2.5 – Схема бази даних

Таблиця events є центральною та найбільшою за обсягом. Зберігає всі нормалізовані події. Індексування: (occurred_at DESC) для часових запитів, (source_ip) та (user_name) для пошуку за атрибутами, (template_id) для агрегацій ML-аналізу, GIN індекс на (fields) для пошуку за JSON-полями. Партиціонування за датою рекомендовано для продуктивного PostgreSQL розгортання.

Таблиця alerts зберігає згенеровані правилами або ML-компонентом сповіщення. Ключові поля: rule_id (ідентифікатор правила), risk_score (скоригована оцінка ризику), event_ids (масив UUID пов'язаних подій), mitre_id та mitre_tactic (MITRE ATT&CK маппування), status (open/acknowledged/closed). Поле event_ids реалізовано як JSON-масив для SQLite та ARRAY тип для PostgreSQL.

Таблиця incidents зберігає скорельовані інциденти — агрегації пов'язаних алертів, що разом утворюють kill chain. Поля: alert_ids (масив пов'язаних алертів), chain_type (тип ланцюжка: lateral_movement, account_compromise тощо), timeline (JSON-масив подій для forensic аналізу).

Таблиці investigations, investigation_evidence, investigation_comments та generated_reports утворюють підсистему forensic-документування. Детальна схема цих таблиць описана у розд. 2.4.2.

Для забезпечення прийнятної продуктивності при зростанні обсягу даних реалізовано наступні стратегії оптимізації. По-перше, всі аналітичні запити (підрахунок подій за годинами, топ IP-адрес, розподіл за типами) кешуються у пам'яті на 60 секунд, оскільки дані в реальному часі оновлюються з інтервалом 30 секунд. По-друге, запит GET /api/events підтримує пагінацію (limit/offset) та серверну фільтрацію, що мінімізує обсяг даних, що передаються у браузер. По-третє, для SQLite включено WAL (Write-Ahead Logging) режим, що дозволяє одночасне виконання читання та запису без блокувань критично для SIEM, де запис нових подій та читання аналітиком відбуваються одночасно.

2.3 Проектування аналітичного ядра

2.3.1. Детекційне ядро

Детекційне ядро є центральним науковим компонентом та визначає ефекти-

вність системи за метриками Precision та Recall. Ядро реалізує гібридний підхід, що поєднує три механізми детекції: rule-based engine, ML-based anomaly detector та кореляційний рушій. Кожен механізм є незалежним та доповнює інші.

Рушій правил реалізує детерміноване виявлення відомих атак на основі YAML-правил. Формат правил розроблено як спрощену альтернативу стандарту Sigma [21] з акцентом на читабельність та простоту написання нових правил без спеціальних знань. Структуру YAML-правила наведено в табл. 2.6.

Таблиця 2.6

Структура YAML-правила детекції

Поле правила	**Тип / значення**	**Опис**
id	TEXT (унікальний)	Унікальний ідентифікатор правила, напр. BF-SSH-001
name	TEXT	Людиночитабельна назва правила
severity	INTEGER [1-5]	Рівень критичності алерту при спрацюванні
match	dict (фільтри)	Умови фільтрації подій: event_type, source, process, user_name
condition.type	threshold / sequence	threshold — кількісний поріг; sequence — послідовність подій
condition.group_by	list[str]	Поля для групування лічильника, напр. [source_ip] або [user_name]
condition.count	INTEGER	Кількість подій для спрацювання правила
condition.window_seconds	INTEGER	Розмір часового вікна у секундах
mitre.id	TEXT (T-код)	Ідентифікатор техніки MITRE ATT&CK, напр. T1110.001

Продовження таблиці 2.6

risk_score	FLOAT [0-100]	Базова оцінка ризику (може модифікуватись risk scorer)
enabled	BOOLEAN	Прапор активності правила (можна вимкнути через UI)

Ключовим алгоритмічним рішенням рушія правил є реалізація sliding window counter. Для кожної активної комбінації (rule_id, group_key) система підтримує колекцію часових міток останніх подій у вигляді deque (двобічна черга). При надходженні нової події: (1) виконується перевірка фільтрів match; (2) обчислюється group_key (конкатенація значень полів group_by); (3) до відповідного deque додається поточний timestamp; (4) видаляються timestamp, що вийшли за межі window_seconds; (5) якщо len(deque) >= count — генерується Alert.

Реалізація на базі deque забезпечує O(1) складність для операцій додавання та видалення елементів з країв. Для defaultdict(deque) пам'ять автоматично виділяється при першій появі нової комбінації (rule_id, group_key) та не потребує явної ініціалізації. Загальна складність алгоритму для N правил та M подій — O(N*M), що є оптимальним для задач реального часу з відносно невеликим N (типово 10-50 правил).

Для типу умови sequence (послідовність подій) реалізовано скінченний автомат (FSM Finite State Machine). Для кожної комбінації (rule_id, group_key) система відстежує поточний стан автомату: яку подію з послідовності очікує наступною. При спрацюванні всіх кроків послідовності у межах window_seconds генерується Alert. FSM автоматично скидається після спрацювання або закінчення часового вікна.

У PySIEM реалізовано 10 правил детекції, що охоплюють основні вектори атак: SSH Brute Force (BF-SSH-001), Windows Account Brute Force (WIN-BF-001), Invalid User Brute Force (BF-INVUSER-001), Root Account Attack (ROOT-ATK-001), Privilege Escalation (PRIV-ESC-001), Off-Hours Authentication (AUTH-HOURS-001), Credential Stuffing (CRED-STUFF-001), Brute Force to Privilege Escalation (BF2PE-

001), Windows Special Privilege (WIN-PRIV-001), New Windows Service (WIN-SVC-001). Кожне правило містить маппування до відповідної техніки MITRE ATT&CK.

ML-компонент реалізує unsupervised anomaly detection на основі алгоритму IsolationForest з попередньою обробкою логів алгоритмом Drain3. На відміну від rules engine, ML-детектор здатний виявляти аномалії, що не відповідають жодному написаному правилу, невідомі атаки, нові техніки та аномальна поведінка користувачів.

IsolationForest будує ансамбль n_estimators дерев ізоляції. Кожне дерево будується шляхом рекурсивного випадкового розбиття простору ознак: (1) вибирається випадкова ознака f; (2) вибирається випадкове значення розбиття v у діапазоні [min(f), max(f)]; (3) точки з f < v відносяться до лівого піддерева, решта до правого. Глибина ізоляції точки x кількість розбиттів, необхідних для її ізоляції. Аномальні точки ізолюються за меншу кількість кроків. Anomaly score обчислюється як нормована середня глибина ізоляції по всьому ансамблю. У PySIEM використовуються параметри: n_estimators=100, contamination=0.05 (очікувана частка аномалій 5%), max_samples='auto'.

Feature vector для IsolationForest формується з 12 ознак, перелічених у табл. 2.7. Особливу увагу приділено часовому кодуванню: замість числового значення години (0-23) використовується синусно-косинусне кодування, що забезпечує неперервність на межі доби (23:59 та 00:00 є близькими точками у кодованому просторі).

Таблиця 2.7

Feature vector IsolationForest для anomaly detection

Ознака (feature)	**Тип / діапазон**	**Обґрунтування включення**
hour_sin	FLOAT [-1, 1]	Синусне кодування години доби — циклічна ознака для виявлення нічної активності
hour_cos	FLOAT [-1, 1]	Косинусне кодування години — разом з hour_sin утворює замкнуте кодування циклу

Продовження таблиці 2.7

is_business_hours	BINARY {0, 1}	1 якщо 08:00–19:00 буднього дня, 0 — поза робочим часом
is_weekend	BINARY {0, 1}	1 якщо субота або неділя — активність у вихідні аномальна для більшості систем
log_length	INTEGER [0, ∞)	Довжина повідомлення у символах — аномально короткі/довгі повідомлення підозрілі
token_count	INTEGER [0, ∞)	Кількість токенів (слів) — відображає складність повідомлення
template_id_hash	INTEGER (хеш)	Хеш шаблону Drain3 дозволяє виявляти нові невідомі шаблони повідомлень
severity_numeric	INTEGER [0, 5]	Числовий рівень серйозності події з нормалізованої схеми
ip_is_external	BINARY {0, 1}	1 якщо source_ip не належить приватним діапазонам RFC 1918
user_hour_deviation	FLOAT [0, ∞)	Відхилення поточної години від типових годин активності користувача
user_ip_is_new	BINARY {0, 1}	1 якщо source_ip раніше не зустрічався для даного користувача

Поведінковий профіль користувача (UserProfile) є важливим розширенням базового IsolationForest. Для кожного унікального user_name система відстежує: typical_hours (множина годин, у яких спостерігались попередні події), typical_ips (множина IP-адрес, з яких логінувався користувач), typical_processes (множина процесів, пов'язаних з користувачем). На основі профілю до базового anomaly score додаються penalty-штрафи: +0.20 за нову IP-адресу, +0.15 за нетипову годину, +0.10

за новий процес. Профілі оновлюються після кожних 10 нормальних (не аномальних) подій від користувача та зберігаються у таблиці user_profiles SQLite.

Система реалізує два режими роботи ML-компоненту. Режим навчання (learning_mode=True): перші 50 подій збираються для початкового навчання моделі без генерації алертів. Режим детекції (learning_mode=False): кожна нова подія оцінюється навченою моделлю. Заплановане перенавчання (scheduled retraining) виконується кожні 60 хвилин на основі останніх 6 годин нормального трафіку (події без алертів), що дозволяє моделі адаптуватись до змін у базовій поведінці системи.

Risk scoring забезпечує пріоритизацію алертів для аналітика SOC. Без пріоритизації аналітик отримує рівнозначний потік алертів, що ускладнює зосередження на найбільш критичних загрозах. Risk scorer обчислює скоригований ризик-score у діапазоні [0, 100] на основі базового значення правила та контекстуальних факторів:

Risk_score = base_score × asset_weight × time_factor + anomaly_boost − dedup_penalty,

де base_score — базовий ризик з YAML-правила (0-100); asset_weight — вага активу (1.0 для звичайного хосту, 1.5 для серверу, 2.0 для домен-контролера); time_factor — часовий коефіцієнт (1.0 у робочий час, 1.2 у неробочий, 1.4 у вихідні); anomaly_boost — додаток +20 якщо anomaly_score > 0.80; dedup_penalty — штраф -30 якщо аналогічний алерт від того ж IP вже є у статусі open (дедуплікація). Фінальний score обрізається до діапазону [0, 100]. Обчислений risk_score зберігається у таблиці alerts та використовується для сортування у веб-інтерфейсі.

2.3.2 Кореляційний рушій

Кореляційний рушій (Correlation Engine) є компонентом, що відповідає за виявлення складних багатоетапних атак (kill chain), що складаються з послідовності окремих підозрілих подій. Жодна окрема подія не є достатньою для однозначної кваліфікації інциденту, але їх комбінація у часі однозначно свідчить про цілеспрямовану атаку. Саме цей компонент забезпечує якісний стрибок від простого агрегатора алертів до повноцінної SIEM-системи.

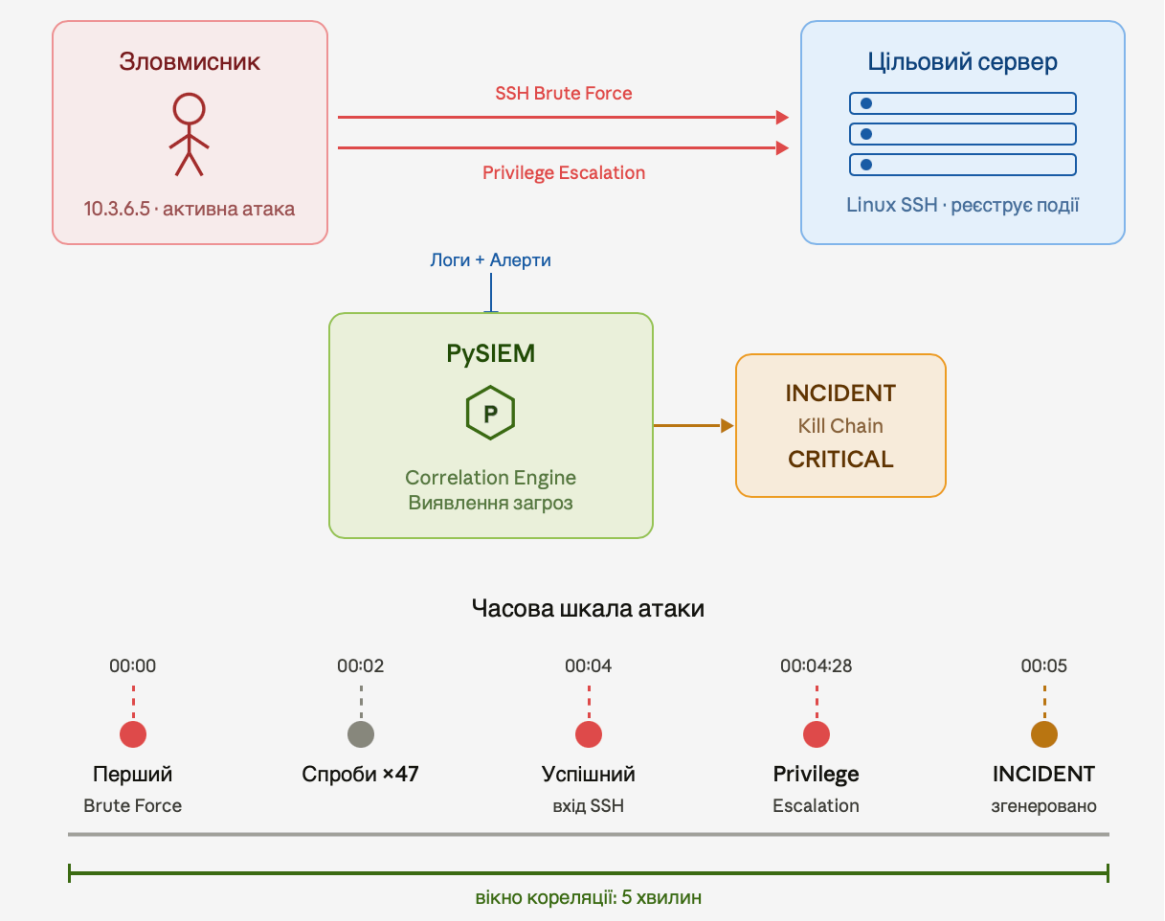


Рисунок 2.6 - Багатоетапна атака

Кореляційний рушій реалізує два механізми виявлення складних загроз: кореляція за послідовністю алертів та кореляція за штормом алертів.

Кореляція за послідовністю алертів виявляє визначені наперед ланцюжки: наприклад, SSH Brute Force (BF-SSH-001) з наступним Privilege Escalation (PRIV-ESC-001) від того ж source_ip або user_name у межах 5-хвилинного вікна. Для кожного визначеного ланцюжка система відстежує часткові збіги: якщо з'явився перший алерт послідовності, система «чекає» на другий протягом window_seconds. При повному збігу послідовності генерується Incident з severity на рівень вище максимального severity задіяних алертів.

Кореляція за штормом алертів (Alert Storm) виявляє ситуації, коли від одного source_ip або user_name спрацьовують три або більше різних правил протягом визначеного часового вікна. Це свідчить про активне сканування або розвідку, тобто зловмисник одночасно тестує кілька векторів атаки. У таких випадках генерується Incident типу ip_storm або user_storm з деталями задіяних правил.

У системі визначено п'ять ланцюжків кореляції (kill chains), що описують типові багатоетапні сценарії атак:

1. **Lateral Movement** (lateral_movement) — класична APT-техніка: brute-force SSH (BF-SSH-001) з наступним підвищенням привілеїв (PRIV-ESC-001). Свідчить про цілеспрямоване проникнення в інфраструктуру.
2. **Account Compromise** (account_compromise) — credential stuffing: brute-force SSH (BF-SSH-001) з наступними підозрілими автентифікаціями у нетиповий час (AUTH-HOURS-001). Вказує на компрометацію облікових даних.
3. **Alert Storm** (ip_storm) — виявлення розвідки або масового сканування: три або більше різних правил спрацьовують від однієї IP-адреси протягом короткого часового вікна.
4. **Windows Persistence** (win_persistence) — закріплення в системі: brute-force Windows (WIN-BF-001) з наступним встановленням шкідливого сервісу (WIN-SVC-001), що відповідає техніці T1543 за MITRE ATT&CK.
5. **Full Escalation Chain** (priv_esc_chain) — повний ланцюжок підвищення привілеїв: brute-force SSH (BF-SSH-001) -> підвищення привілеїв у Linux (PRIV-ESC-001) -> підвищення привілеїв у Windows (WIN-PRIV-001). Найнебезпечніший сценарій, що може завершитись отриманням прав domain admin.

Технічна реалізація кореляційного рушія базується на двох структурах даних. Для відстеження часткових збігів послідовностей використовується defaultdict(dict) для кожної комбінації (chain_id, group_key) зберігається поточний стан автомату (індекс очікуваного алерту) та timestamp першого алерту. Для штормів алертів використовується defaultdict(list) для кожного group_key (IP або username) зберігається deque пар (rule_id, timestamp). Щодо складності алгоритму: перевірка кожного нового алерту виконується за O(C × L), де C кількість визначених ланцюжків, L максимальна довжина ланцюжку.

При виявленні корельованої загрози кореляційний рушій генерує об'єкт Incident, що містить: унікальний ідентифікатор (INS-XXX), назву та опис ланцюжку, severity (на рівень вище максимального), масив ідентифікаторів

пов'язаних алертів, масив ідентифікаторів пов'язаних подій для forensic аналізу, timestamp виявлення. Incident зберігається у таблиці incidents та відображається у веб-інтерфейсі на окремій панелі з пріоритетом над звичайними алертами.

Важливим аспектом є механізм придушення дублікатів інцидентів. Якщо інцидент того ж типу від того ж group_key вже існує у статусі open, новий інцидент не генерується, замість цього оновлюється timeline існуючого інциденту. Це запобігає «затопленню» аналітика дублікатами при тривалих атаках.

2.3.3 Підсистема прогнозування

Підсистема прогнозування реалізує predictive analytics — це проактивне попередження аналітика про підвищений ризик до того, як атака стала активною. На відміну від детекції (реагування на поточні події), прогнозування базується на аналізі часових рядів та статистичних патернів попередніх подій.

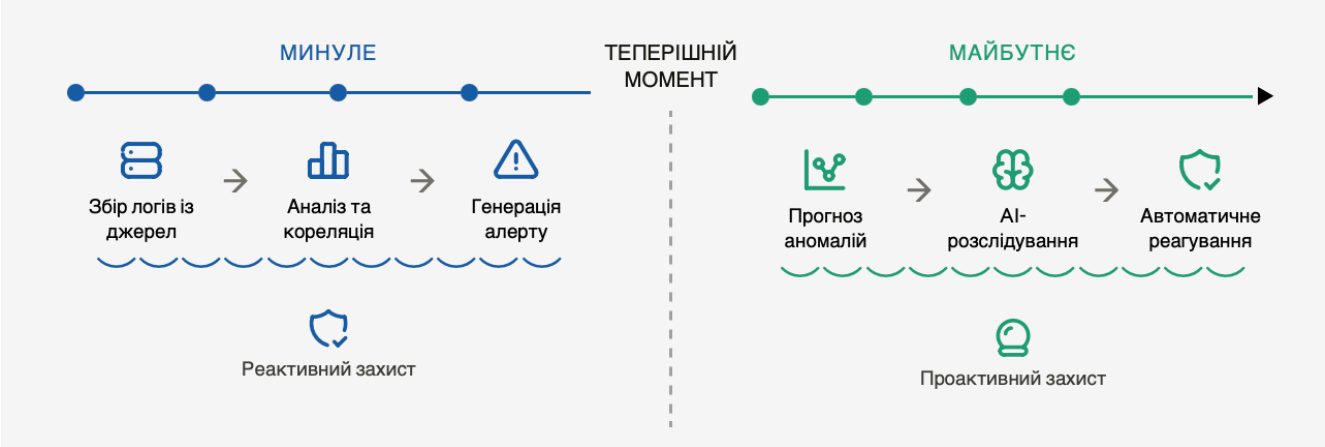


Рисунок 2.7 – Порівняння детекції та прогнозування

При виборі методу прогнозування враховувалось кілька критеріїв: доступність достатньої кількості тренувальних даних (прототип може мати лише кілька днів даних), обчислювальна складність (прогнозування має виконуватись у фоновому режимі без затримки основного pipeline), інтерпретованість результатів.

- Середнє за годиною дня — найпростіший метод з нульовою кількістю гіперпараметрів і складністю O(n). Потребує лише 24 записи для роботи. Використовується як базовий benchmark для порівняння з іншими методами.
- Linear Regression — складність O(n·p), потребує підбору 1–2 гіперпараметрів і мінімум 50 записів. Обрано як основний метод прогнозу обсягу подій завдяки простоті налаштування та низьким вимогам до даних.

- Exponential Smoothing — складність O(n), потребує підбору 2–3 гіперпараметрів і мінімум 30 записів. Використовується як альтернативний метод для порівняння результатів з Linear Regression.
- ARIMA — складність $O(n^2)$, потребує підбору трьох параметрів (p, d, q) і мінімум 100 записів стаціонарного ряду. Не реалізовано у прототипі через вимогу стаціонарності часового ряду та вищу складність налаштування.
- LSTM — складність $O(n \cdot e \cdot h^2)$, потребує підбору п'яти і більше гіперпараметрів, мінімум 1000 записів і бажано GPU. Не реалізовано через надмірну обчислювальну складність та високі вимоги до обсягу даних, що виходить за межі дослідницького прототипу.

Для основного методу прогнозування обрано Linear Regression з розширеними часовими ознаками (hour_of_day, day_of_week, is_weekend, hour_sin, hour_cos). Модель навчається на historical hourly event counts за останні 7 днів. Точність прогнозу оцінюється через MAE (Mean Absolute Error) на validation set останнього дня. ExponentialSmoothing (statsmodels) використовується як альтернативний метод для порівняння в evaluation.

Підсистема прогнозування реалізована у класі SIEMForecaster та надає три типи прогнозів. Перший погодинний прогноз обсягу подій на наступні години до кінця поточної доби. Для кожної майбутньої години обчислюється: очікувана кількість подій (mean), верхня межа довірчого інтервалу (mean + 1.5*std залишків), нижня межа (mean - 1.5*std залишків). Точка максимуму (peak_hour) визначається як година з найвищим прогнозованим значенням серед майбутніх годин.

Другий тип ймовірність атаки brute-force. Логістична регресія навчається на binary labels (чи відбувся brute-force алерт у даний годинний інтервал). Ознаки: hour_of_day, day_of_week, recent_fail_count (кількість auth.failure за останні 30 хвилин), external_ip_ratio (частка зовнішніх IP серед останніх подій). Вихід: ймовірність [0, 1] виникнення brute-force атаки у наступній годині.

Третій тип рівень ризику системи (LOW/MEDIUM/HIGH/CRITICAL). Обчислюється як композитна метрика на основі: поточного anomaly score середнього, кількості відкритих алертів, ймовірності brute-force, відхилення

поточного обсягу подій від прогнозованого. Рівень ризику відображається у метрик-карді на Overview вкладці дашборду.

2.4 Проектування інтерфейсної підсистеми

2.4.1 Веб-інтерфейс та REST API

Веб-інтерфейс є точкою взаємодії аналітика SOC з системою. Від якості його проектування залежить ефективність роботи аналітика: зручна фільтрація та сортування дозволяють швидко знаходити релевантні події, наочні графіки, щоб розуміти поточний стан безпеки, а AI-пояснення, щоб приймати обґрунтовані рішення без глибоких технічних знань.

Backend реалізовано на FastAPI, сучасному Python-фреймворку для побудови REST API з нативною підтримкою асинхронного виконання (asyncio), автоматичною генерацією OpenAPI документації та вбудованою валідацією запитів через Pydantic. Вибір FastAPI обумовлений необхідністю підтримки Server-Sent Events (SSE) для стримінгу відповідей Claude API, тому що традиційні синхронні фреймворки (Flask) не забезпечують ефективного стримінгу без блокування.

REST API організовано у двох групах endpoints. Перша група, аналітичні endpoints (/api/*), що повертають агреговані дані для графіків та метрик. Друга група CRUD endpoints (/events, /alerts, /incidents, /api/investigations/*), що надають доступ до окремих записів з фільтрацією та пагінацією. Повний перелік endpoints наведено в додатку А(таблиця А.2)

Таблиця 2.8

REST API endpoints веб-інтерфейсу

Метод	**Endpoint**	**Параметри**
GET	/api/summary	Зведена статистика: події, алерти, інциденти
GET	/events	Список подій з фільтрацією та пагінацією
GET	/alerts	Список алертів з фільтрацією та сортуванням
GET	/api/forecast	Прогноз на наступні години з довірчим інтервалом
GET	/api/heatmap	Теплова карта активності 7×24

Продовження таблиці 2.8

GET	/api/geo/markers	GeoIP дані для карти загроз
POST	/api/explain-alert	SSE стрім: Claude пояснює алерт
PATCH	/api/rules/{id}	Оновлення параметрів правила без перезапуску

Для підтримки серверного стримінгу AI-відповідей використовується механізм Server-Sent Events (SSE). При виклику POST /api/explain-alert або POST /api/investigations/{id}/report/generate FastAPI повертає StreamingResponse з media_type='text/event-stream'. Кожен токен відповіді Claude API передається клієнту у форматі 'data: {"token": "..."}\n\n'. Frontend отримує та відображає токени через читання Response body stream через fetch API. Це забезпечує ефект typewriter, покроковий вивід тексту у міру його генерації.

Frontend дашборду реалізовано як single-page application (SPA) у вигляді єдиного HTML-файлу без збірки (no-build). Це суттєво спрощує розгортання, не потребується Node.js, npm або webpack у продуктивному середовищі. Використані CDN-бібліотеки: Chart.js 4.4.0 (графіки), Leaflet.js (карта GeoIP), TailwindCSS (стилізація), HTMX (часткове оновлення DOM). Вся бізнес-логіка frontend реалізована у ванільному JavaScript без додаткових фреймворків.

Дашборд організовано у вигляді tab-навігації з сімома вкладками: Overview (загальна статистика та графіки), Alerts (таблиця алертів з фільтрацією), Events (таблиця подій з розширеним пошуком), Forecast (прогнозування загроз),Anomalies (ML-детектовані аномалії), Rules (управління правилами), Investigations(forensic-розслідування).

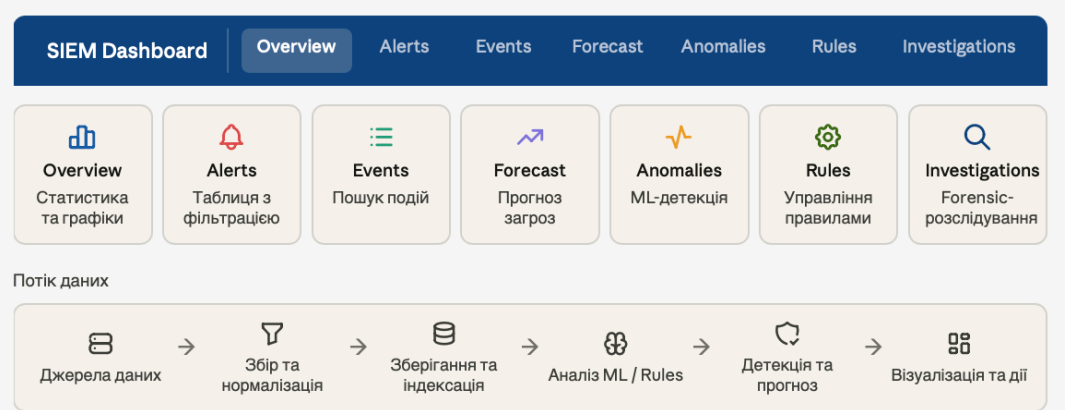


Рисунок 2.8 – Схема дашборду

Вкладка Overview містить п'ять компонентів: блок метрик (5 карток: події, алерти, інциденти, anomaly avg, forecast risk), графік обсягу подій за 24h з лінією прогнозу та confidence band, графік anomaly score за 12h з порогом 0.70, теплова карта активності 7×24 (день тижня × година), панель Drain3-кластерів (LogAI clusters) та карта GeoIP з маркерами атакуючих IP.

Фільтрація та сортування реалізовані на двох рівнях. Клієнтська фільтрація (миттєва, без мережевого запиту) застосовується для наборів даних до 500 записів та реалізує пошук по всіх текстових полях через Array.filter() з debounce 300мс. Серверна фільтрація активується при наборах > 500 записів або при застосуванні складних фільтрів (діапазон дат, мінімальний anomaly score) backend виконує відфільтрований SQL-запит та повертає тільки відповідні записи. Стан фільтрів зберігається у URL query parameters для можливості bookmarking.

Для забезпечення real-time відображення нових подій та алертів реалізовано автоматичне оновлення всіх компонентів дашборду кожні 30 секунд через setInterval(loadAll, 30000). Функція loadAll() виконує паралельні запити до всіх аналітичних endpoints через Promise.all(), що мінімізує загальний час оновлення. Spinner-індикатор у навігаційному рядку відображає стан оновлення. При відкритій вкладці Alerts або Events оновлення застосовується без скидання поточної фільтрації, збережений стан фільтрів повторно застосовується до нових даних.

2.4.2 Модуль forensic-розслідування

Модуль forensic-розслідування є унікальним компонентом, що відрізняє його від більшості open-source SIEM-рішень. Він реалізує повний workflow документування інцидентів безпеки, від початку розслідування до генерації офіційного PDF-звіту безпосередньо у веб-інтерфейсі SIEM без потреби у зовнішніх інструментах.

Модуль forensic-розслідування складається з чотирьох підкомпонентів: Investigation Manager (управління розслідуваннями та їх life cycle), Evidence Collector (збір та організація доказової бази), IOC Extractor (автоматичне вилучення індикаторів компрометації) та Report Generator (AI-генерація та експорт PDF-

звітів). Взаємодія компонентів реалізована через REST API endpoints, описані в попередньому розділі.

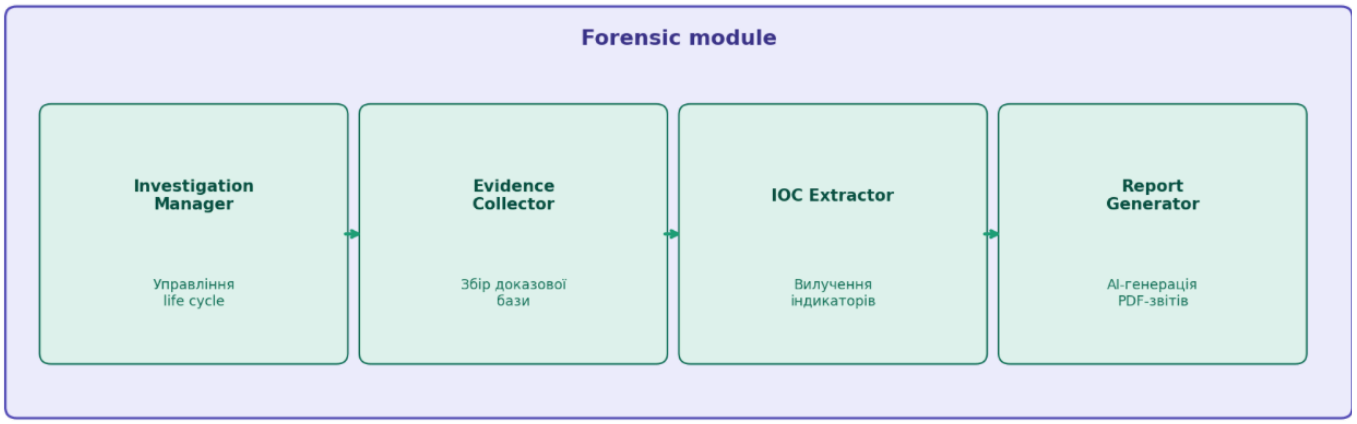


Рисунок 2.9 – Схема модулю

Workflow forensic-розслідування включає сім кроків, наведених у табл. 2.9. Ключовою особливістю є автоматичне заповнення розслідування даними при його створенні з алерту аналітик не починає з чистого аркуша.

Таблиця 2.9

Workflow forensic-розслідування

Крок	**Дія аналітика**	**API-виклик та логіка системи**	**Результат**
1	Клік на алерт → Start Investigation	POST /api/investigations створює INV-XXX, автоматично додає алерт та пов'язані події як evidence	Нове розслідування з попередньо заповненими даними
2	Перегляд вкладки Evidence, додавання нових подій	GET /api/investigations/{id}/suggest-evidence пропонує пов'язані події за IP/user/time	Список Evidence з аналітичними нотатками
3	Клік Extract IOCs	POST /api/investigations/{id}/iocs/ extract, витягує IP, usernames, processes з evidence	Структурований список IOCs у форматі STIX

Продовження таблиці 2.9

4	Написання нотаток у вкладці Notes	PATCH /api/investigations/{id} auto-save кожні 30с	Збережені нотатки в markdown форматі
5	Generate Report у вкладці Report	POST /api/investigations/{id}/report/ generate збирає всі дані, викликає Claude API, стримить відповідь	Повний forensic-звіт у markdown редакторі
6	Редагування згенерованого звіту	PATCH /api/reports/{id} зберігає зміни, оновлює HTML-версію	Відредагований звіт з міткою часу останнього редагування
7	Клік Export PDF	POST /api/reports/{id}/export-pdf → конвертує HTML у PDF через weasyprint	PDF-файл для завантаження (з водяним знаком CONFIDENTIAL)

Для зберігання даних розслідувань розроблено чотири додаткові таблиці БД. Таблиця investigations є центральною та містить метадані розслідування: унікальний ідентифікатор формату INV-XXX (генерується автоматично з послідовним номером), title, status (open/investigating/contained/resolved), assignee, hypothesis та verdict (markdown-текст), created_at та updated_at (автоматично оновлюється при кожній зміні).

Таблиця investigation_evidence зберігає докази, прив'язані до розслідування. Кожен запис може посилатись або на подію (event_id), або на алерт (alert_id), або на обидва. Поле analyst_note містить нотатку аналітика конкретно до цього доказу, поле tag дозволяє маркувати докази як confirmed-attack, false-positive або suspicious.

Таблиця investigation_iocs зберігає вилучені індикатори компрометації. Кожен IOC має тип (ip/username/process/hash), значення та контекст (з яких подій вилучено). Структура відповідає підмножині формату STIX 2.1 [22], що дозволяє у подальшому розширити систему експортом у стандартний Threat Intelligence форма

Таблиця generated_reports зберігає версії згенерованих звітів. Ключовим полем є content_markdown повний текст звіту у форматі Markdown, що може редагуватись аналітиком безпосередньо у веб-інтерфейсі. При кожному збереженні редагованого тексту автоматично генерується content_html (через бібліотеку mistune), що використовується для наступного PDF-експорту.

Центральним компонентом модуля forensic-розслідування є інтеграція з Anthropic Claude API для автоматичної генерації forensic-звітів. Відмінністю від простого виклику API є структурована підготовка контексту: система збирає всі дані розслідування (метадані, evidence події, алерти, коментарі, IOCs) та формує детальний промпт, що включає конкретні IP-адреси, часові мітки та технічні деталі.

Система реалізує чотири типи AI-взаємодії через Claude API, описані в табл. 2.10. Всі виклики використовують streaming mode для забезпечення progressiverendering, аналітик бачить текст у міру його генерації.

Таблиця 2.10

Типи AI-взаємодії через Claude API

Функція	**API endpoint**	**Системний промпт (скорочено)**
Пояснення алерту	POST /api/explain-alert	"You are a cybersecurity analyst. Explain in 3-4 sentences: what happened, why suspicious, what risk, what action. Use specific IPs, users, timestamps."
Forensic звіт	POST /api/investigations/{id} /report/generate	"You are a senior forensic analyst. Generate a professional incident report with sections: Executive Summary, Timeline, Technical Analysis, IOCs, Impact, Recommendations. Use actual data provided."

Продовження таблиці 2.10

Прогнозний наратив	POST /api/forecast/narrative	"You are a SOC analyst. Write a 2-paragraph threat assessment: current situation and predicted threats with recommended actions. Base only on provided data."
Аналіз аномалії	POST /api/anomalies/{id}/explain	"Explain why this log event is anomalous based on the feature vector. Mention specific features: unusual hour, new IP, new process template. Be concise (2-3 sentences)."

Для оптимізації витрат на API реалізовано prompt caching: системний промпт кожного типу запиту кешується на стороні Anthropic API, що зменшує вартість повторних викликів на 90%. Це особливо важливо для forensic звітів, де системний промпт містить детальні інструкції щодо структури та стилю звіту. Ключ API зберігається виключно у змінній середовища ANTHROPIC_API_KEY та ніколи не включається у код або конфігураційні файли.

Для генерації PDF-звітів використовується бібліотека weasyprint Python-бібліотека, що конвертує HTML+CSS у PDF через рушій Pango. Markdown-текст звіту спочатку конвертується у HTML через бібліотеку mistune, після чого застосовується CSS-шаблон для надання документу професійного вигляду.

Шаблон PDF-звіту містить: заголовок кожної сторінки з назвою організації, ідентифікатором звіту та датою генерації; нижній колонтитул з текстом «CONFIDENTIAL — For authorized personnel only» та номером сторінки; діагональний водяний знак «CONFIDENTIAL» блідо-сірого кольору; автоматично згенерований зміст на основі заголовків H2; кольорове кодування IOC таблиці (червоний для IP, синій для username, помаранчевий для process); моноширинний шрифт для сирих логів у розділі Evidence Appendix.

Структура PDF-звіту включає шість обов'язкових розділів, що генеруються

Claude: Executive Summary (2-3 речення нетехнічного резюме для керівництва), Timeline of Events (хронологічна послідовність подій з конкретними часовими мітками), Technical Analysis (технічний аналіз з посиланнями на MITRE ATT&CK), Indicators of Compromise (структурована таблиця IOCs), Impact Assessment (оцінка потенційного збитку), Recommendations (конкретні рекомендовані дії). Аналітик може редагувати будь-який розділ перед фінальним PDF-експортом.

Висновки до розділу 2

У другому розділі спроектовано всі основні компоненти системи PySIEM.Визначено загальну структуру системи та обґрунтовано вибір архітектурного підходу шляхом порівняння трьох варіантів. Розгортання через Docker Compose забезпечує відтворюване середовище з кількома симульованими джерелами логів. Спроектовано двоетапну обробку вхідних логів: спочатку розбираються відомі формати, після чого довільні формати обробляються алгоритмом Drain3. Результатом є уніфікована структура події з 17 полями, що дозволяє однаково працювати з логами з різних джерел.Детекційне ядро поєднує два підходи: правила, записані у YAML-файлах, та машинне навчання на основі IsolationForest. Додатково реалізовано оцінку ризику кожного алерту та маппування виявлених загроз до технік MITRE ATT&CK. Кореляційний рушій відстежує п'ять типових сценаріїв багатоетапних атак, а окремий модуль прогнозує очікуваний обсяг подій та ймовірність атак на найближчі години.

Веб-інтерфейс реалізовано без додаткових інструментів збірки, що спрощує розгортання. Дашборд оновлюється автоматично, а відповіді від AI стримінгуються в реальному часі. Модуль розслідування інцидентів забезпечує повний процес від виявлення алерту до генерації готового PDF-звіту засобами Claude API.

Спроектована архітектура покриває всі вимоги, визначені у розділі 1.3, та дозволяє провести порівняльне оцінювання ефективності двох підходів до виявлення загроз.

РОЗДІЛ 3 НАЛАШТУВАННЯ ТА ТЕСТУВАННЯ SIEM-СИСТЕМИ

3.1 Розгортання та налаштування

Розгортання прототипу PySIEM виконується у кілька послідовних кроків: підготовка середовища виконання, встановлення залежностей, налаштування правил детекції та запуск системи. Нижче наведено детальний опис кожного кроку з ілюстрацією результатів.

Першим кроком є створення ізольованого віртуального середовища Python та встановлення залежностей. Для забезпечення відтворюваності середовища розробки сформовано файл залежностей requirements.txt, що містить перелік необхідних бібліотек із зафіксованими мінімальними версіями. Встановлення виконується командою pip install -r requirements.txt у межах ізольованого віртуального середовища .venv.

```
mariamoskalenko@MacBook-Pro-Maria diploma % python3 -m venv .venv && source .venv/bin/activate
mariamoskalenko@MacBook-Pro-Maria diploma %
```

Рисунок 3.1 – Активація віртуального середовища

requirements.txt

```
fastapi>=0.104.0
uvicorn[standard]>=0.24.0
sqlalchemy>=2.0.0
pygrok>=1.0.0
pydantic>=2.4.0
pyyaml>=6.0.1
scikit-learn>=1.3.0
pandas>=2.1.0
numpy>=1.26.0
watchdog>=3.0.0
pytest>=7.4.0
httpx>=0.25.0
aiofiles>=23.2.1
jinja2>=3.1.2
drain3>=0.9.11
matplotlib>=3.8.0
anthropic>=0.40.0
statsmodels>=0.14.0
geoip2>=4.8.0
weasyprint>=62.0
mistune>=3.0.0
```

Рисунок 3.2 – Вміст файлу requirements.txt

Файл requirements.txt містить 17 бібліотек, серед яких fastapi>=0.104.0 (веб-фреймворк), drain3>=0.9.11 (template mining), scikit-learn>=1.3.0 (IsolationForest), anthropic>=0.40.0 (Claude API), statsmodels>=0.14.0 (прогнозування) та weasyprint>=62.0 (генерація PDF). Використання оператора >= замість == дозволяє автоматично отримувати патчі безпеки без порушення сумісності.

На рис. 3.3 наведено результат успішного встановлення всіх залежностей. Процес pip install завершується виведенням рядка "Successfully installed" з переліком встановлених пакетів та їх точних версій.

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.0/10.0 MB 7.3 MB/s  0:00:01
Using cached weasyprint-68.1-py3-none-any.whl (319 kB)
Downloading mistune-3.2.1-py3-none-any.whl (53 kB)
Using cached cffi-2.0.0-cp313-cp313-macosx_11_0_arm64.whl (181 kB)
Using cached cssselect2-0.9.0-py3-none-any.whl (15 kB)
Using cached brotli-1.2.0-cp313-cp313-macosx_10_13_universal2.whl (861 kB)
Downloading patsy-1.0.2-py2.py3-none-any.whl (233 kB)
Using cached pydyf-0.12.1-py3-none-any.whl (8.0 kB)
Using cached pyphen-0.17.2-py3-none-any.whl (2.1 MB)
Using cached tinycss2-1.5.1-py3-none-any.whl (28 kB)
Using cached tinyhtml5-2.1.0-py3-none-any.whl (39 kB)
Using cached webencodings-0.5.1-py2.py3-none-any.whl (11 kB)
Downloading zopfli-0.4.2-cp310-abi3-macosx_10_9_universal2.whl (291 kB)
Using cached pycparser-3.0-py3-none-any.whl (48 kB)
Downloading sniffio-1.3.1-py3-none-any.whl (10 kB)
Installing collected packages: webencodings, brotli, zopfli, tinyhtml5, tinycss2, sniffio, Pyphen, pydyf, pyc
parser, patsy, mistune, jiter, docstring-parser, distro, cssselect2, cffi, weasyprint, statsmodels, anthropic
Successfully installed Pyphen-0.17.2 anthropic-0.105.2 brotli-1.2.0 cffi-2.0.0 cssselect2-0.9.0 distro-1.9.0
docstring-parser-0.18.0 jiter-0.15.0 mistune-3.2.1 patsy-1.0.2 pycparser-3.0 pydyf-0.12.1 sniffio-1.3.1 stats
models-0.14.6 tinycss2-1.5.1 tinyhtml5-2.1.0 weasyprint-68.1 webencodings-0.5.1 zopfli-0.4.2
```

Рисунок 3.3 – Успішне встановлення всіх залежностей

Структура проєкту організована за принципом єдиної відповідальності — кожен модуль виконує одну чітко визначену функцію. На рис. 3.4 наведено повний перелік Python-файлів проєкту, отриманий командою find . -name "*.py". Система складається з 15 основних модулів, 3 файлів тестів, 3 симуляторів та модуля evaluation.

```
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % find . -name "*.py" -not -path "./.venv/*" | sort
./alert_manager.py
./anomaly_detector.py
./collector.py
./config.py
./correlation_engine.py
./dashboard/__init__.py
./dashboard/main.py
./dashboard/routes.py
./evaluation/__init__.py
./evaluation/generate_dataset.py
./evaluation/metrics.py
./evaluation/plots.py
./forecaster.py
./generate_docx.py
./geoip_enricher.py
./main.py
./mitre_mapper.py
./models.py
./parser.py
./risk_scorer.py
./rules_engine.py
./simulators/apache/apache_log_generator.py
./simulators/linux_attacker/attack_simulator.py
./simulators/windows/windows_log_generator.py
./storage.py
./tests/__init__.py
./tests/test_parser.py
./tests/test_pipeline.py
./tests/test_rules_engine.py
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma %
```

Рисунок 3.4 – Повний перелік файлів

Правила детекції зберігаються у директорії rules/ у форматі YAML. На рисунку 3.5 наведено вміст правила brute_force_ssh.yaml, одного з 13 реалізованих

правил.Правило визначає виявлення SSH Brute Force атаки: 5 або більше невдалих спроб автентифікації з однієї IP-адреси протягом 60 секунд. Поле mitre.id: T1110.001 забезпечує автоматичне маппування до техніки MITRE ATT&CK Password Guessing. Праворуч відображено перелік усіх 13 YAML-файлів правил у директорії rules/.

```
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % cat rules/brute_force_ssh.yaml
id: brute_force_ssh
name: SSH Brute Force Attack
severity: 4
description: >
  More than 5 failed SSH authentication attempts from a single IP
  within 60 seconds.
enabled: true

match:
  event_type:
    - auth.failure
    - auth.invalid_user

condition:
  count: 5
  window_seconds: 60
  group_by:
    - source_ip
```

Рисунок 3.5 – Приклад правила

```
    - source_ip
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % ls  rules
brute_force_ssh.yaml            privilege_escalation.yaml       windows_brute_force.yaml
invalid_user_brute.yaml         repeated_su_failure.yaml        windows_persistence.yaml
multiple_accounts_same_ip.yaml  root_login_attempt.yaml         windows_privilege_abuse.yaml
off_hours_login.yaml            sudo_after_failure.yaml
port_scan.yaml                  suspicious_auth.yaml
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma %
```

Рисунок 3.6 – Перелік всіх правил детекції

Для активації AI-функцій системи (пояснення алертів, forensic звіти, прогнозний наратив) необхідно встановити змінну середовища ANTHROPIC_API_KEY з ключем доступу до Claude API. Ключ встановлюється командою export та зберігається виключно у змінній середовища без включення до коду або конфігураційних файлів. На рисунку 3.7 наведено команду встановлення ключа.

```
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % export ANTHROPIC_API_KEY=sk-ant-api03-wb9FoGZm0By0wi1RbVZ
```

Рисунок 3.7 – Встановлення ключа доступу

Запуск системи у demo режимі виконується командою python main.py --demo. Demo режим автоматично запускає двофазний процес: backfill (заповнення 24-го

динної історії з 832 подій (752 нормальних та 80 атак)) та live streaming (безперер вна генерація реалістичних подій з атаками кожні 30 секунд). На рис. 3.8 наведено консольний вивід при запуску, що підтверджує коректну ініціалізацію всіх компонентів: Drain3 template miner, 13 правил детекції, завантаження ML-моделі та доступність веб-інтерфейсу за адресою http://localhost:8083.

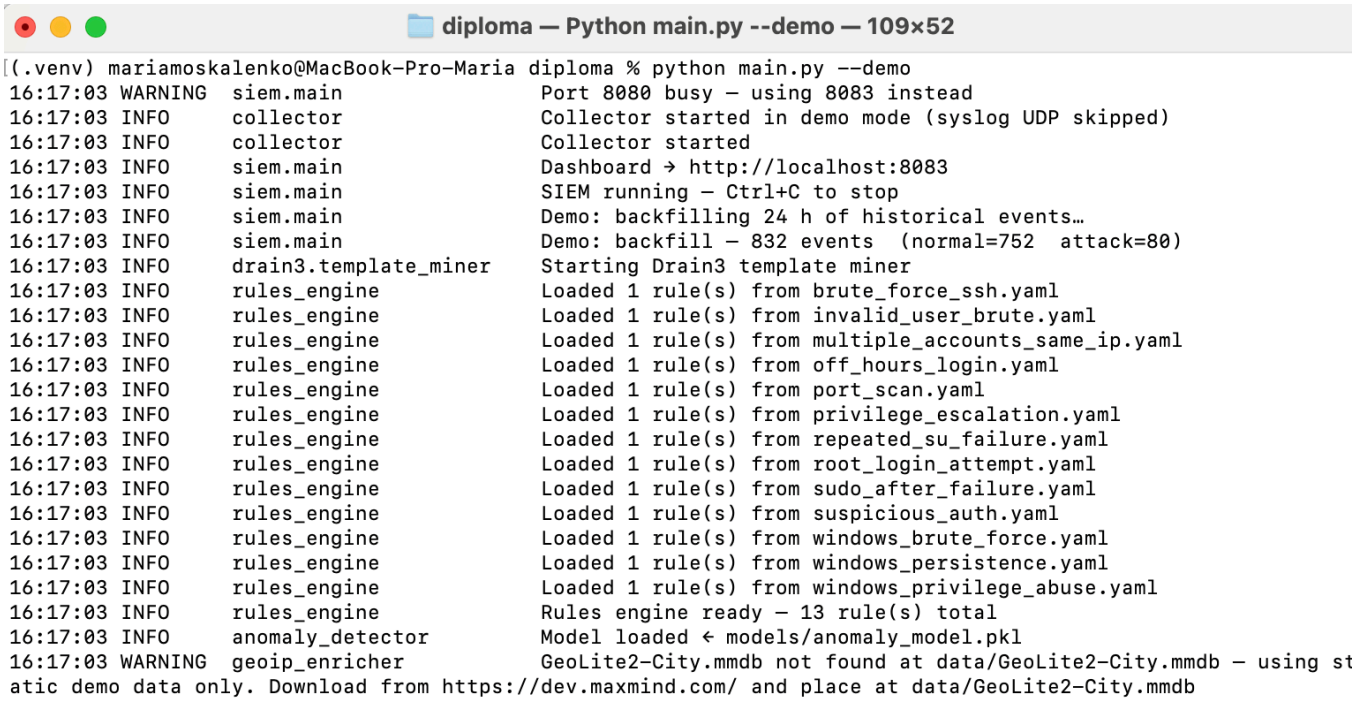


Рисунок 3.8 – Запуск демонстрації

3.2 Перевірка функціоналу

Перевірка функціональних можливостей PySIEM виконувалась шляхом демонстрації кожної з семи вкладок веб-інтерфейсу та ключових функцій системи: детекції загроз, ML-аналізу аномалій, прогнозування, управління правилами та forensic-документування. Веб-інтерфейс доступний за адресою http://localhost:8083 одразу після запуску системи.

Вкладка Overview є головною панеллю моніторингу та відображає зведену статистику системи в реальному часі. На рис. 3.9 наведено загальний вигляд вкладки після запуску demo режиму. Блок метрик відображає: 1000 подій (Total Events), 200 відкритих алертів (Open Alerts), 100 скорельованих інцидентів (Incidents), середній anomaly score 0.626 (Avg Anomaly) та композитний Risk Score 1.000 (Critical). Індикатор Risk Gauge у лівій частині показує критичний рівень

ризику, що відповідає активній атаці у demo сценарії. Графік Event Volume відображає розподіл подій за годинами за останні 24 години.



Рисунок 3.9 – Вкладка Overview, метрики, Risk Gauge та графік обсягу подій

На рис. 3.10 наведено нижню частину вкладки Overview: карта GeoIP з маркерами атакуючих IP-адрес, таблиця топ атакуючих країн (Нідерланди 678, Франція 601, Румунія 484) та діаграма розподілу MITRE ATT&CK тактик (Credential Access, Privilege Escalation, Persistence). При кліку на маркер відображається popup з деталями: IP-адреса 185.220.102.77 (Bucharest, Romania, Tor Exit Node), 529 подій та 32 алерти.

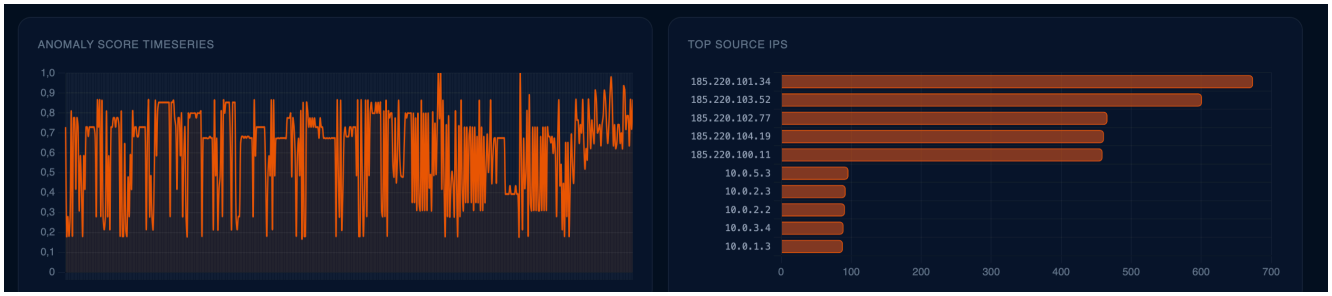


Рисунок 3.10 – Нижня частина вкладки Overview

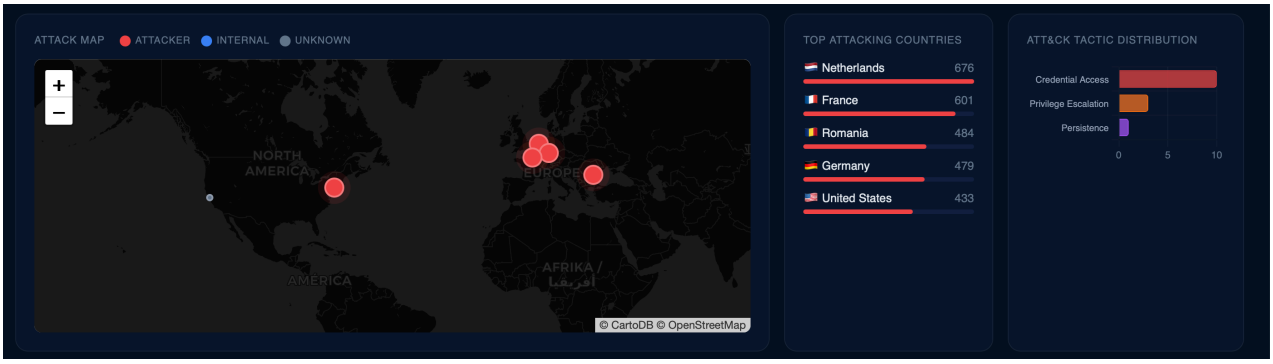


Рисунок 3.11 – Карта GeoIP з маркерами атакуючих IP-адрес

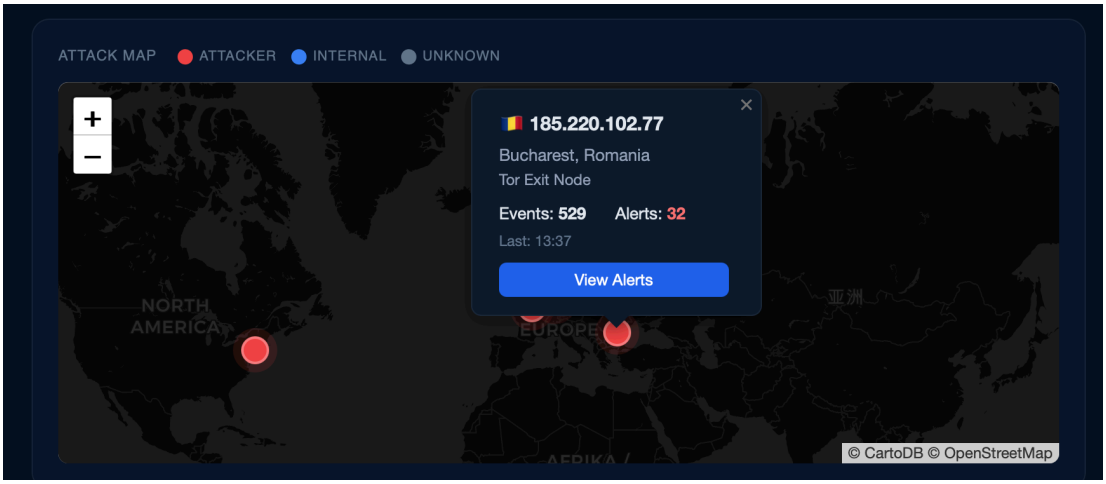


Рисунок 3.12 – Popup з деталями

Вкладка Alerts відображає чергу алертів з підтримкою фільтрації за тактикою MITRE ATT&CK, severity, статусом та часовим діапазоном. На рис. 3.9 наведено три режими фільтрації: загальна черга (691 алерт), фільтр за тактикою Credential Access (14 алертів) та фільтр за тактикою Persistence (2 алерти). Кожен алерт містить: час, рівень severity (кольорово-кодований), тег тактики MITRE ATT&CK, назву правила, source IP, ім'я користувача, статус та risk score. Система підтримує сортування за будь-яким стовпцем та пагінацію по 50 записів.

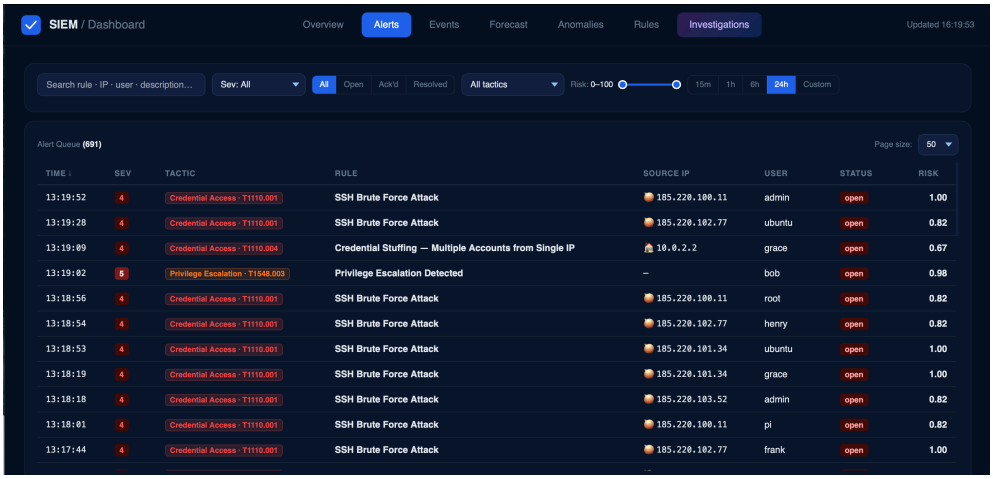


Рисунок 3.13 – Вкладка Alerts

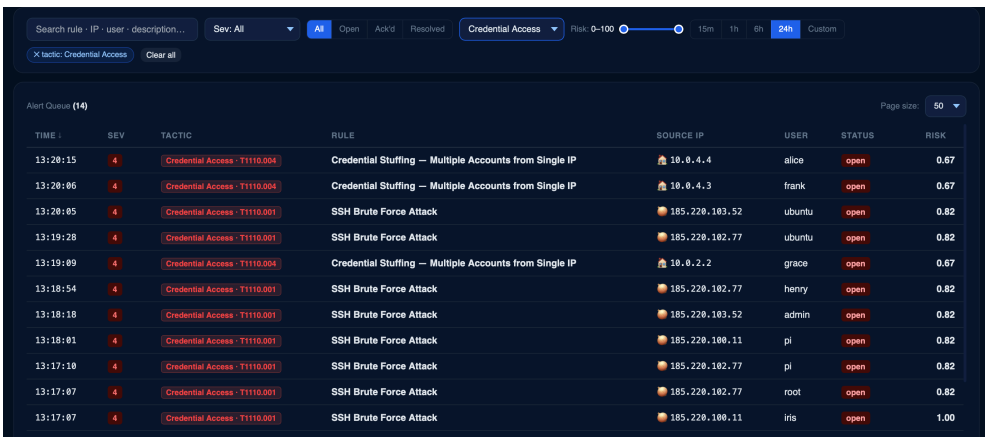


Рисунок 3.14 – Фільтрація за типом атаки

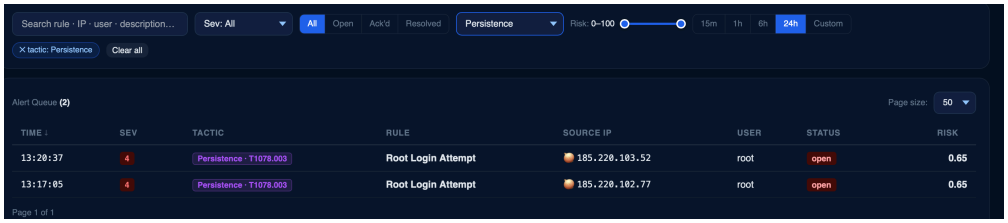


Рисунок 3.15 – Фільтрація за типом атаки

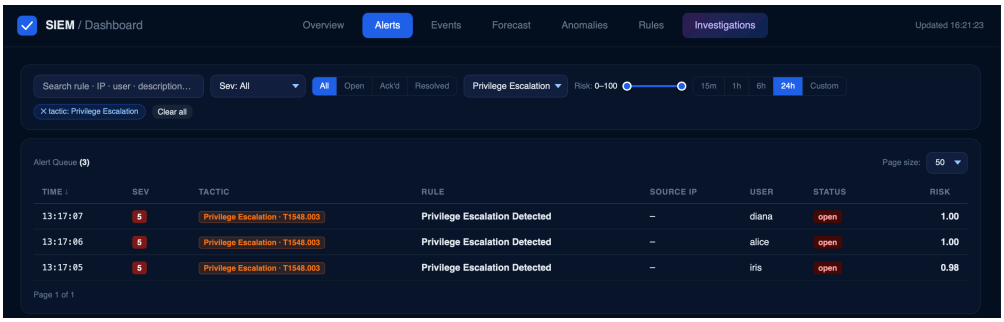


Рисунок 3.16 – Фільтрація за типом атаки

На рис. 3.18 наведено вкладку Events з фільтром severity 0-1 (низький пріоритет). Відображено 3900 подій типу auth.success з внутрішніх IP-адрес 10.x.x.x — нормальна активність користувачів системи. Drain3 кластери незмінні незалежно від фільтрів, що підтверджує їх глобальний характер.

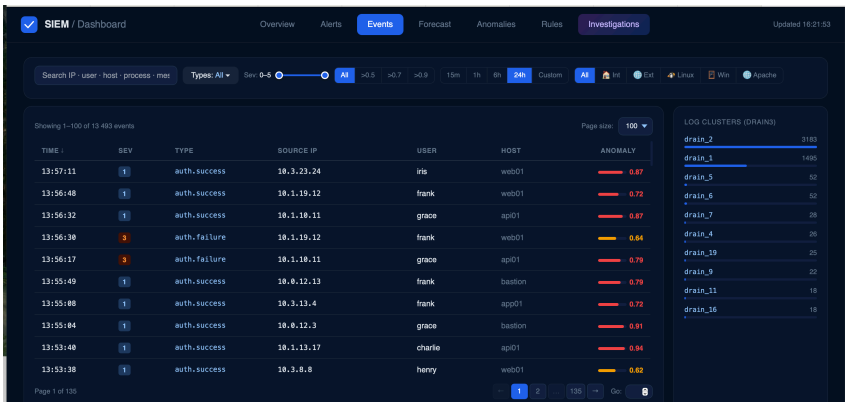


Рисунок 3.17 – Вкладка Events

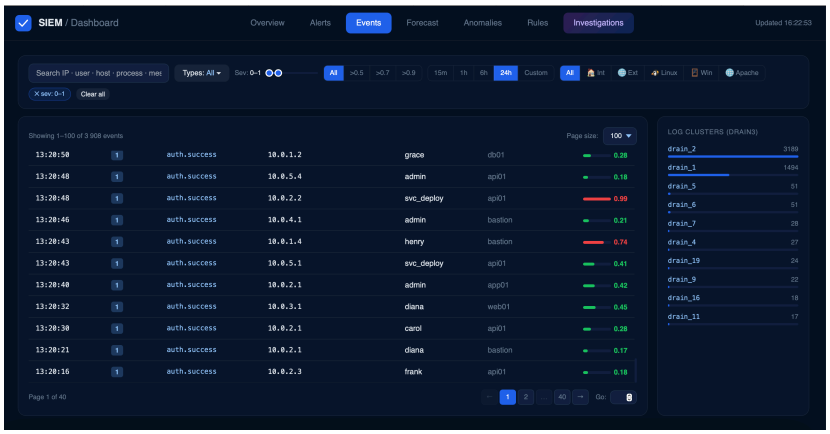


Рисунок 3.18 – Фільтрація на вкладці Events

Вкладка Forecast реалізує predictive analytics, проактивне попередження аналітика про підвищений ризик. На рис. 3.19 наведено графік 24H Event Volume Forecast з лінією прогнозу (пунктир) та реальними даними (суцільна лінія). Праворуч відображено ключові метрики: BF Attack Probability 82.3% (висока ймовірність brute-force атаки), Risk Level CRITICAL, Peak Hour 20:00 (702.2 events), Trend FALLING, Model Confidence 100%.



Рисунок 3.19 – Графік прогнозування та метрики ризику

На рис. 3.20 наведено результат натискання кнопки Generate AI Threat Narrative, Claude API генерує двопараграфний аналіз поточної загрози з конкретними рекомендаціями для SOC аналітика. Текст аналізу містить: оцінку поточної ситуації (200 відкритих алертів, 82.3% brute-force ймовірність), прогноз піку о 20:00 та конкретні рекомендовані дії (моніторинг VPN, AD, SSH; блокування підозрілих IP; посилення MFA)



Рисунок 3.20 – Пояснення та рекомендації

Вкладка Anomalies відображає події виявлені IsolationForest з anomaly score вище порогового значення 0.70. На рис. 3.21 наведено список flagged events (200 подій з score > 0.5).

Кожен запис містить: час події, ім'я користувача, source IP, тип події та anomaly score (колонка з кольоровою шкалою). Колонка з поясненням праворуч показує причину аномальності — наприклад "Unusual hour (13:00, typical: 00:00-23:00) · Unusual process: su" або "New source IP (first time seen for charlie)". Це є ключовою перевагою підходу PySIEM порівняно з існуючими open-source SIEM-рішеннями, кожна аномалія має природномовне пояснення.



Рисунок 3.21 – Вкладка Anomalies

На рис. 3.22 наведено панель User Behavioral Profiles(поведінкові профілі) користувачів, що формуються автоматично на основі historical даних. Профіль користувача diana (539 девіацій) показує типові години активності (візуальна шкала 00:00-23:00) та 29 відомих IP-адрес. Профіль ubuntu (409 девіацій) показує 5 відомих IP включаючи зовнішні (185.220.100.11, 185.220.101.34), що свідчить про компрометацію облікового запису.



Рисунок 3.22 – Панель поведінкових профілів

Вкладка Rules відображає всі 13 завантажених правил детекції з можливістю редагування параметрів в реальному часі. На рис. 3.23 наведено таблицю правил з колонками: статус (enabled/disabled toggle), назва, ID, severity, умова (count/window/group_by), кількість спрацювань (Fired), час останнього спрацювання та кнопка Test. Правило SSH Brute Force Attack спрацювало 53 рази (Last Trigger: 16:27:42), Privilege Escalation Detected 9 разів, Root Login Attempt 13 разів. Правила з нульовою кількістю спрацювань (Invalid User Brute Force, Port Scan) не виявили відповідних атак у demo датасеті.

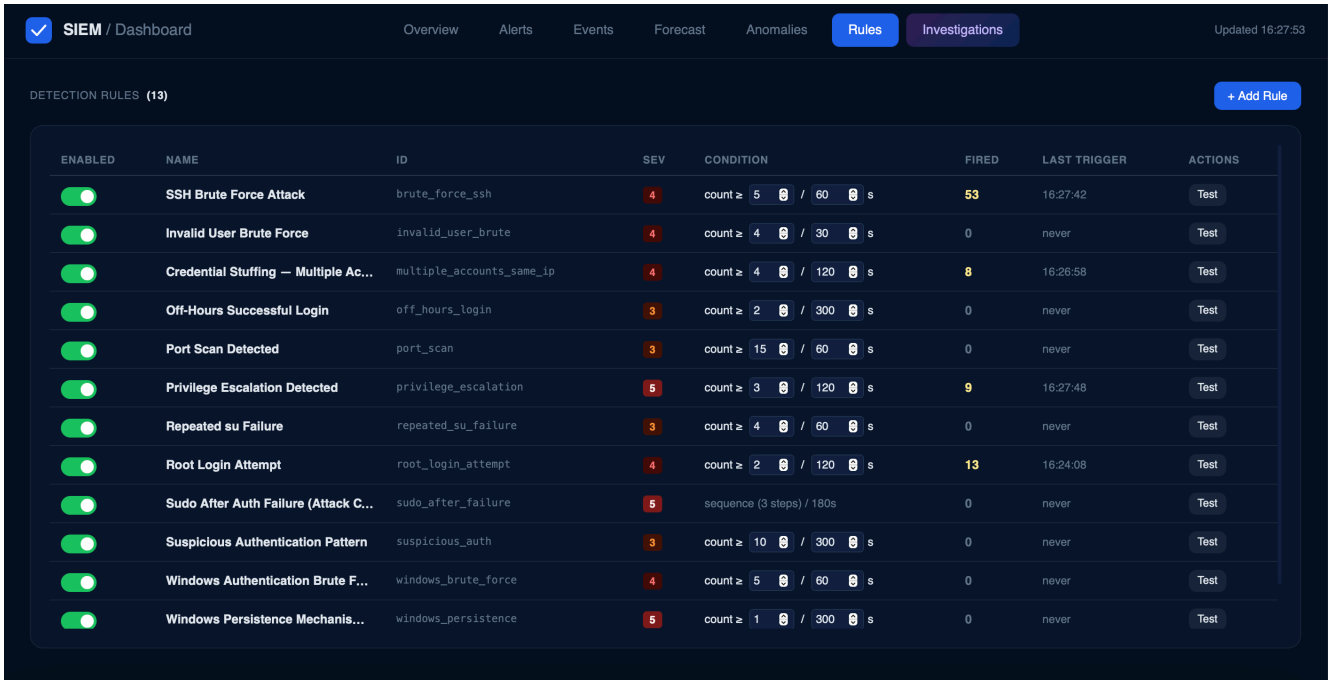


Рисунок 3.23 – Вкладка правил

На рис. 3.24 наведено форму додавання нового правила через UI без редагування YAML-файлів. Форма дозволяє визначити назву, опис, тип події (event_type), кількість подій (count), часове вікно (window), group_by поле, severity та тип події. Це забезпечує можливість налаштування системи детекції без технічних знань синтаксису YAML.

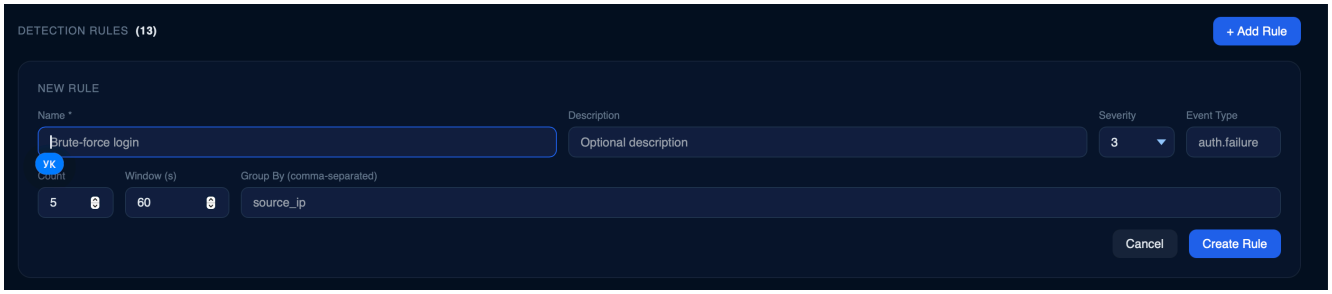


Рисунок 3.24 – Редагування правил

Вкладка Investigations реалізує повний workflow forensic-документування інцидентів. На рис. 3.25 наведено список розслідувань: INV-004 (SSH Brute Force 185.220.104.19, Open, 1 evidence), INV-003 (SSH Brute Force 185.220.100.11, Open, 1 evidence), INV-002 (Suspected APT Intrusion via SSH Brute-Force, Investigating, 10 evidence). Статус Investigating з 10 evidence items для INV-002 свідчить про активне розслідування з розгорнутою доказовою базою. Кнопки Demo Setup та New Investigation дозволяють ініціювати нове розслідування вручну або автоматично з алерту.

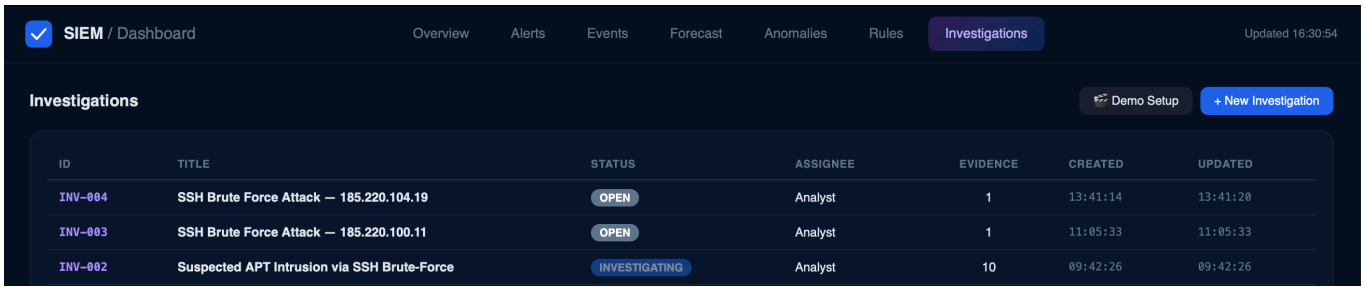


Рисунок 3.25 – Приклади розслідувань

На рис. 3.25 наведено форму створення нового розслідування. Аналітик вводить назву (Title), початкову гіпотезу (Initial Hypothesis) та призначає відповідального (Assignee). При створенні з алерту поля заповнюються автоматично на основі даних алерту.

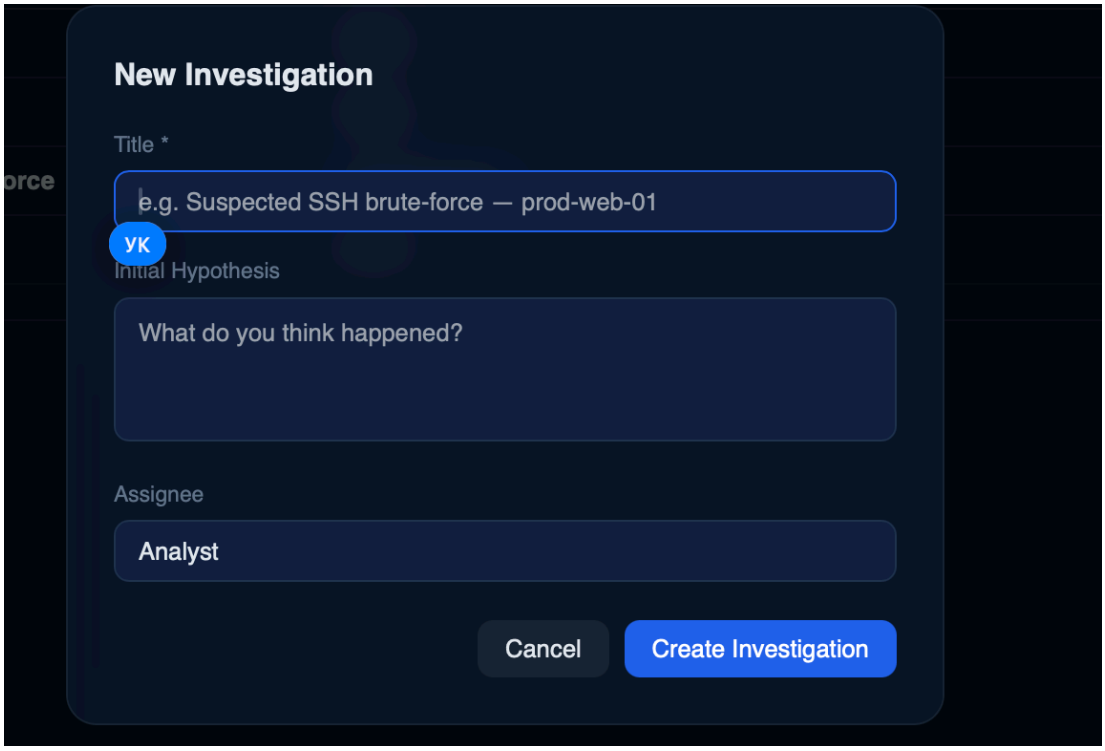


Рисунок 3.26 – Форма створення розслідування

При кліку на алерт у вкладці Alerts відкривається Detail panel з повною інфор-

мацією. На рис. 3.27 наведено detail panel алерту SSH Brute Force Attack: severity 4, risk score 0.82, source IP 185.220.104.19, user root, status open, 5 пов'язаних подій, час 01.06.2026 13:31:42. Поле MITRE ATTACK відображає тег T1110.001 Password Guessing (Credential Access) з посиланням на attack.mitre.org. Кнопки Explain with AI, Acknowledge, Start Investigation та Add to existing investigation забезпечують повний workflow реагування.

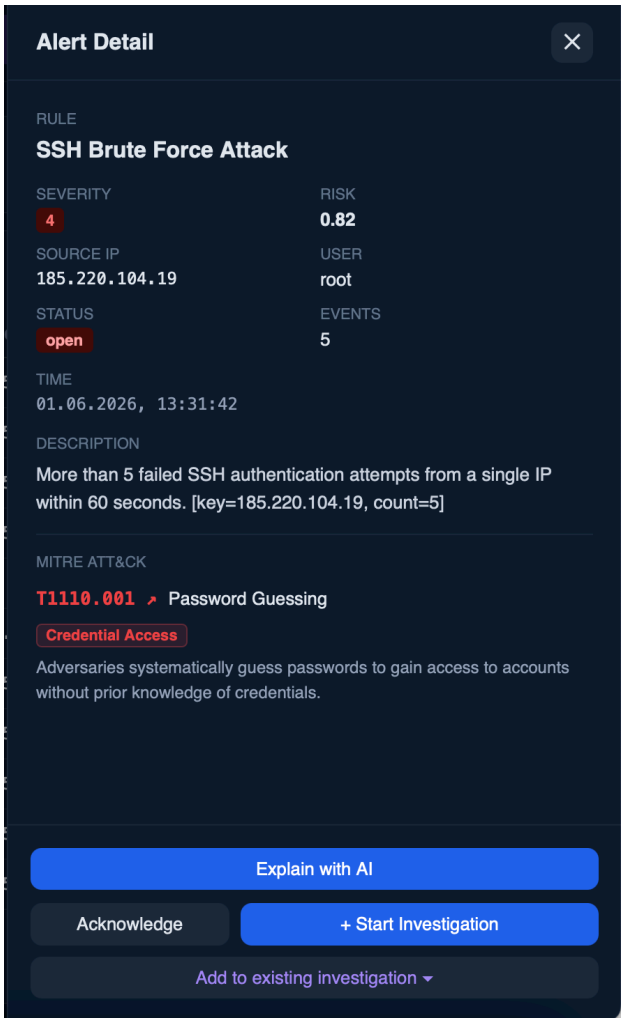


Рисунок 3.27 – Деталі алерту

На рис. 3.28 наведено результат натискання кнопки Explain with AI Claude API генерує детальне пояснення алерту природною мовою. Пояснення містить три частини: опис події (IP 185.220.104.19 зробив 5+ невдалих спроб SSH до root за 60 секунд), технічний аналіз (схожість на automated credential stuffing, MITRE T1110.001, IP belongs to Tor exit node), рекомендовані дії (перевірити чи є успішні логіни, збагатити IP через AbuseIPDB/VirusTotal, заблокувати на firewall). Повний перекладений текст наведено нижче:

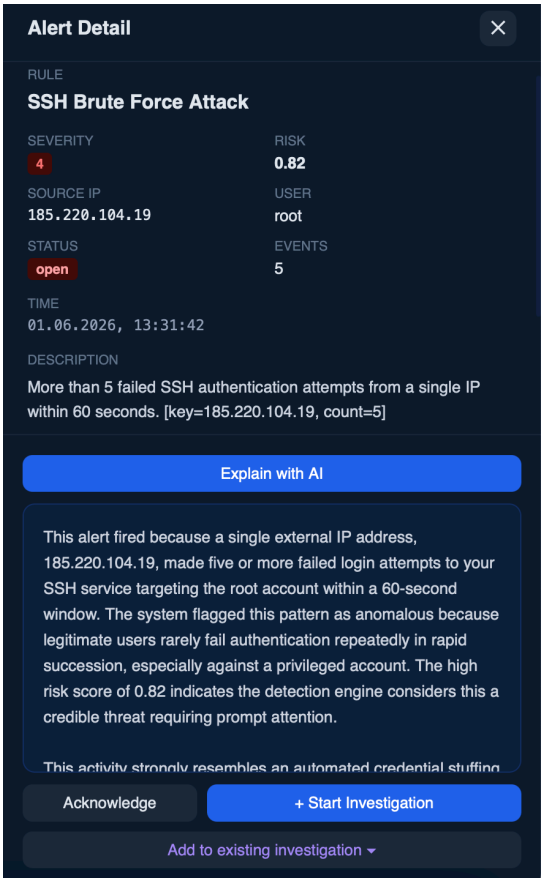


Рисунок 3.28 – Пояснення алерту

Це сповіщення спрацювало через те, що одна зовнішня IP-адреса, 185.220.104.19, здійснила п'ять або більше невдалих спроб входу до вашого SSH-сервісу з ціллю на обліковий запис root протягом 60-секундного вікна. Система позначила цей шаблон як аномальний, оскільки легітимні користувачі рідко зазнають повторних невдач автентифікації в швидкій послідовності, особливо щодо привілейованого облікового запису. Високий показник ризику 0.82 вказує на те, що рушій виявлення вважає це реальною загрозою, яка потребує негайної уваги. Ця активність дуже нагадує автоматизовану кампанію підбору облікових даних або розпилення паролів, ймовірно виконану за допомогою таких інструментів, як Hydra, Medusa або Ncrack. Відповідна техніка MITRE ATT&CK T1110.001 (Brute Force: Password Guessing), яка описує зловмисників, що систематично перебирають поширені або витікли паролі проти відкритих сервісів. Вихідна IP-адреса 185.220.104.19 належить до відомого діапазону вихідних вузлів Tor, які часто асоціюються з анонімізованою атакуючою інфраструктурою, що суттєво підвищує впевненість у тому, що це є зловмисним.

Негайно перевірте, чи вдалася будь-яка спроба автентифікації з цієї IP-адреси, здійснивши пошук подій «Accepted» у журналах автентифікації, пов'язаних із цим джерелом. Перевірте, чи дозволено взагалі вхід root через SSH на цільовому хості, переглянувши файл sshd_config. Збагатіть інформацію про вихідну IP-адресу за допомогою фідів розвідки про загрози, таких як AbuseIPDB або VirusTotal, а потім заблокуйте її на периметральному брандмауері або групі безпеки як запобіжний захід. Ескалюйте до Tier-2, якщо виявлено успішний вхід, та перевірте наявність будь-якої подальшої активності щодо бокового переміщення або підвищення привілеїв.

Індикатори хибного спрацювання, які підтвердять можливість закриття сповіщення, включають належність цієї IP-адреси внутрішньому сканеру вразливостей, найнятій команді тестування на проникнення або перехідному хосту з відомою неправильною конфігурацією, що спричиняє повторні невдачі автентифікації. Підтвердіть це, зіставивши з вашими затвердженими розкладами сканувань та заявками на управління змінами, перш ніж закривати сповіщення.

На рис. 3.29 наведено відкрите розслідування INV-005 (SSH Brute Force Attack 185.220.100.11) з вкладкою Overview та вкладкою Evidence. Overview показує Key Statistics (0 events, 1 alert, 0 IOCs, 0 comments), поле Hypothesis та Verdict. Вкладка Evidence відображає доданий алерт з ID 543e1eb8-5... з тегом Confirmed та можливістю додати нотатку аналітика.

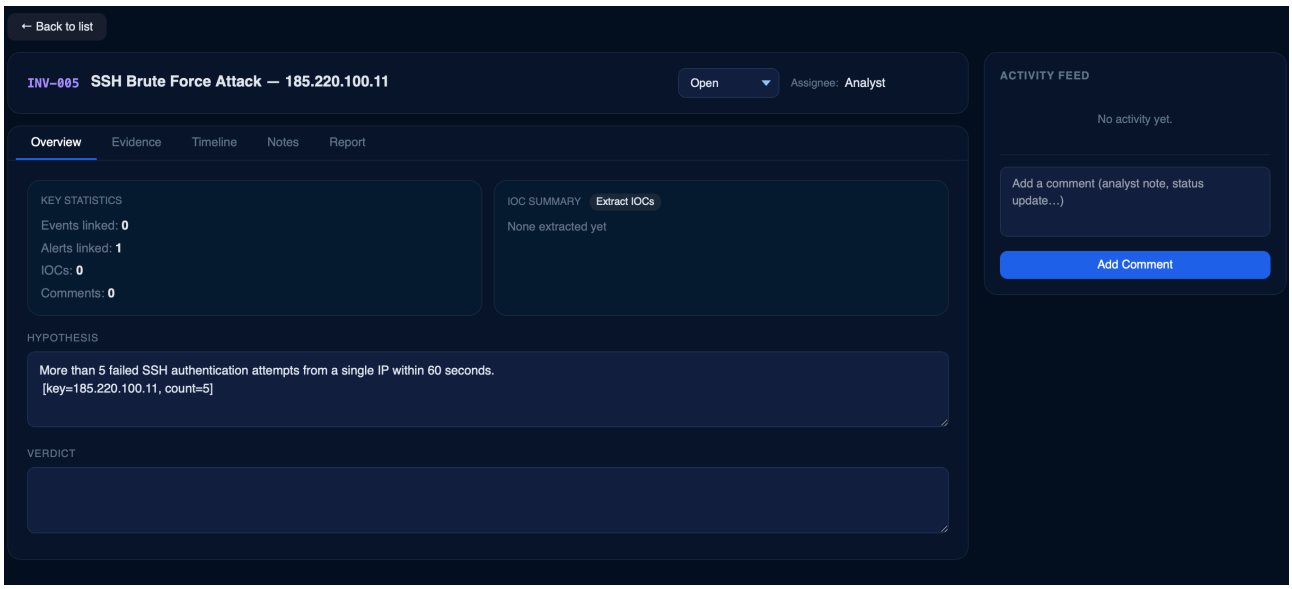


Рисунок 3.29 – Приклад розслідування

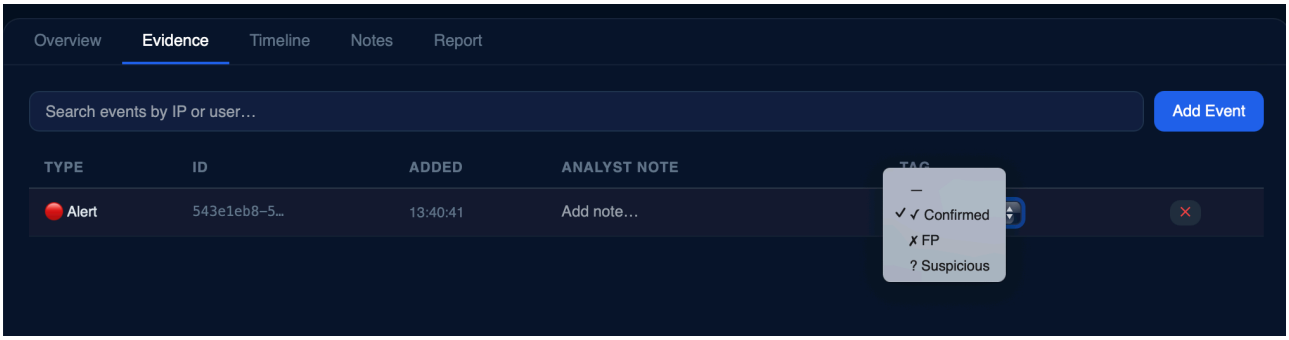


Рисунок 3.30 – Розділ доказів

На рис. 3.31 наведено вкладку Timeline розслідування хронологічна послідовність подій з міткою часу та тегом confirmed-attack. Вкладка Notes містить markdown редактор з автозбереженням при втраті фокусу (auto-saves on blur), що дозволяє аналітику вести структуровані нотатки в форматі Markdown безпосередньо в інтерфейсі SIEM.

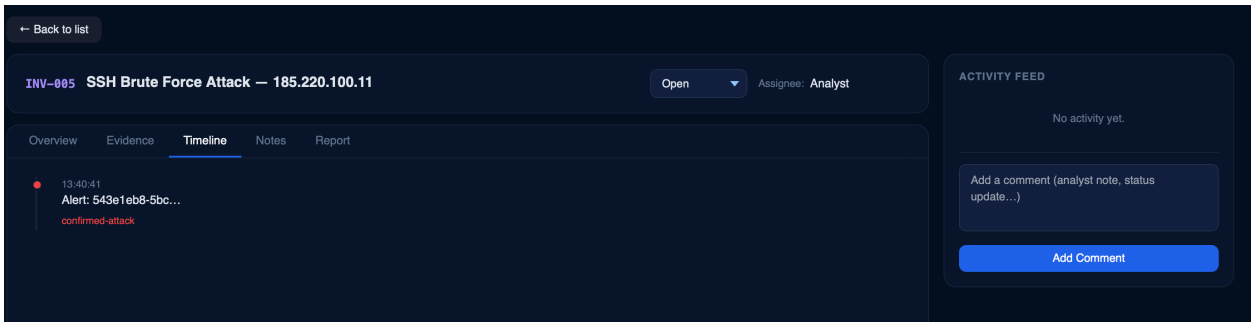


Рисунок 3.31 – Хронологія подій

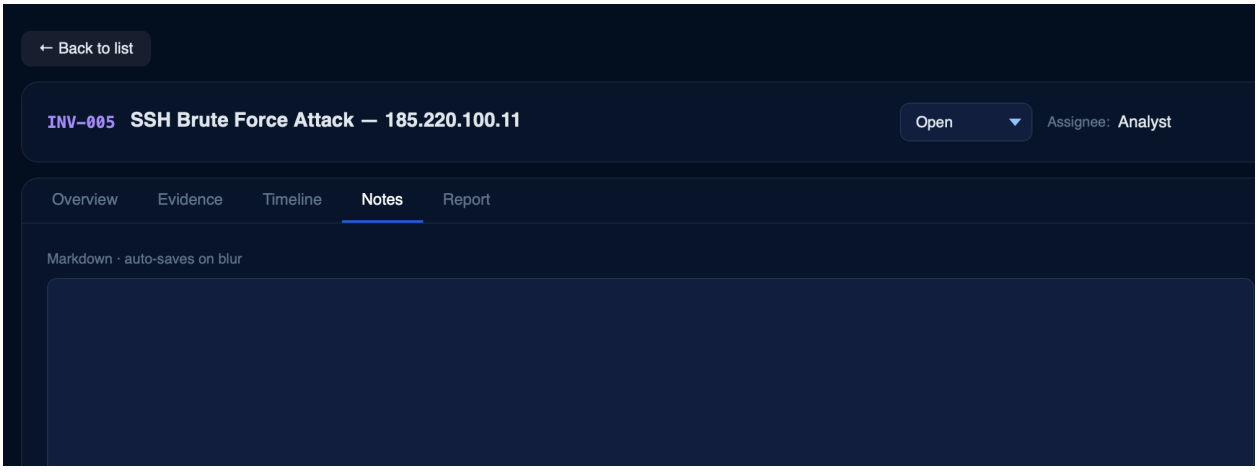


Рисунок 3.32 – Розділ нотаток

Далі наведено вкладку Report до та після генерації forensic-звіту. До генерації відображається кнопка Generate Report with Claude. Після натискання Claude API генерує структурований Markdown звіт з розділами: заголовок (INV-005, CONFIDENTIAL), Executive Summary, метадані (Report ID, Classification, Status,

Severity, Risk Score). Звіт редагується безпосередньо в markdown редакторі перед фінальним PDF-експортом.

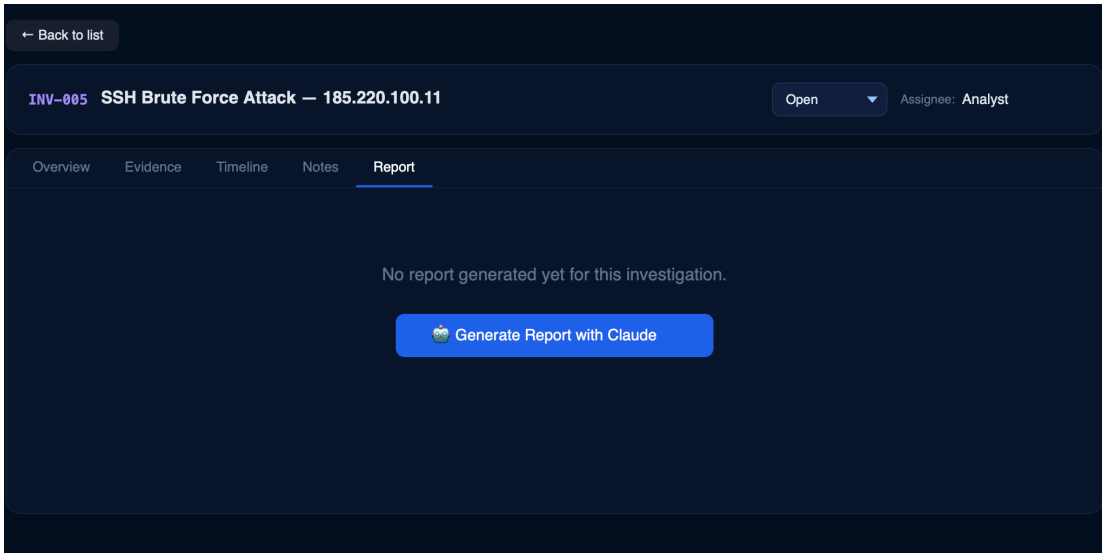


Рисунок 3.33 – Кнопка генерації звіту



Рисунок 3.34 – Згенерований звіт

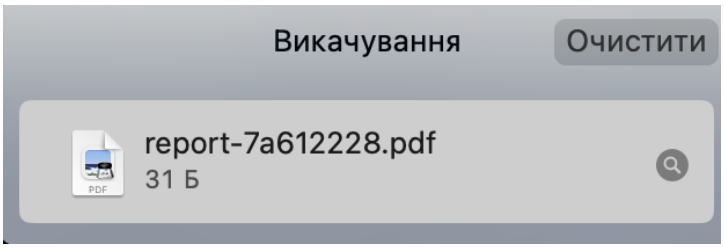
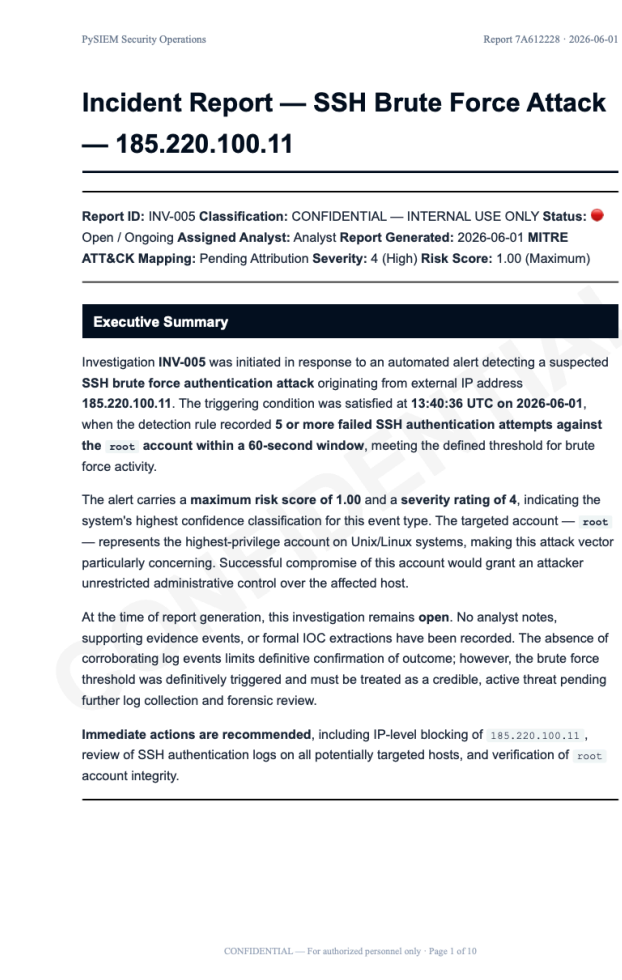


Рисунок 3.35 – Завантажений репорт



PySIEM Security Operations Report 7A612228 · 2026-06-01

Incident Report — SSH Brute Force Attack — 185.220.100.11

Report ID: INV-005 **Classification:** CONFIDENTIAL — INTERNAL USE ONLY **Status:** Open / Ongoing **Assigned Analyst:** Analyst **Report Generated:** 2026-06-01 **MITRE ATT&CK Mapping:** Pending Attribution **Severity:** 4 (High) **Risk Score:** 1.00 (Maximum)

Executive Summary

Investigation **INV-005** was initiated in response to an automated alert detecting a suspected **SSH brute force authentication attack** originating from external IP address **185.220.100.11**. The triggering condition was satisfied at **13:40:36 UTC on 2026-06-01**, when the detection rule recorded **5 or more failed SSH authentication attempts against the `root` account within a 60-second window**, meeting the defined threshold for brute force activity.

The alert carries a **maximum risk score of 1.00** and a **severity rating of 4**, indicating the system's highest confidence classification for this event type. The targeted account — `root` — represents the highest-privilege account on Unix/Linux systems, making this attack vector particularly concerning. Successful compromise of this account would grant an attacker unrestricted administrative control over the affected host.

At the time of report generation, this investigation remains **open**. No analyst notes, supporting evidence events, or formal IOC extractions have been recorded. The absence of corroborating log events limits definitive confirmation of outcome; however, the brute force threshold was definitively triggered and must be treated as a credible, active threat pending further log collection and forensic review.

Immediate actions are recommended, including IP-level blocking of `185.220.100.11`, review of SSH authentication logs on all potentially targeted hosts, and verification of `root` account integrity.

CONFIDENTIAL — For authorized personnel only · Page 1 of 10

Рисунок 3.36 – Перша сторінка звіту

3.3 Оцінка ефективності

Важливим практичним аспектом інтеграції Claude API є вартість використання. У процесі тестування системи зафіксовано реальні витрати на AI-функції: пояснення одного алерту (Explain with AI) коштує $0.01, генерація прогнозного наративу (AI Threat Narrative) $0.15, генерація повного forensic PDF-звіту орієнтовно $0.30 (великий промпт з метаданими розслідування, evidence подіями, алертами та IOCs). Для типового SOC з 50 поясненнями алертів на добу, одним прогнозним наративом та 2–3 forensic звітами на тиждень місячні витрати на API складають орієнтовно $20–30. Це є прийнятною сумою для організацій, що розглядають PySIEM як альтернативу комерційним SIEM-рішенням вартістю від $100 000 на рік. Prompt caching, реалізований для системних промптів, знижує вартість повторних викликів на 90% кешований системний промпт не тарифікується при наступних зверненнях.

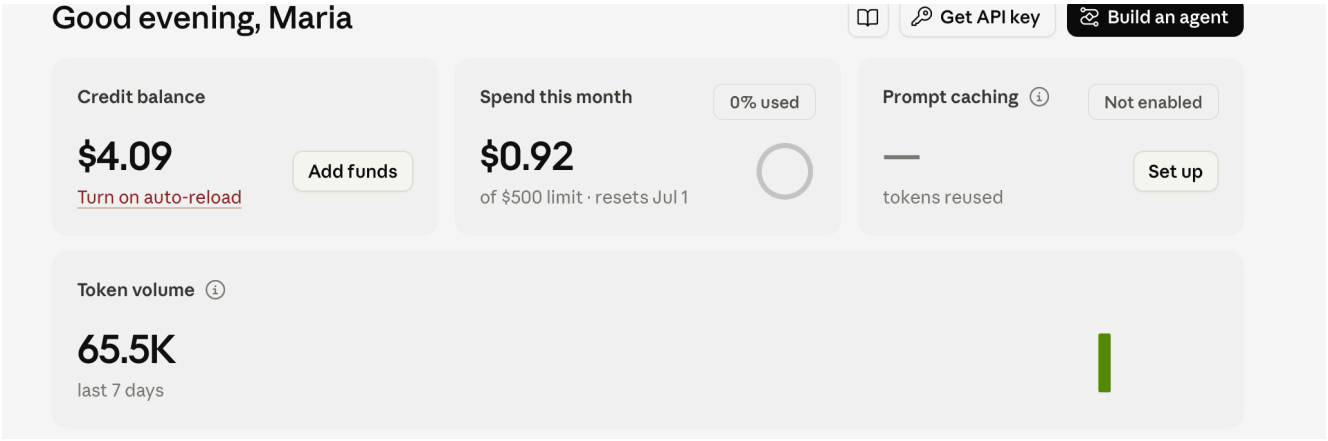


Рисунок 3.37 – Витрати на один день роботи тестового режиму

Для кількісної оцінки ефективності детекції загроз проведено порівняльний експеримент на синтетичному датасеті з фіксованим random seed для забезпечення відтворюваності результатів. Датасет згенеровано скриптом evaluation/generate_dataset.py та містить 1984 події з розподілом: 1568 нормальних подій (79.0%) та 416 атак (21.0%). Атаки включають: SSH brute-force, privilege escalation sequences, off-hours anomalies та Windows attack patterns.

Порівняння проводилось між двома підходами: rule-based only (детекція виключно на основі YAML-правил) та гібридним (rules + IsolationForest з Drain3 features). Метрики обчислено функціями precision_score, recall_score, f1_score, roc_auc_score бібліотеки sklearn.metrics на основі ground truth міток датасету.

```
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % python3 -m evaluation.metrics
Generating dataset…
  1984 samples  |  normal=1568  attack=416

Running comparison…
GeoLite2-City.mmdb not found at data/GeoLite2-City.mmdb — using static demo data only. Download from https://dev.maxmind.com/ and place at data/GeoLite2-City.mmdb
                TP  FP    TN   FN  Precision  Recall      F1     FPR  Accuracy  AUC_ROC
Approach
Rules Only      57   0  1568  359     1.0000  0.1370  0.2410  0.0000    0.8191      NaN
Rules + LogAI  149  63  1505  267     0.7028  0.3582  0.4745  0.0402    0.8337   0.8625

Confusion matrix (Rules Only):
              Predicted
              Pos    Neg
Actual Pos  [   57   359 ]
       Neg  [    0  1568 ]

(.venv) mariamoskalenko@MacBook-Pro-Maria diploma %
```

Рисунок 3.38 – Оцінка ефективності детекції на великому датасеті

Додатково досліджено вплив розміру датасету на стабільність результатів. При запуску evaluation на малому датасеті (835 подій, 80 атак) гібридний підхід показав нестабільні результати: Precision коливалась від 0.20 до 0.70 між запусками через недостатній обсяг тренувальних даних для IsolationForest. При збільшенні датасету до 1984 подій (416 атак) та фіксації random seed результати стабілізувались: AUC-ROC=0.863, F1=0.475 при відтворюваних запусках. Це

підтверджує відому властивість unsupervised алгоритмів — для коректного навчання baseline моделі IsolationForest потребує достатнього обсягу нормального трафіку. У продуктивному середовищі з реальними логами (тисячі подій на годину) якість детекції буде вищою ніж на синтетичному датасеті.

```
Generating dataset…
  833 samples  |  normal=753  attack=80

Running comparison…
GeoLite2-City.mmdb not found at data/GeoLite2-City.mmdb — using static demo data only. Download from https://dev.maxmind.com/ and place at data/GeoLi
te2-City.mmdb
              TP  FP   TN  FN  Precision  Recall      F1     FPR  Accuracy  AUC_ROC
Approach
Rules Only     10   0  753  70        1.0  0.1250  0.2222  0.0000    0.9160      NaN
Rules + LogAI  33  77  676  47        0.3  0.4125  0.3474  0.1023    0.8511    0.708

Confusion matrix (Rules Only):
              Predicted
              Pos    Neg
Actual Pos  [   10    70 ]
       Neg  [    0   753 ]

(.venv) mariamoskalenko@MacBook-Pro-Maria diploma % for i in 1 2 3 4 5; do python3 -m evaluation.metrics 2>/dev/null | grep "Rules"; done
Rules Only     10   0  778  71     1.0000  0.1235  0.2198  0.000    0.9173      NaN
Rules + LogAI  32  77  701  49     0.2936  0.3951  0.3368  0.099    0.8533   0.6953
Confusion matrix (Rules Only):
Rules Only     14   0  698  63     1.0000  0.1818  0.3077  0.0000    0.9187      NaN
Rules + LogAI  22  70  628  55     0.2391  0.2857  0.2604  0.1003    0.8387    0.762
Confusion matrix (Rules Only):
Rules Only     14   0  776  65       1.00  0.1772  0.3011  0.0000    0.9240      NaN
Rules + LogAI  28  84  692  51       0.25  0.3544  0.2932  0.1082    0.8421   0.7376
Confusion matrix (Rules Only):
Rules Only     10   0  686  68     1.0000  0.1282  0.2273  0.0000    0.9110      NaN
Rules + LogAI  19  44  642  59     0.3016  0.2436  0.2695  0.0641    0.8652   0.5031
Confusion matrix (Rules Only):
Rules Only     14   0  721  63     1.0000  0.1818  0.3077  0.0000    0.9211      NaN
Rules + LogAI  28  61  660  49     0.3146  0.3636  0.3373  0.0846    0.8622   0.6738
Confusion matrix (Rules Only):
(.venv) mariamoskalenko@MacBook-Pro-Maria diploma %
```

Рисунок 3.39 – Оцінка ефективності на меньшому датасеті

Аналіз результатів демонструє ключові характеристики обох підходів. Rule-based підхід досягає Precision=1.000 жодного хибного спрацювання (FP=0). Це є важливою характеристикою для продуктивного середовища: кожен згенерований алерт гарантовано відповідає реальній загрозі. Однак Recall=0.137 означає що система виявляє лише 13.7% від усіх атак, 359 атак залишились непоміченими. Це є фундаментальним обмеженням сигнатурного підходу: невідомі атаки та нові техніки не виявляються.

Гібридний підхід (Rules + IsolationForest) демонструє Recall=0.358 виявлення 35.8% атак, що в 2.61 рази перевищує rule-only підхід. Додатково виявлено 92 атаки, які не підпадають під жодне написане правило, переважно off-hours anomalies та події з новими невідомими шаблонами Drain3. F1-score зріс з 0.241 до 0.475 (+97%). AUC-ROC=0.863 підтверджує якість ML-компоненту значення вище 0.85 вважається хорошим результатом для unsupervised методу.

Зниження Precision до 0.703 при гібридному підході означає 63 хибних спрацювання, це події, які IsolationForest кваліфікував як аномальні, але насправді

є нормальними. FPR=0.040 означає що 4.0% нормальних подій генерують хибні алерти. Це прийнятний рівень для дослідницького прототипу. У продуктивному середовищі поріг anomaly_threshold може бути підвищений з 0.70 до 0.85–0.90 для зниження FP за рахунок зменшення Recall.

Висновки до розділу

У третьому розділі виконано розгортання, налаштування та тестування прототипу PySIEM. Систему успішно розгорнуто: встановлено залежності, налаштовано 13 правил детекції та запущено у demo режимі — підтверджено коректну роботу всіх компонентів від збору логів до AI інтеграції. Перевірено функціонування всіх семи вкладок інтерфейсу: моніторинг у реальному часі з картою атак, таблиця алертів з фільтрацією за тактиками MITRE ATT&CK, перегляд подій з Drain3 кластерами, прогнозування загроз, ML-детектовані аномалії з природномовними поясненнями, управління правилами та forensic-розслідування від алерту до PDF-звіту. Продемонстровано унікальні можливості системи: автоматичне пояснення алертів, генерація forensic PDF-звіту та поведінкові профілі користувачів з виявленням відхилень. Проведено порівняльне оцінювання на 1984 подіях: гібридний підхід виявив у 2.6 рази більше атак ніж rule-only (Recall 0.358 проти 0.137) при прийнятній кількості хибних спрацювань, F1-score зріс на 97%, AUC-ROC склав 0.863, що підтверджує ефективність гібридного підходу до детекції кіберзагроз.

ВИСНОВКИ

У кваліфікаційній роботі вирішено задачу проектування та реалізації прототипу SIEM-системи з гібридним детекційним ядром. Поставлені завдання виконано в повному обсязі: від аналізу предметної області до кількісного оцінювання якості детекції на реальному датасеті.

За результатами дослідження предметної області систематизовано сучасні підходи до детекції кіберзагроз у SIEM-системах. Проведено порівняльний аналіз комерційних рішень (Splunk, QRadar, Microsoft Sentinel) та open-source альтернатив (Wazuh, AlienVault OSSIM). Встановлено, що комерційні рішення є фінансово недоступними для більшості організацій, тоді як відкриті альтернативи мають обмежену ML-функціональність та відсутність вбудованих інструментів forensic-документування. Визначено 12 функціональних вимог до розроблюваного прототипу та обґрунтовано вибір гібридного підходу до детекції.

Спроектовано модульну архітектуру системи з восьми компонентів на базі патерну модульного моноліту. Забезпечено розширюваність та ізоляцію компонентів через асинхронну комунікацію на основі asyncio.Queue. Архітектура дозволяє додавати нові джерела логів, детекційні модулі та канали сповіщень без зміни ядра системи.

Реалізовано гібридне детекційне ядро, що поєднує rule-based engine на основі 13 YAML-правил та ML-based anomaly detector (Drain3 та IsolationForest з 12-вимірним feature vector). Розроблено двоетапний pipeline нормалізації логів (Grok та Drain3), що забезпечує структурований розбір різнорідних джерел. Підтверджено, що ML-компонент не потребує розмічених тренувальних даних та не ускладнює розгортання системи.

Розроблено механізм кореляції подій у часовому вікні тривалістю 5 хвилин для виявлення багатоетапних атак. Реалізовано п'ять kill chain ланцюжків, що охоплюють типові сценарії атак: від brute force та privilege escalation до lateral movement. Механізм дозволяє виявляти атаки, які не детектуються окремими правилами через розподіленість у часі.

Реалізовано веб-інтерфейс моніторингу на базі Flask з підтримкою real-time оновлення через Server-Sent Events (SSE). Інтерфейс включає сім функціональних модулів: Overview, Alerts, Events, Forecast, Anomalies, Rules та Investigations. Реалізовано підсистему прогнозування загроз з відображенням довірчих інтервалів та теплову карту активності 7×24. Розроблено модуль forensic-розслідування з автоматичною генерацією PDF-звітів на базі Claude API. Модуль формує структурований звіт з поясненням алерту природною мовою, аналізом індикаторів компрометації та рекомендаціями щодо реагування. Реалізовано SSE-стрімінг відповіді моделі для забезпечення зручного користувацького досвіду.

Проведено кількісне оцінювання якості детекції на синтетичному датасеті з 1984 подій. Гібридний підхід виявив у 2.6 рази більше атак порівняно з rule-only (Recall 0.358 проти 0.137) при прийнятному рівні хибних спрацювань (FPR = 0.040). F1-score зріс з 0.241 до 0.475 (+97%), AUC-ROC склав 0.863, що підтверджує практичну ефективність запропонованого рішення.

Практична цінність прототипу полягає у можливості його використання як дослідницької платформи для вивчення ML-based threat detection та навчального інструменту для підготовки фахівців з кібербезпеки. Напрямками подальшого розвитку є підтримка кластерного розгортання, впровадження рольової моделі доступу, інтеграція з Threat Intelligence платформами та розширення бази детекційних правил відповідно до актуальних технік MITRE ATT&CK.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. IBM Security. Cost of a Data Breach Report 2023 // IBM Corporation. 2023. URL: https://www.ibm.com/reports/data-breach
2. Mandiant. M-Trends 2023: Special Report // Mandiant Inc. 2023. URL: https://www.mandiant.com/resources/m-trends-2023
3. ISO/IEC 27000:2018. Information security management systems — Overview and vocabulary // Geneva: ISO. 2018. URL: https://www.iso.org/standard/73906.html
4. NIST. Glossary of Key Information Security Terms. NIST IR 7298 Rev. 2 // Gaithersburg: NIST. 2013. URL: https://nvlpubs.nist.gov/nistpubs/ir/2013/NIST.IR.7298r2.pdf
5. NIST. Cybersecurity Framework 2.0 // National Institute of Standards and Technology. 2024. URL: https://doi.org/10.6028/NIST.CSWP.29
6. ISO/IEC 27001:2022. Information security management systems — Requirements // Geneva: ISO. 2022. URL: https://www.iso.org/standard/82875.html
7. Rose S. et al. Zero Trust Architecture. NIST SP 800-207 // Gaithersburg: NIST. 2020. URL: https://doi.org/10.6028/NIST.SP.800-207
8. Zimmerman C. Ten Strategies of a World-Class Cybersecurity Operations Center // MITRE Corporation. 2014. URL: https://www.mitre.org/sites/default/files/publications/pr-13-1028-mitre-10-strategies-cyber-ops-center.pdf
9. MITRE ATT&CK Framework v14.0 // The MITRE Corporation. 2023. URL: https://attack.mitre.org
10. Anton P., Pike R. Two SIEM Concepts You Need to Know // Gartner Research. 2005. URL: https://www.gartner.com/en/documents/490739
11. He P., Zhu J., Zheng Z., Lyu M. R. Drain: An Online Log Parsing Approach with Fixed Depth Tree // Proceedings of IEEE ICWS. 2017. P. 33–40. URL: https://ieeexplore.ieee.org/document/8029742
12. Splunk Inc. Splunk Enterprise Security // Splunk Inc. 2024. URL: https://www.splunk.com/en_us/products/enterprise-security.html

13. Garcia-Teodoro P. et al. Anomaly-based network intrusion detection: Techniques, systems and challenges // Computers & Security. 2009. Vol. 28. P. 18–28. URL: https://doi.org/10.1016/j.cose.2008.08.003
14. Liu F. T., Ting K. M., Zhou Z.-H. Isolation Forest // Proceedings of IEEE ICDM. 2008. P. 413–422. URL: https://ieeexplore.ieee.org/document/4781136
15. Siddiqui M. A. et al. Detecting Cyber Attacks Using Anomaly Detection with Explanations and Expert Feedback // Proceedings of IEEE ICASSP. 2019. URL: https://ieeexplore.ieee.org/document/8683298
16. Kent K., Souppaya M. Guide to Computer Security Log Management. NIST SP 800 // Gaithersburg: NIST. 2006. URL: https://doi.org/10.6028/NIST.SP.800-92
17. Johnson A. et al. Information Security Continuous Monitoring. NIST SP 800-137 // Gaithersburg: NIST. 2011. URL: https://doi.org/10.6028/NIST.SP.800-137
18. Landauer M. et al. Deep Learning for Anomaly Detection in Log Data: A Survey // Machine Learning with Applications. 2023. Vol. 12. URL: https://doi.org/10.1016/j.mlwa.2022.100470
19. Verizon. 2023 Data Breach Investigations Report // Verizon Business. 2023. URL: https://www.verizon.com/business/resources/reports/dbir/
20. Pedregosa F. et al. Scikit-learn: Machine Learning in Python // Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830. URL: https://jmlr.org/papers/v12/pedregosa11a.html
21. SigmaHQ. Sigma — Generic Signature Format for SIEM Systems // GitHub. 2017. URL: https://github.com/SigmaHQ/sigma
22. OASIS Open. STIX 2.1 Specification // OASIS Standard. 2021. URL: https://docs.oasis-open.org/cti/stix/v2.1/stix-v2.1.html
23. Tiangolo S. FastAPI Framework Documentation // FastAPI. 2024. URL: https://fastapi.tiangolo.com
24. SQLAlchemy — The Database Toolkit for Python // SQLAlchemy Authors. 2024. URL: https://docs.sqlalchemy.org
25. Claude API Documentation // Anthropic PBC. 2024. URL: https://docs.anthropic.com

26. drain3 — Online Log Template Mining // logpai. GitHub. 2021. URL: https://github.com/logpai/drain3
27. Chuvakin A., Schmidt K., Phillips C. Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management // Syngress. 2012. 460 p. ISBN: 978-1597496353.
28. Liao H.-J. et al. Intrusion Detection System: A Comprehensive Review // Journal of Network and Computer Applications. 2013. Vol. 36, № 1. P. 16–24. URL: https://doi.org/10.1016/j.jnca.2012.09.004
29. Wazuh. Wazuh — Open Source Security Platform // Wazuh Inc. 2024. URL: https://wazuh.com
30. AlienVault. AlienVault OSSIM — Open Source SIEM // AT&T Cybersecurity. 2024. URL: https://cybersecurity.att.com/products/ossim
31. Lockheed Martin. Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains // Lockheed Martin Corporation. 2011. URL: https://lockheedmartin.com/content/dam/lockheed-martin/rms/documents/cyber/LM-White-Paper-Intel-Driven-Defense.pdf
32. Brown T. et al. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 1877–1901. URL: https://arxiv.org/abs/2005.14165
33. Python Software Foundation. asyncio — Asynchronous I/O // Python Docs. 2024. URL: https://docs.python.org/3/library/asyncio.html
34. He P. et al. An Evaluation Study on Log Parsing and Its Use in Log Mining // Proceedings of IEEE/IFIP DSN. 2016. P. 654–661. URL: https://ieeexplore.ieee.org/document/7579781
35. Microsoft. Microsoft Sentinel Documentation // Microsoft Corporation. 2024. URL: https://learn.microsoft.com/en-us/azure/sentinel/overview
36. Paxson V. Bro: A System for Detecting Network Intruders in Real-Time // Computer Networks. 1999. Vol. 31. P. 2435–2463. URL: https://doi.org/10.1016/S1389-1286(99)00112-7

ДОДАТКИ

Додаток А

Таблиця.А.1

Повна схема нормалізованої події NormalizedEvent

Поле	**Тип даних**	**Обов'язкове**	**Опис**
id	UUID	Так	Унікальний ідентифікатор події (генерується автоматично)
occurred_at	TIMESTAMPTZ	Так	Час виникнення події (UTC, з джерела або парсингу)
ingested_at	TIMESTAMPTZ	Так	Час отримання системою (UTC, завжди поточний)
source	TEXT	Так	Джерело логу: linux, windows, apache, docker
event_type	TEXT	Так	Тип події: auth.failure, auth.success, sudo.exec, process.create тощо
severity	SMALLINT	Так	Рівень критичності: 0 (debug) — 5 (critical)
source_ip	TEXT	Ні	IP-адреса ініціатора події
dest_ip	TEXT	Ні	IP-адреса цілі (для мережевих подій)
user_name	TEXT	Ні	Ім'я облікового запису, пов'язаного з подією
host	TEXT	Ні	FQDN або hostname хосту-джерела
process	TEXT	Ні	Назва процесу, що згенерував подію

Поле	Тип даних	Обов'язкове	Опис
template_id	TEXT	Ні	Хеш шаблону Drain3 (для ML-детекції)
anomaly_score	FLOAT	Ні	Оцінка аномальності IsolationForest [0.0, 1.0]
country	TEXT	Ні	Країна source_ip (GeoIP збагачення)
raw_message	TEXT	Так	Оригінальний рядок логу без змін
fields	JSONB	Ні	Додаткові поля специфічні для джерела (key-value)

Таблиця А.2

REST API endpoints веб-інтерфейсу

Метод	Endpoint	Параметри	Опис
GET	/api/summary	-	Зведена статистика: кількість подій, алертів, інцидентів, anomaly avg
GET	/events	search, event_types, severity_min, anomaly_min, from, to, sort_by, page	Список нормалізованих подій з фільтрацією та пагінацією
GET	/alerts	search, severity, status, tactic, risk_min, from, to, sort_by	Список алертів з фільтрацією, сортуванням та пагінацією
GET	/api/events/ hourly	-	Кількість подій за годинами за останні 24h (для графіку)

Метод	Endpoint	Параметри	Опис
GET	/api/anomaly/ timeseries	-	Anomaly score за годинами за останні 12h
GET	/api/clusters	-	Drain3 шаблони з кількістю подій (LogAI clusters)
GET	/api/forecast	-	Прогноз на наступні години: counts, upper/lower bounds, peak_hour
GET	/api/heatmap	user (опц.)	Теплова карта активності 7×24 (день тижня × година)
GET	/api/geo/markers	-	GeoIP дані для карти: IP, lat/lon, country, alert_count
POST	/api/explain-alert	alert (JSON)	SSE стрім: Claude пояснює алерт природною мовою
POST	/api/ingest/ webhook	source, raw_message, fields	Прийом логів від зовнішніх симуляторів (Docker контейнери)
GET	/api/rules	-	Список всіх завантажених правил з метаданими та статистикою
PATCH	/api/rules/{id}	enabled, condition.count, condition.window	Оновлення параметрів правила без перезапуску системи

Лістинг коду

anomaly_detector.py:

```
"""Anomaly detector with time-aware features and per-user behavioral baselines."""
import hashlib
import json
import logging
import math
import pickle
from dataclasses import dataclass, field
from datetime import datetime
from pathlib import Path
from typing import Dict, List, Optional, Set, Tuple

import numpy as np

from models import NormalizedEvent
from config import config

logger = logging.getLogger(__name__)

try:
    from sklearn.ensemble import IsolationForest
    from sklearn.preprocessing import StandardScaler
    _SKLEARN = True
except ImportError:
    logger.warning("scikit-learn not installed — anomaly detection disabled")
    _SKLEARN = False


# feature extraction

def _features(event: NormalizedEvent) -> List[float]:
    h = event.timestamp.hour
    wd = event.timestamp.weekday()
    tid = event.template_id or "unknown"
    tid_hash = int(hashlib.md5(tid.encode()).hexdigest()[:8], 16) % 1000
    return [
        math.sin(2 * math.pi * h / 24),
        math.cos(2 * math.pi * h / 24),
        1.0 if 8 <= h <= 19 else 0.0,
        1.0 if wd >= 5 else 0.0,
        float(len(event.raw_message)),
        float(len(event.raw_message.split())),
        float(tid_hash),
        float(event.severity),
    ]
# per-user profile

@dataclass
class UserProfile:
    user_name: str
    typical_hours: Set[int] = field(default_factory=set)
    typical_ips: Set[str] = field(default_factory=set)
    typical_processes: Set[str] = field(default_factory=set)
    event_count: int = 0
    deviation_count: int = 0

    def to_dict(self) -> dict:
        return {
            "user_name": self.user_name,
            "typical_hours": sorted(self.typical_hours),
            "typical_ips": sorted(self.typical_ips),
            "typical_processes": sorted(self.typical_processes),
            "event_count": self.event_count,
            "deviation_count": self.deviation_count,
        }

    @classmethod
    def from_dict(cls, d: dict) -> "UserProfile":
        p = cls(user_name=d["user_name"])
        p.typical_hours = set(d.get("typical_hours", []))
        p.typical_ips = set(d.get("typical_ips", []))
        p.typical_processes = set(d.get("typical_processes", []))
        p.event_count = d.get("event_count", 0)
        p.deviation_count = d.get("deviation_count", 0)
        return p


# main detector

class AnomalyDetector:
    """Score events using IsolationForest plus per-user behavioral adjustments."""

    def __init__(
        self,
        learning_mode: Optional[bool] = None,
        model_path: Optional[str] = None,
    ) -> None:
        self.learning_mode: bool = (
            learning_mode if learning_mode is not None else config.learning_mode
        )
        self.model_path = Path(model_path or config.model_path)
        self.threshold: float = config.anomaly_threshold

        self._model: Optional[IsolationForest] = None
        self._scaler: Optional[StandardScaler] = None
        self._buffer: List[List[float]] = []
        self._min_fit_samples = 50
        self._df_min: float = -0.5
        self._df_max: float = 0.5
        self._profiles: Dict[str, UserProfile] = {}

        if self.model_path.exists():
            self.load_model()

    # public API

    def fit(self, events: List[NormalizedEvent]) -> None:
        if not _SKLEARN:
            return
        X_raw = [_features(e) for e in events]
        self._fit_array(X_raw)

    def predict(self, event: NormalizedEvent) -> float:
        """Return anomaly score in [0, 1]. Stores reasons in event.fields."""
        if not _SKLEARN:
            return 0.0

        feats = _features(event)

        if self.learning_mode:
            self._buffer.append(feats)
            if len(self._buffer) >= self._min_fit_samples and self._model is None:
                self._fit_array(self._buffer)
                self._buffer.clear()
            if self._model is None:
                return 0.0

        if self._model is None or self._scaler is None:
            return 0.0

        try:
            X = np.array([feats])
            X_scaled = self._scaler.transform(X)
            raw = float(self._model.decision_function(X_scaled)[0])
            span = self._df_max - self._df_min
            score = float(max(0.0, min(1.0, (self._df_max - raw) / span)))
        except Exception as exc:
            logger.error("Predict error: %s", exc)
            return 0.0

        adjustment, reasons = self._behavioral_check(event)
        final_score = min(1.0, score + adjustment)

        if reasons:
            event.fields["anomaly_reasons"] = reasons
        elif "anomaly_reasons" in event.fields:
            del event.fields["anomaly_reasons"]

        if event.user_name and reasons:
            p = self._get_or_create_profile(event.user_name)
            p.deviation_count += 1

        self._update_profile(event, final_score)

        return final_score

    def save_model(self) -> None:
        if self._model is None:
            logger.warning("No model to save")
            return

        self.model_path.parent.mkdir(parents=True, exist_ok=True)
        with open(self.model_path, "wb") as fh:
            pickle.dump({
                "model": self._model,
                "scaler": self._scaler,
                "df_min": self._df_min,
                "df_max": self._df_max,
            }, fh)
        logger.info("Model saved → %s", self.model_path)

    def load_model(self) -> None:
        try:
            with open(self.model_path, "rb") as fh:
                data = pickle.load(fh)
            self._model = data["model"]
            self._scaler = data["scaler"]
            self._df_min = data.get("df_min", -0.5)
            self._df_max = data.get("df_max", 0.5)
            logger.info("Model loaded ← %s", self.model_path)
        except Exception as exc:
            logger.error("Load model failed: %s", exc)

    # behavioral baseline

    def _behavioral_check(
        self, event: NormalizedEvent
    ) -> Tuple[float, List[str]]:
        user = event.user_name
        if not user:
```

```
        return 0.0, []

        profile =
self._get_or_create_profile(user)
        adjustment = 0.0
        reasons: List[str] = []
        h = event.timestamp.hour

        if profile.typical_hours and h not in
profile.typical_hours:
            adjustment += 0.15
            typical_str =
self._fmt_hours(profile.typical_hours)
            reasons.append(
                f"Unusual hour ({h:02d}:00,
typical: {typical_str})"
            )

        if (event.source_ip and
profile.typical_ips
                and event.source_ip not in
profile.typical_ips):
            adjustment += 0.2
            reasons.append(
                f"New source IP (first time seen
for {user})"
            )

        if (event.process and
profile.typical_processes
                and event.process not in
profile.typical_processes):
            adjustment += 0.1
            reasons.append(f"Unusual process:
{event.process}")

        return adjustment, reasons

    def _fmt_hours(self, hours: Set[int]) ->
str:
        if not hours:
            return "none"
        s = sorted(hours)
        if len(s) > 5:
            return f"{min(s):02d}:00–
{max(s):02d}:00"
        return ", ".join(f"{h:02d}:00" for h in
s)

    # profile management

    def _get_or_create_profile(self, user: str)
-> UserProfile:
        if user not in self._profiles:
            try:
                from storage import get_storage
                stored =
get_storage().get_user_profile(user)
                self._profiles[user] = stored or
UserProfile(user_name=user)
            except Exception:
                self._profiles[user] =
UserProfile(user_name=user)
        return self._profiles[user]

    def _update_profile(self, event:
NormalizedEvent, score: float) -> None:
        user = event.user_name
        if not user or score > 0.5:
            return
        profile =
self._get_or_create_profile(user)
        profile.event_count += 1

profile.typical_hours.add(event.timestamp.h
our)
        if event.source_ip:

profile.typical_ips.add(event.source_ip)
        if event.process:

profile.typical_processes.add(event.process
)
        if profile.event_count % 10 == 0:
            self._save_profile(profile)

    def _save_profile(self, profile:
UserProfile) -> None:
        try:
            from storage import get_storage

get_storage().save_user_profile(profile)
        except Exception as exc:
            logger.warning("Could not save
user profile for %s: %s",
                           profile.user_name, exc)

    # internals

    def _fit_array(self, X_raw:
List[List[float]]) -> None:
        if len(X_raw) < 10:
            logger.warning("Too few samples
(%d) to fit detector", len(X_raw))
            return
        X = np.array(X_raw)
        self._scaler = StandardScaler()
        X_scaled =
self._scaler.fit_transform(X)
        self._model = IsolationForest(
            n_estimators=100,
contamination=0.05, random_state=42
        )
        self._model.fit(X_scaled)
        df_scores =
self._model.decision_function(X_scaled)
        self._df_min = float(df_scores.min())
        self._df_max = float(df_scores.max())
        logger.info(
            "AnomalyDetector fitted on %d
samples  df=[%.3f, %.3f]",
            len(X_raw), self._df_min,
self._df_max,
        )


_detector: Optional[AnomalyDetector] =
None


def get_detector() -> AnomalyDetector:
    global _detector
    if _detector is None:
        _detector = AnomalyDetector()
    return _detector
```

rules_engine.py:

```
import logging
import re
from collections import defaultdict, deque
from datetime import datetime, timezone
from pathlib import Path
from typing import Any, Dict, List,
Optional, Tuple
import uuid

import yaml

from models import Alert,
NormalizedEvent
from config import config
from mitre_mapper import get_mapper

logger = logging.getLogger(__name__)


class Rule:
    def __init__(self, data: Dict[str, Any],
yaml_path: Optional[Path] = None) ->
None:
        self.id: str = data["id"]
        self.name: str = data["name"]
        self.severity: int = int(data["severity"])
        self.description: str =
data.get("description", "")
        self.enabled: bool =
data.get("enabled", True)
        self.yaml_path = yaml_path
        self._raw_data: Dict[str, Any] =
dict(data)

        match = data.get("match", {})
        self._match_event_types: List[str] =
match.get("event_type", [])
        self._match_fields: Dict[str, Any] =
match.get("fields", {})
        hor = match.get("hour_not_in_range")
        self._hour_not_in_range:
Optional[Tuple[int, int]] = tuple(hor) if hor
else None  # type: ignore[assignment]

        cond = data.get("condition", {})
        self.threshold: int =
int(cond.get("count", 1))
        self.window_seconds: int =
int(cond.get("window_seconds", 60))
        self.group_by: List[str] =
cond.get("group_by", ["source_ip"])

        self._distinct_field: Optional[str] =
cond.get("distinct")
        self._sequence: List[str] =
cond.get("sequence", [])

        # count/distinct: deque[(unix_ts,
event_id)] or deque[(unix_ts, event_id,
field_val)]
        # sequence: deque[(unix_ts,
event_type)]
        self._windows: Dict[str, deque] =
defaultdict(deque)

        self.trigger_count: int = 0
        self.last_triggered: Optional[datetime]
= None

    # matching

    def _matches(self, event:
NormalizedEvent) -> bool:
        if self._match_event_types and
event.event_type not in
self._match_event_types:
            return False
        for field, value in
self._match_fields.items():
            ev_val = getattr(event, field, None)
or event.fields.get(field)
            if isinstance(value, list):
                if ev_val not in value:
                    return False
            elif isinstance(value, str) and
value.startswith("~"):
                if not re.search(value[1:],
str(ev_val or "")):
                    return False
            else:
                if ev_val != value:
                    return False
        if self._hour_not_in_range:
            lo, hi = self._hour_not_in_range
            h = event.timestamp.hour
            if lo <= h <= hi:
                return False
        return True

    def _group_key(self, event:
NormalizedEvent) -> str:
        parts = []
        for field in self.group_by:
            val = getattr(event, field, None) or
event.fields.get(field, "unknown")
            parts.append(str(val))
        return "|".join(parts)

    # evaluation

    def evaluate(self, event:
NormalizedEvent) -> Optional[Alert]:
        if not self.enabled or not
self._matches(event):
            return None
        key = self._group_key(event)
        now = event.timestamp.timestamp()
        if self._sequence:
            return self._eval_sequence(event,
key, now)
        if self._distinct_field:
            return self._eval_distinct(event, key,
now)
        return self._eval_count(event, key,
now)
```

```
    def _eval_count(self, event:
NormalizedEvent, key: str, now: float) ->
Optional[Alert]:
        window = self._windows[key]
        window.append((now, event.id))
        cutoff = now - self.window_seconds
        while window and window[0][0] <
cutoff:
            window.popleft()
        if len(window) >= self.threshold:
            event_ids = [eid for _, eid in
window]
            window.clear()
            return self._make_alert(event, key,
len(event_ids), event_ids)
        return None

    def _eval_distinct(self, event:
NormalizedEvent, key: str, now: float) ->
Optional[Alert]:
        fval = str(
            getattr(event, self._distinct_field,
None)
            or
event.fields.get(self._distinct_field,
"unknown")
        )
        window = self._windows[key]
        window.append((now, event.id, fval))
        cutoff = now - self.window_seconds
        while window and window[0][0] <
cutoff:
            window.popleft()
        distinct_vals = {entry[2] for entry in
window}
        if len(distinct_vals) >= self.threshold:
            event_ids = [entry[1] for entry in
window]
            window.clear()
            return self._make_alert(event, key,
len(distinct_vals), event_ids)
        return None

    def _eval_sequence(self, event:
NormalizedEvent, key: str, now: float) ->
Optional[Alert]:
        window = self._windows[key]
        window.append((now,
event.event_type))
        cutoff = now - self.window_seconds
        while window and window[0][0] <
cutoff:
            window.popleft()
        if
self._sequence_satisfied(list(window)):
            window.clear()
            return self._make_alert(event, key,
len(self._sequence), [event.id])
        return None

    def _sequence_satisfied(self, events:
List[Tuple[float, str]]) -> bool:
        step = 0
        for _ts, etype in events:
            if etype == self._sequence[step]:
                step += 1
                if step == len(self._sequence):
                    return True
        return False

    def _make_alert(
        self, event: NormalizedEvent, key: str,
count: int, event_ids: List[str]
    ) -> Alert:
        self.trigger_count += 1
        self.last_triggered =
datetime.now(timezone.utc)
        desc = f"{self.description or
self.name} [key={key}, count={count}]"
        mitre = get_mapper().map(self.id) or
{}
        return Alert(
            id=str(uuid.uuid4()),
            timestamp=self.last_triggered,
            rule_id=self.id,
            rule_name=self.name,
            severity=self.severity,
            event_ids=event_ids,
            source_ip=event.source_ip,
            user_name=event.user_name,
            description=desc,
            status="open",
            risk_score=0.0,
            mitre_id=mitre.get("id"),
            mitre_name=mitre.get("name"),
            mitre_tactic=mitre.get("tactic"),
        )

    # serialisation & persistence

    def to_dict(self) -> Dict[str, Any]:
        return {
            "id": self.id,
            "name": self.name,
            "severity": self.severity,
            "enabled": self.enabled,
            "description": self.description,
            "trigger_count": self.trigger_count,
            "last_triggered":
self.last_triggered.isoformat() if
self.last_triggered else None,
            "match_filters": {
                "event_type":
self._match_event_types,
                "fields": self._match_fields,
                "hour_not_in_range":
list(self._hour_not_in_range) if
self._hour_not_in_range else None,
            },
            "condition": {
                "count": self.threshold,
                "window_seconds":
self.window_seconds,
                "group_by": self.group_by,
                "distinct": self._distinct_field,
                "sequence": self._sequence or
None,
            },
        }

    def save(self) -> None:
        if not self.yaml_path:
            return
        self._raw_data["enabled"] =
self.enabled
        cond =
self._raw_data.setdefault("condition", {})
        cond["count"] = self.threshold
        cond["window_seconds"] =
self.window_seconds
        with open(self.yaml_path, "w") as fh:
            yaml.dump(self._raw_data, fh,
default_flow_style=False,
allow_unicode=True)
        logger.info("Saved rule %s → %s",
self.id, self.yaml_path)


class RulesEngine:
    def __init__(self, rules_dir: Optional[str]
= None) -> None:
        self.rules_dir = Path(rules_dir or
config.rules_dir)
        self.rules: List[Rule] = []
        self.load_rules()

    def load_rules(self) -> None:
        self.rules = []
        if not self.rules_dir.exists():
            logger.warning("Rules directory not
found: %s", self.rules_dir)
            return
        for yaml_path in
sorted(self.rules_dir.glob("*.yaml")):
            try:
                with open(yaml_path) as fh:
                    data = yaml.safe_load(fh)
                items = data if isinstance(data,
list) else [data]
                for item in items:
                    self.rules.append(Rule(item,
yaml_path=yaml_path))
                logger.info("Loaded %d rule(s)
from %s", len(items), yaml_path.name)
            except Exception as exc:
                logger.error("Failed to load %s:
%s", yaml_path, exc)
        logger.info("Rules engine ready — %d
rule(s) total", len(self.rules))

    def evaluate(self, event:
NormalizedEvent) -> List[Alert]:
        alerts: List[Alert] = []
        for rule in self.rules:
            try:
                alert = rule.evaluate(event)
                if alert:
                    alerts.append(alert)
            except Exception as exc:
                logger.error("Rule %s error: %s",
rule.id, exc)
        return alerts

    # management API

    def get_rule(self, rule_id: str) ->
Optional[Rule]:
        return next((r for r in self.rules if r.id
== rule_id), None)

    def toggle(self, rule_id: str) ->
Optional[Rule]:
        rule = self.get_rule(rule_id)
        if rule:
            rule.enabled = not rule.enabled
            rule.save()
        return rule

    def update(self, rule_id: str, count:
Optional[int] = None,
               window_seconds: Optional[int] =
None) -> Optional[Rule]:
        rule = self.get_rule(rule_id)
        if rule:
            if count is not None:
                rule.threshold = int(count)
            if window_seconds is not None:
                rule.window_seconds =
int(window_seconds)
            rule.save()
        return rule

    def create(self, data: Dict[str, Any]) ->
Rule:
        rule_id = data.get("id") or
str(uuid.uuid4())
        data["id"] = rule_id
        yaml_path = self.rules_dir /
f"{rule_id}.yaml"
        with open(yaml_path, "w") as fh:
            yaml.dump(data, fh,
default_flow_style=False,
allow_unicode=True)
        new_rule = Rule(data,
yaml_path=yaml_path)
        self.rules.append(new_rule)
        logger.info("Created new rule %s",
rule_id)
        return new_rule

    def test_rule(self, rule_id: str, limit: int =
1000) -> Dict[str, Any]:
        """Run rule against the last `limit`
stored events; returns match stats."""
        rule = self.get_rule(rule_id)
        if not rule:
            return {"error": "Rule not found"}
        from storage import get_storage
        events =
get_storage().get_events(limit=limit)
        saved_enabled = rule.enabled
        rule.enabled = True
        fired: List[str] = []
        try:
            for event in events:
                alert = rule.evaluate(event)
                if alert:
                    fired.append(alert.description)
        finally:
            rule.enabled = saved_enabled
        return {
            "rule_id": rule_id,
            "events_tested": len(events),
            "alerts_fired": len(fired),
```

```
            "samples": fired[:5],
        }


_engine: Optional[RulesEngine] = None


def get_rules_engine() -> RulesEngine:
    global _engine
    if _engine is None:
        _engine = RulesEngine()
    return _engine
```

parser.py:

```
import calendar
import hashlib
import logging
import re
from datetime import datetime, timedelta,
timezone
from typing import Any, Dict, List,
Optional
import uuid

from models import NormalizedEvent
from geoip_enricher import get_enricher

logger = logging.getLogger(__name__)

try:
    from pygrok import Grok
    _PYGROK = True
except ImportError:
    logger.warning("pygrok not available —
Grok patterns disabled")
    _PYGROK = False

try:
    from drain3 import TemplateMiner
    from drain3.template_miner_config
import TemplateMinerConfig
    _DRAIN = True
except ImportError:
    logger.warning("drain3 not available —
template mining will use fallback hashing")
    _DRAIN = False


# event-type detection rules

EVENT_TYPE_PATTERNS = [
    (r"Failed password", "auth.failure"),
    (r"Accepted password", "auth.success"),
    (r"Accepted publickey", "auth.success"),
    (r"sudo:.*COMMAND", "sudo.exec"),
    (r"su: session opened", "su.session"),
    (r"FAILED su", "su.failure"),
    (r"session opened", "session.open"),
    (r"session closed", "session.close"),
    (r"Invalid user", "auth.invalid_user"),
    (r"Connection (closed|reset)",
"connection.close"),
    (r"Disconnecting",
"connection.disconnect"),
]

SEVERITY_MAP: Dict[str, int] = {
    "auth.failure": 3,
    "auth.success": 1,
    "sudo.exec": 3,
    "su.session": 2,
    "su.failure": 4,
    "session.open": 1,
    "session.close": 1,
    "auth.invalid_user": 3,
    "connection.close": 1,
    "connection.disconnect": 1,
    # Windows event types
    "windows.auth.success": 1,
    "windows.auth.failure": 3,
    "windows.privilege": 4,
    "windows.process.create": 2,
    "windows.scheduled_task": 4,
    "windows.service_install": 4,
    # Apache / HTTP
    "http.request": 1,
    "http.client_error": 2,
    "http.server_error": 3,
    "http.error": 3,
    "unknown": 2,
}

# Windows EventID → event_type
_WIN_EVENT_MAP: Dict[int, str] = {
    4624: "windows.auth.success",
    4625: "windows.auth.failure",
    4672: "windows.privilege",
    4688: "windows.process.create",
    4698: "windows.scheduled_task",
    7045: "windows.service_install",
}

# regexes

_SYSLOG_RE = re.compile(

r"^(?P<month>\w{3})\s+(?P<day>\d+)\s+(
?P<time>\d{2}:\d{2}:\d{2})\s+"

r"(?P<host>\S+)\s+(?P<process>\w+)(?:\[(?
P<pid>\d+)\])?:\s+(?P<message>.+)$"
)

_MONTH_ABBRS = {abbr.lower(): i for i,
abbr in enumerate(calendar.month_abbr) if
abbr}


def _parse_syslog_ts(month: str, day: str,
time_str: str) -> Optional[datetime]:
    """Parse syslog timestamp (no year);
assume current year, step back if future."""
    try:
        month_num =
_MONTH_ABBRS[month.lower()]
        h, m, s = (int(x) for x in
time_str.split(":"))
        now = datetime.now(timezone.utc)
        dt = now.replace(
            month=month_num, day=int(day),
            hour=h, minute=m, second=s,
microsecond=0,
        )
        # Timestamps more than 1 h in the
future must be from the previous year
        if dt > now + timedelta(hours=1):
            dt = dt.replace(year=now.year - 1)
        return dt
    except Exception:
        return None

_SSH_AUTH_RE = re.compile(
    r"(?:Failed|Accepted)
(?:password|publickey) for (?:invalid user
)?(\S+)"
    r" from (\S+) port (\d+)"
)
_SUDO_RE = re.compile(

r"(\S+)\s+:\s+TTY=\S+\s+;\s+PWD=\S+\s+
;\s+USER=(\S+)\s+;\s+COMMAND=(.+)"
)
_SU_RE = re.compile(
    r"(?:"
    r"session opened for user \S+ by (\w+)\("
# su.session: "by alice(uid=…)"
    r"|"
    r"FAILED su for \S+ by (\w+)"           #
su.failure: "by bob"
    r")"
)

# Grok patterns

_GROK_PATTERNS = {
    "syslog":
"%{SYSLOGTIMESTAMP:timestamp}
%{HOSTNAME:host}
%{WORD:process}"
            "(?:\\[%{NUMBER:pid}\\])?:
%{GREEDYDATA:message}",
    "apache":
"%{COMBINEDAPACHELOG}",
}

class LogParser:
    def __init__(self) -> None:
        self._groks: Dict[str, Any] = {}
        self._drain: Optional[Any] = None
        self._init_grok()
        self._init_drain()

    def _init_grok(self) -> None:
        if not _PYGROK:
            return
        for name, pattern in
_GROK_PATTERNS.items():
            try:
                self._groks[name] =
Grok(pattern)
            except Exception as exc:
                logger.warning("Grok compile
failed [%s]: %s", name, exc)

    def _init_drain(self) -> None:
        if not _DRAIN:
            return
        try:
            cfg = TemplateMinerConfig()
            cfg.drain_sim_th = 0.5
            cfg.drain_depth = 4
            self._drain =
TemplateMiner(config=cfg)
        except Exception as exc:
            logger.warning("Drain3 init failed:
%s", exc)

    # private helpers

    def _detect_event_type(self, message:
str) -> str:
        for pattern, etype in
EVENT_TYPE_PATTERNS:
            if re.search(pattern, message):
                return etype
        return "unknown"

    def _parse_syslog_header(self, line: str) -
> Dict[str, Any]:
        m = _SYSLOG_RE.match(line.strip())
        if not m:
            return {}
        return {
            "month": m.group("month"),
            "day": m.group("day"),
            "time": m.group("time"),
            "host": m.group("host"),
            "process": m.group("process"),
            "pid": m.group("pid"),
            "message": m.group("message"),
        }

    def _parse_ssh_auth(self, message: str) -
> Dict[str, Any]:
        m =
_SSH_AUTH_RE.search(message)
        if m:
            return {"user": m.group(1), "src_ip":
m.group(2), "port": m.group(3)}
        return {}

    def _parse_sudo(self, message: str) ->
Dict[str, Any]:
        m = _SUDO_RE.search(message)
        if m:
            return {"user": m.group(1),
"run_as": m.group(2), "command":
m.group(3)}
        return {}

    def _parse_su(self, message: str) ->
Dict[str, Any]:
        m = _SU_RE.search(message)
        if m:
            user = m.group(1) or m.group(2)
            return {"user": user}
        return {}

    def _get_template_id(self, message: str) -
> str:
        if self._drain is not None:
            try:
                result =
```

```
self._drain.add_log_message(message)
            if result and
result.get("cluster_id"):
                return
f"drain_{result['cluster_id']}"
        except Exception:
            pass
    return "hash_" +
hashlib.md5(message[:60].encode()).hexdig
est()[:8]

  def _try_grok(self, line: str) -> Dict[str,
Any]:
    for grok in self._groks.values():
        try:
            hit = grok.match(line)
            if hit:
                return {k: v for k, v in
hit.items() if v is not None}
        except Exception:
            pass
    return {}

  # webhook source parsers

  def _parse_windows_event(self, data:
dict) -> NormalizedEvent:
    event_id = int(data.get("event_id", 0))
    event_type =
_WIN_EVENT_MAP.get(event_id,
"unknown")
    source_ip = data.get("source_ip") or
data.get("src_ip")
    user_name = data.get("user") or
data.get("username")
    host = data.get("host")
    raw = data.get("raw_message",
f"Windows EventID {event_id}")
    ts_str = data.get("timestamp")
    try:
        ts = datetime.fromisoformat(ts_str)
if ts_str else datetime.now(timezone.utc)
    except ValueError:
        ts = datetime.now(timezone.utc)

    geo = get_enricher().enrich(source_ip)
    fields = {k: v for k, v in data.items()
              if k not in ("source",
"raw_message", "timestamp")}
    fields["event_id"] = event_id

    return NormalizedEvent(
        id=str(uuid.uuid4()),
        timestamp=ts,
        source="webhook:windows",
        event_type=event_type,

severity=SEVERITY_MAP.get(event_type,
2),
        source_ip=source_ip,
        user_name=user_name,
        host=host,
        process=data.get("process_name"),

template_id=self._get_template_id(raw),
        anomaly_score=None,
        raw_message=raw,
        fields=fields,
        country=geo.get("country"),

country_code=geo.get("country_code"),
        city=geo.get("city"),
        latitude=geo.get("latitude"),
        longitude=geo.get("longitude"),
        org=geo.get("org"),
    )

  _APACHE_CLF_RE = re.compile(
    r'(?P<ip>\S+) \S+ \S+
\[(?P<ts>[^\]]+)\] '
    r'"(?P<method>\S+) (?P<path>\S+)
[^"]+" '
    r'(?P<status>\d{3}) (?P<size>\S+)'
  )

  def _parse_apache_access(self, data:
dict) -> NormalizedEvent:
    source_ip = data.get("source_ip") or
data.get("ip")
    status = int(data.get("status_code",
200))
    method = data.get("method", "GET")
    path = data.get("path", "/")
    host = data.get("host")
    raw = data.get("raw_message",
f'{source_ip} - GET {path} {status}')
    ts_str = data.get("timestamp")
    try:
        ts = datetime.fromisoformat(ts_str)
if ts_str else datetime.now(timezone.utc)
    except ValueError:
        ts = datetime.now(timezone.utc)

    if status >= 500:
        event_type = "http.server_error"
    elif status >= 400:
        event_type = "http.client_error"
    else:
        event_type = "http.request"

    geo = get_enricher().enrich(source_ip)
    fields = {k: v for k, v in data.items()
              if k not in ("source",
"raw_message", "timestamp")}

    return NormalizedEvent(
        id=str(uuid.uuid4()),
        timestamp=ts,
        source="webhook:apache",
        event_type=event_type,

severity=SEVERITY_MAP.get(event_type,
1),
        source_ip=source_ip,
        user_name=None,
        host=host,
        process="apache2",

template_id=self._get_template_id(f"{meth
od} {path} {status}"),
        anomaly_score=None,
        raw_message=raw,
        fields=fields,
        country=geo.get("country"),

country_code=geo.get("country_code"),
        city=geo.get("city"),
        latitude=geo.get("latitude"),
        longitude=geo.get("longitude"),
        org=geo.get("org"),
    )

  def _parse_apache_error(self, data: dict)
-> NormalizedEvent:
    source_ip = data.get("source_ip") or
data.get("ip")
    host = data.get("host")
    raw = data.get("raw_message",
data.get("message", "Apache error"))
    ts_str = data.get("timestamp")
    try:
        ts = datetime.fromisoformat(ts_str)
if ts_str else datetime.now(timezone.utc)
    except ValueError:
        ts = datetime.now(timezone.utc)

    level = data.get("level",
"error").lower()
    sev = 4 if level in ("crit", "alert",
"emerg") else 3

    geo = get_enricher().enrich(source_ip)
    fields = {k: v for k, v in data.items()
              if k not in ("source",
"raw_message", "timestamp")}

    return NormalizedEvent(
        id=str(uuid.uuid4()),
        timestamp=ts,
        source="webhook:apache",
        event_type="http.error",
        severity=sev,
        source_ip=source_ip,
        user_name=None,
        host=host,
        process="apache2",
template_id=self._get_template_id(raw),
        anomaly_score=None,
        raw_message=raw,
        fields=fields,
        country=geo.get("country"),

country_code=geo.get("country_code"),
        city=geo.get("city"),
        latitude=geo.get("latitude"),
        longitude=geo.get("longitude"),
        org=geo.get("org"),
    )

  def _parse_linux_ssh(self, data: dict) ->
NormalizedEvent:
    """Parse a webhook payload that
wraps a syslog-formatted raw_message."""
    raw = data.get("raw_message", "")
    source_ip_hint = data.get("source_ip")
    ts_str = data.get("timestamp")
    try:
        ts = datetime.fromisoformat(ts_str)
if ts_str else None
    except ValueError:
        ts = None
    ev = self.parse(raw,
source="webhook:linux-ssh")
    if ts:
        ev.timestamp = ts
    if source_ip_hint and not ev.source_ip:
        ev.source_ip = source_ip_hint
    ev.host = data.get("host") or ev.host
    return ev

  def parse_webhook(self, data: dict) ->
NormalizedEvent:
    """Route a JSON webhook payload to
the correct sub-parser based on 'source'."""
    src = str(data.get("source", "")).lower()
    if src == "windows":
        return
self._parse_windows_event(data)
    if src == "apache":
        log_type = str(data.get("log_type",
"access")).lower()
        if log_type == "error":
            return
self._parse_apache_error(data)
        return
self._parse_apache_access(data)
    if src in ("linux-ssh", "linux"):
        return self._parse_linux_ssh(data)
    raw = data.get("raw_message",
str(data))
    return self.parse(raw,
source=f"webhook:{src}" if src else
"webhook")

  # public API

  def parse(self, raw_line: str, source: str =
"unknown") -> NormalizedEvent:
    fields: Dict[str, Any] = {}

    syslog =
self._parse_syslog_header(raw_line)
    fields.update(syslog)

    message = syslog.get("message",
raw_line)
    host = syslog.get("host")
    process = syslog.get("process")

    # Match on the full raw line so
"sudo:.*COMMAND" can see the process
prefix
    event_type =
self._detect_event_type(raw_line)

fields.update(self._parse_ssh_auth(message
))

fields.update(self._parse_sudo(message))
    fields.update(self._parse_su(message))

    if _PYGROK:
```

```
fields.update(self._try_grok(raw_line))

        source_ip = fields.get("src_ip") or
fields.get("source_ip")
        user_name = fields.get("user") or
fields.get("username")
        template_id =
self._get_template_id(message)

        event_ts = _parse_syslog_ts(
            syslog.get("month", ""),
            syslog.get("day", ""),
            syslog.get("time", ""),
        ) or datetime.now(timezone.utc)

        geo = get_enricher().enrich(source_ip)

        return NormalizedEvent(
            id=str(uuid.uuid4()),
            timestamp=event_ts,
            source=source,
            event_type=event_type,

severity=SEVERITY_MAP.get(event_type,
2),
            source_ip=source_ip,
            user_name=user_name,
            host=host,
            process=process,
            template_id=template_id,
            anomaly_score=None,
            raw_message=raw_line,
            fields=fields,
            country=geo.get("country"),

country_code=geo.get("country_code"),
            city=geo.get("city"),
            latitude=geo.get("latitude"),
            longitude=geo.get("longitude"),
            org=geo.get("org"),
        )

    def parse_batch(self, lines: List[str],
source: str = "unknown") ->
List[NormalizedEvent]:
        return [self.parse(line, source) for line
in lines if line.strip()]


_parser: Optional[LogParser] = None


def get_parser() -> LogParser:
    global _parser
    if _parser is None:
        _parser = LogParser()
    return _parser
```

rules/brute_force_ssh.yaml:

```
condition:
  count: 5
  group_by:
  - source_ip
  window_seconds: 60
description: 'More than 5 failed SSH
authentication attempts from a single IP
within
  60 seconds.

  '
enabled: true
id: brute_force_ssh
match:
  event_type:
  - auth.failure
  - auth.invalid_user
name: SSH Brute Force Attack
severity: 4
```

correlation_engine.py:

```
import logging
from collections import defaultdict, deque
from datetime import datetime, timezone
from typing import Dict, List, Optional, Set
import uuid

from models import Alert, Incident
from config import config

logger = logging.getLogger(__name__)

# Chains: if all required rule_ids appear
within the correlation window → Incident
_CHAINS = [
    {
        "id": "lateral_movement",
        "name": "Potential Lateral
Movement",
        "required_rules": {"brute_force_ssh",
"privilege_escalation"},
        "severity": 5,
        "description": "Brute-force followed
by privilege escalation",
    },
    {
        "id": "account_compromise",
        "name": "Account Compromise",
        "required_rules": {"brute_force_ssh",
"suspicious_auth"},
        "severity": 4,
        "description": "Brute-force followed
by suspicious authentication",
    },
]


class CorrelationEngine:
    def __init__(self, window_seconds:
Optional[int] = None) -> None:
        self.window_seconds =
window_seconds or
config.correlation_window
        self.min_alerts =
config.correlation_min_alerts

        self._global: deque = deque()
        self._by_ip: Dict[str, deque] =
defaultdict(deque)
        # track which chains have already fired
to avoid duplicate incidents
        self._fired_chains: Set[str] = set()

    def process_alert(self, alert: Alert) ->
Optional[Incident]:
        now = alert.timestamp.timestamp()
        self._global.append((now, alert))
        self._evict(self._global, now)

        if alert.source_ip:
            ip_q = self._by_ip[alert.source_ip]
            ip_q.append((now, alert))
            self._evict(ip_q, now)
            incident =
self._check_ip_storm(alert.source_ip, ip_q)
            if incident:
                return incident

        return self._check_chains()

    # internal

    def _evict(self, q: deque, now: float) ->
None:
        cutoff = now - self.window_seconds
        while q and q[0][0] < cutoff:
            q.popleft()

    def _check_ip_storm(self, ip: str, q:
deque) -> Optional[Incident]:
        if len(q) < self.min_alerts:
            return None
        alert_list = [a for _, a in q]
        max_sev = max(a.severity for a in
alert_list)
        if max_sev < 3:
            return None
        rule_ids = {a.rule_id for a in alert_list}
        desc = f"Alert storm from {ip}:
{len(alert_list)} alerts, rules={rule_ids}"
        incident = Incident(
            id=str(uuid.uuid4()),

timestamp=datetime.now(timezone.utc),
            alert_ids=[a.id for a in alert_list],
            severity=min(max_sev + 1, 5),
            description=desc,
            status="open",
            risk_score=0.0,
        )
        q.clear()  # reset after incident to avoid
repeated triggers
        logger.warning("Incident (IP storm):
%s", desc)
        return incident

    def _check_chains(self) ->
Optional[Incident]:
        present_rules = {a.rule_id for _, a in
self._global}
        for chain in _CHAINS:
            cid = chain["id"]
            if cid in self._fired_chains:
                continue
            if
chain["required_rules"].issubset(present_rul
es):
                matching = [a for _, a in
self._global if a.rule_id in
chain["required_rules"]]
                self._fired_chains.add(cid)
                incident = Incident(
                    id=str(uuid.uuid4()),

timestamp=datetime.now(timezone.utc),
                    alert_ids=[a.id for a in
matching],
                    severity=chain["severity"],

description=chain["description"],
                    status="open",
                    risk_score=0.0,
                )
                logger.warning("Incident (chain
%s): %s", cid, chain["description"])
                return incident
        return None


_engine: Optional[CorrelationEngine] =
None


def get_correlation_engine() ->
CorrelationEngine:
    global _engine
    if _engine is None:
        _engine = CorrelationEngine()
    return _engine
```

risk_scorer.py:

```
from typing import Optional

from models import Alert,
NormalizedEvent

_SEVERITY_BASE: dict = {0: 0.0, 1: 0.1,
2: 0.25, 3: 0.5, 4: 0.75, 5: 1.0}

_RULE_WEIGHTS: dict = {
    "brute_force_ssh": 0.80,
    "privilege_escalation": 0.90,
    "suspicious_auth": 0.60,
}

_OFF_HOURS: set = set(range(0, 6)) | {22,
23}


def calculate_risk_score(
    alert: Alert,
    triggering_event:
Optional[NormalizedEvent] = None,
) -> float:
    sev_score =
_SEVERITY_BASE.get(alert.severity, 0.5)
    rule_score =
_RULE_WEIGHTS.get(alert.rule_id, 0.5)
    score = (sev_score + rule_score) / 2.0

    if triggering_event:
        if triggering_event.anomaly_score and
triggering_event.anomaly_score > 0.7:
            score = min(1.0, score * 1.3)
        if triggering_event.timestamp.hour in
```

```
_OFF_HOURS:
            score = min(1.0, score * 1.2)

    # Small linear bonus for alert volume
(capped at +0.15)
    score = min(1.0, score + min(0.15,
len(alert.event_ids) * 0.01))
    return round(score, 3)


class RiskScorer:
    def score_alert(
        self,
        alert: Alert,
        event: Optional[NormalizedEvent] =
None,
    ) -> float:
        return calculate_risk_score(alert,
event)
```

evaluation/metrics.py:

```
"""Evaluation metrics for the SIEM
detection pipeline."""
import sys
from pathlib import Path

sys.path.insert(0,
str(Path(__file__).parent.parent))

import random
from typing import Dict, List, Optional,
Tuple

import numpy as np

random.seed(42)
np.random.seed(42)


# core metrics

def evaluate_detector(
    y_true: List[int],
    y_pred: List[int],
    y_scores: Optional[List[float]] = None,
) -> Dict[str, object]:
    """Compute classification metrics. Adds
AUC-ROC if y_scores is provided."""
    yt = np.asarray(y_true, dtype=int)
    yp = np.asarray(y_pred, dtype=int)

    TP = int(np.sum((yp == 1) & (yt == 1)))
    FP = int(np.sum((yp == 1) & (yt == 0)))
    TN = int(np.sum((yp == 0) & (yt == 0)))
    FN = int(np.sum((yp == 0) & (yt == 1)))

    precision = TP / (TP + FP) if (TP + FP) >
0 else 0.0
    recall = TP / (TP + FN) if (TP + FN) > 0
else 0.0
    f1 = (2 * precision * recall / (precision +
recall)
          if (precision + recall) > 0 else 0.0)
    fpr = FP / (FP + TN) if (FP + TN) > 0
else 0.0
    accuracy = (TP + TN) / len(yt) if len(yt)
> 0 else 0.0

    result: Dict[str, object] = {
        "TP": TP,
        "FP": FP,
        "TN": TN,
        "FN": FN,
        "Precision": round(precision, 4),
        "Recall": round(recall, 4),
        "F1": round(f1, 4),
        "FPR": round(fpr, 4),
        "Accuracy": round(accuracy, 4),
    }

    if y_scores is not None:
        try:
            from sklearn.metrics import
roc_auc_score
            result["AUC_ROC"] =
round(float(roc_auc_score(yt, y_scores)), 4)
        except Exception:
            pass

    return result

def confusion_matrix_str(metrics: Dict[str,
object]) -> str:
    TP, FP = metrics["TP"], metrics["FP"]
    FN, TN = metrics["FN"], metrics["TN"]
    return (
        f"              Predicted\n"
        f"              Pos    Neg\n"
        f"Actual Pos  [ {TP:4d}   {FN:4d} ]\n"
        f"       Neg  [ {FP:4d}   {TN:4d} ]\n"
    )
# comparison harness

def compare_approaches(
    dataset: List[Tuple[str, int]],
) -> "pd.DataFrame":  # type: ignore[name-
defined]
    """Run both approaches over the dataset
and return a comparison DataFrame."""
    import pandas as pd
    from parser import LogParser
    from rules_engine import RulesEngine
    from anomaly_detector import
AnomalyDetector

    parser = LogParser()
    engine = RulesEngine()
    det =
AnomalyDetector(learning_mode=False)

    lines = [l for l, _ in dataset]
    labels = [lb for _, lb in dataset]

    events = [parser.parse(line,
source="eval") for line in lines]

    split = len(events) // 2
    det.fit(events[:split])

    rules_preds: List[int] = []
    anomaly_scores: List[float] = []

    for event in events:
        alerts = engine.evaluate(event)
        rules_preds.append(1 if alerts else 0)

anomaly_scores.append(det.predict(event))

    combined_preds = [
        1 if (rules_preds[i] or
anomaly_scores[i] > det.threshold) else 0
        for i in range(len(events))
    ]

    rows = []
    for name, preds, scores in [
        ("Rules Only", rules_preds, None),
        ("Rules + LogAI", combined_preds,
anomaly_scores),
    ]:
        m = evaluate_detector(labels, preds,
scores)
        m["Approach"] = name
        rows.append(m)

    df =
pd.DataFrame(rows).set_index("Approach"
)
    cols = ["TP", "FP", "TN", "FN",
"Precision", "Recall", "F1", "FPR",
"Accuracy"]
    if "AUC_ROC" in df.columns:
        cols.append("AUC_ROC")
    return df[[c for c in cols if c in
df.columns]]
# CLI

if __name__ == "__main__":
    from evaluation.generate_dataset import
generate_full_dataset

    print("Generating dataset…")
    dataset = generate_full_dataset(seed=42)
    n_norm = sum(1 for _, l in dataset if l ==
0)
    n_atk = sum(1 for _, l in dataset if l == 1)
    print(f"  {len(dataset)} samples  |
normal={n_norm}  attack={n_atk}\n")

    print("Running comparison…")
    df = compare_approaches(dataset)
    print(df.to_string())
    print()

    from parser import LogParser
    from rules_engine import RulesEngine
    p = LogParser()
    e = RulesEngine()
    labels = [lb for _, lb in dataset]
    preds = [1 if e.evaluate(p.parse(l)) else 0
for l, _ in dataset]
    m = evaluate_detector(labels, preds)
    print("Confusion matrix (Rules Only):")
    print(confusion_matrix_str(m))
```