

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

«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: Варіант програмного застосунку для автоматизованого збору та
аналізу локальних ІоС для сімейства систем Windows

Виконав(ла) здобувачка вищої освіти ІV курсу, групи ФБ-14

(шифр групи)

_____ Татаренко Анастасія Олександрівна

(прізвище, ім'я, по батькові)

_____ (підпис)

Керівник старший викладач каф. ІБ Козленко Олег Віталійович

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

_____ (підпис)

Рецензент доцент к.ф.-м.н. Терещенко Іван Миколайович

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

_____ (підпис)

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Здобувач ступеня бакалавра _____

(підпис)

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ 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

Татаренко Анастасії Олександрівні
(прізвище, ім'я, по батькові)

1. Тема роботи Варіант програмного застосунку для автоматизованого збору та аналізу локальних ІоС для сімейства систем Windows

керівник роботи старший викладач каф. ІБ Козленко Олег Віталійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

2. Термін подання здобувачем дипломної роботи «16» червня 2025 р.

3. Вихідні дані до роботи: технічна документація, Інтернет-джерела про графові бази даних, наукові та практичні джерела з аналізу ІоС та кореляції подій

4. Зміст роботи: дослідження природи індикаторів компрометації (ІоС) та їх ролі в системах виявлення загроз, аналіз обмежень традиційних засобів у роботі з ізольованими ІоС, розробка архітектури рішення для збору, нормалізації та графового аналізу подій Sysmon, реалізація програмного застосунку на основі DAG-моделі у Neo4j, моделювання сценаріїв хибнопозитивів і реальної компрометації, порівняння результатів з існуючими підходами та обґрунтування ефективності графового підходу

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

6. Дата видачі завдання: 16.12.2024

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Постановка задачі, формулювання теми дипломної роботи	04.11.2024 – 16.12.2024	Виконано
2	Теоретичне дослідження, аналіз джерел та існуючих рішень	16.12.2024 – 13.01.2025	Виконано
3	Розробка архітектури рішення	13.01.2025 – 27.01.2024	Виконано
4	Визначення компонентів, побудова моделі графа	27.01.2025 – 10.02.2025	Виконано
5	Опис та оформлення першого розділу дипломної роботи	10.02.2025 – 10.03.2025	Виконано
6	Опис та оформлення другого розділу дипломної роботи	10.03.2025 – 14.04.2025	Виконано
7	Реалізація модулів: збір подій, нормалізація, побудова DAG	14.04.2025 – 18.05.2025	Виконано
8	Тестування та порівняння з альтернативними інструментами	18.05.2025 – 26.05.2025	Виконано
9	Опис та оформлення третього розділу дипломної роботи	26.05.2025 – 02.06.2025	Виконано
10	Визначення напрямів подальших досліджень	02.06.2025 – 09.06.2025	Виконано
11	Підготовка до передзахисту та передзахист дипломної роботи	09.06.2025 – 16.06.2025	Виконано
12	Підготовка до захисту та захист дипломної роботи	16.06.2025 – 23.06.2025	

Здобувач вищої освіти _____
(підпис)

Анастасія ТАТАРЕНКО
(власне ім'я, ПРІЗВИЩЕ)

Науковий керівник _____
(підпис)

Олег КОЗЛЕНКО
(власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Обсяг роботи 81 сторінка, 8 ілюстрацій, 5 таблиць, 4 додатки, 22 джерела літератури.

Об'єкт дослідження: Процеси виявлення компрометації інформаційних систем на основі індикаторів (IoC) та подій у середовищі Windows.

Предмет дослідження: Методи збору, кореляції та аналізу локальних індикаторів компрометації (IoC) у Windows, з використанням графових структур і логічних зв'язків подій.

Мета дослідження: Розробка програмного застосунку, що дозволяє збирати локальні IoC у Windows-середовищі, корелювати їх з відповідними подіями в системі та виявляти ланцюги дій, які свідчать про реальну компрометацію.

Методи дослідження: аналіз джерел з теми IoC та кореляції подій, експериментальне моделювання інцидентів безпеки у середовищі Windows, збір та нормалізація даних у форматі JSON з застосуванням бібліотек мови Python, побудова графової моделі подій у Neo4j, тестування запропонованого підходу, порівняльний аналіз з результатами традиційної системи

Отримані результати: Розроблено програмний застосунок для збору та аналізу локальних IoC у Windows, реалізовано нормалізацію подій Sysmon і побудову графової моделі у Neo4j. Здійснено тестування та порівняння з існуючими підходами. Підтверджено, що графовий підхід дозволяє ефективно виявляти ланцюги подій і знижує кількість хибних спрацювань у порівнянні з класичними системами.

Результати роботи були представлені на XXIII Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених.

Ключові слова: індикатори компрометації (IoC), системні події, Sysmon, графова модель, Neo4j, кореляція, хибнопозитив

ABSTRACT

The volume of the work is 81 pages, 8 illustrations, 5 tables, 4 appendices, 22 literature references.

Object of research: Processes for detecting compromise of information systems based on indicators (IoC) and events in the Windows environment.

Subject of research: Methods of collecting, correlating and analysing local indicators of compromise (IoC) in Windows, using graph structures and logical connections of events.

Purpose: To develop a software application that allows collecting local IoCs in a Windows environment, correlating them with relevant events in the system, and identifying chains of actions that indicate a real compromise.

Research methods: analysis of sources on the topic of IoC and event correlation, experimental modelling of security incidents in the Windows environment, collection and normalisation of data in JSON format using Python libraries, building a graph model of events in Neo4j, testing of the proposed approach, comparative analysis with the results of the traditional system

Results: A software application for collecting and analysing local IoCs in Windows has been developed, Sysmon event normalisation and graph model building in Neo4j have been implemented. Testing and comparison with existing approaches were carried out. It was confirmed that the graph approach allows to effectively detect chains of events and reduces the number of false positives compared to classical systems.

The results of the work were presented at the XXIII All-Ukrainian Scientific and Practical Conference of Students, Postgraduates and Young Scientists.

Keywords: indicators of compromise (IoC), system events, Sysmon, graph model, Neo4j, correlation, false positive

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	8
ВСТУП	10
1 ТЕОРЕТИЧНІ ОСНОВИ ЗБОРУ ТА АНАЛІЗУ ІНДИКАТОРІВ КОМПРОМЕНТАЦІЇ	13
1.1 Поняття індикаторів компрометації (IoC): класифікація та призначення	13
1.2 Архітектура систем виявлення загроз (IDS, EDR, SIEM): роль IoC у виявленні інцидентів.....	14
1.3 Проблематика ізольованих IoC: ризик хибнопозитивних спрацювань..	18
1.4 Логування подій у Windows: можливості Sysmon	21
1.5 Існуючі підходи до кореляції IoC з подіями системи	31
Висновки до розділу 1	37
2 ЗАПРОПОНОВАНИЙ ПІДХІД ДО КОРЕЛЯЦІЇ IoC З ПОДІЯМИ	39
2.1 Дані та ключові об'єкти	39
2.2 Графова модель та типи ребер.....	40
2.3 Алгоритм нарощування графа з журналів.....	42
2.4 Правила формування ланцюга компрометації.....	44
2.5 Критерії підтвердженої компрометації.....	46
2.6 Архітектурна схема компонентів	47
Висновки до розділу 2	48
3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ ТА РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ	50
3.1 Архітектура реалізованого рішення.....	50
3.2 Побудова та візуалізація графа в Neo4j	55
3.3 Тестування методики на контрольних сценаріях	58
3.4 Аналіз ефективності методу	60
3.5 Пропозиції щодо розширення функціональності	63
Висновки до розділу 3	65
ВИСНОВКИ.....	67
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	70
ДОДАТОК А.....	73

ДОДАТОК Б	76
ДОДАТОК В.....	78
ДОДАТОК Г	79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoC (Indicator of Compromise) – індикатор компрометації; цифровий артефакт (наприклад, IP-адреса, хеш файлу, домен), який може свідчити про порушення інформаційної безпеки.

EDR (Endpoint Detection and Response) – система виявлення та реагування на загрози на кінцевих пристроях, що здійснює моніторинг та аналіз поведінки процесів.

SIEM (Security Information and Event Management) – система управління інформаційною безпекою, що збирає, корелює та аналізує події з різних джерел.

SOC (Security Operations Center) – центр операцій інформаційної безпеки, який забезпечує моніторинг, аналіз та реагування на інциденти безпеки.

Sysmon (System Monitor) – утиліта Microsoft для детального журналювання подій у Windows, зокрема запуску процесів, створення файлів та мережевих підключень.

JSON (JavaScript Object Notation) – легкий текстовий формат для обміну структурованими даними, що використовується в логах та API.

DAG (Directed Acyclic Graph) – напрямлений ациклічний граф, що не містить циклів і використовується для відображення послідовних залежностей між подіями.

Hash (Хеш) – криптографічна функція, що однозначно ідентифікує файл або об'єкт даних (наприклад, SHA-256, MD5).

CommandLine – командний рядок, з якого був ініційований запуск процесу, використовується для аналізу поведінки.

JA3 / JA4 – TLS-фінгерпринти, що дозволяють ідентифікувати клієнтів або сервери за параметрами SSL-сеансу.

YARA – мова опису сигнатур для виявлення шкідливого ПЗ за поведінковими або бінарними ознаками.

CSV (Comma-Separated Values) – табличний формат збереження даних, де значення розділені комами або іншими символами.

IP-адреса – унікальний ідентифікатор пристрою в мережі, що може використовуватись як IoC.

URL (Uniform Resource Locator) – уніфікований вказівник ресурсу в Інтернеті; може бути частиною IoC.

Neo4j – графова база даних, що дозволяє зберігати й обробляти об'єкти та зв'язки між ними у вигляді вузлів і ребер.

PowerShell – потужна командна оболонка для автоматизації та адміністрування в Windows, яка часто використовується в атаках.

Triage – первинне сортування сповіщень безпеки з метою визначення їх пріоритету для подальшої обробки.

Alert Fatigue – зниження уваги аналітиків до алертів унаслідок великої кількості хибнопозитивних або неважливих спрацювань.

Lookup – перевірка артефакта (наприклад, IP чи хешу) у відомому списку загроз.

Payload – шкідливе навантаження або код, що виконується після проникнення в систему.

ВСТУП

У сучасних умовах постійного зростання кількості кіберзагроз, зокрема спрямованих на користувачів операційної системи Windows, питання своєчасного виявлення ознак компрометації (IoC – Indicators of Compromise) набуває особливої важливості.

Більшість інструментів виявлення загроз, таких як антивірусне ПЗ, системи виявлення вторгнень (IDS) або рішення класу EDR, працюють за принципом пошуку конкретних, заздалегідь відомих ознак атак. Проте виявлення окремого IoC без розуміння його контексту нерідко призводить до хибнопозитивних спрацювань або, навпаки, пропущених інцидентів.

Актуальність теми: Не всі індикатори компрометації (IoC) свідчать про реальну загрозу інформаційній безпеці. Деякі з них, такі як хеші відомих шкідливих файлів або підозрілі командні рядки, можуть бути присутні в системі з неінцидентних причин—у ході тестування, резервного копіювання чи некоректної класифікації. Таким чином, лише комплексний аналіз, що враховує повний контекст появи IoC (процесне дерево, мережеві підключення та дії користувача), дозволяє здійснити коректну оцінку ризику.

У зв'язку з цим актуальним є підхід, при якому IoC аналізуються не ізольовано, а у контексті подій, що відбулися в системі, з метою виявлення ланцюгів дій, які можна інтерпретувати як реальну компрометацію. Цей підхід поєднує методи збору системних подій, нормалізацію та графовий аналіз для виявлення логічних залежностей.

Мета і завдання дослідження: Метою даної роботи є розробка програмного застосунку, що дозволяє збирати локальні IoC у Windows-середовищі, корелювати їх з відповідними подіями в системі та виявляти ланцюги дій, які свідчать про реальну компрометацію.

Для досягнення цієї мети були поставлені такі основні завдання:

1. дослідити джерела збору локальних подій та IoC в ОС Windows;
2. реалізувати механізм збору та збереження подій та IoC (зокрема, з використанням утиліти Sysmon);
3. розробити модуль нормалізації даних до уніфікованого формату;
4. реалізувати механізм побудови логічних зв'язків між подіями;
5. побудувати граф для представлення IoC та подій у базі даних Neo4j;
6. визначити критерії, за якими можливо зробити висновок про наявність компрометації;
7. протестувати розробку на практичних прикладах.

Методи дослідження. У процесі виконання роботи використовувались такі методи:

1. метод аналізу логів безпеки та системних подій (зокрема, Windows Event Log і Sysmon);
2. метод нормалізації структурованих даних для подальшого аналізу;
3. метод побудови графів на основі логічних зв'язків між подіями;
4. метод кореляції IoC з діями, виявленими у системі;
5. експериментальні методи – побудова контрольних тест-кейсів з симульованими атаками;
6. візуальний аналіз графа подій у базі даних Neo4j.

Структура роботи: Дипломна робота складається з трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі розглянуто теоретичні основи збору та інтерпретації IoC, а також сучасні підходи до виявлення інцидентів безпеки на основі логів.

Другий розділ присвячено опису запропонованого методу побудови графа подій та індикаторів компрометації, а також формалізації логічних критеріїв для визначення компрометації.

У третьому розділі описано реалізацію програмного застосунку, експерименти з його використанням, результати тестування на прикладах, що імітують хибнопозитивні та реальні інциденти, а також порівняння з існуючим рішенням.

1 ТЕОРЕТИЧНІ ОСНОВИ ЗБОРУ ТА АНАЛІЗУ ІНДИКАТОРІВ КОМПРОМЕНТАЦІЇ

1.1 Поняття індикаторів компрометації (ІоС): класифікація та призначення

Індикатор компрометації (ІОС) - це частина цифрової криміналістики, яка вказує на те, що кінцеву точку або мережу, можливо, було зламано. Так само, як і речові докази, ці цифрові підказки допомагають фахівцям з інформаційної безпеки виявити зловмисну діяльність або загрози безпеці, такі як витік даних, інсайдерські загрози або атаки шкідливого програмного забезпечення.

Слідчі можуть збирати індикатори компрометації вручну, помітивши підозрілу активність, або автоматично. Ця інформація може бути використана для пом'якшення поточної атаки або усунення існуючого інциденту безпеки, а також для створення «розумніших» інструментів, які можуть виявляти і відправляти на карантин підозрілі файли в майбутньому.

Моніторинг ІОС має здебільшого реактивний характер, оскільки їх виявлення часто свідчить про те, що система вже зазнала компрометації. Водночас, оперативне ідентифікування ІоС у процесі атаки дозволяє нейтралізувати загрозу на ранніх етапах її розвитку, мінімізуючи потенційний вплив на інформаційну безпеку організації. [\[1\]](#)

Класифікація ІоС. Залежно від характеру артефактів, ІоС можна класифікувати наступним чином [\[2\]](#):

1. Файлові індикатори: імена файлів, абсолютні шляхи до файлів, контрольні суми (хеші) — MD5, SHA1, SHA256, ім'я оригінального файлу.

2. Мережеві індикатори: IP-адреси, доменні імена, URL-адреси, порти. Наприклад: домен, до якого встановлюється з'єднання при командно-контрольній взаємодії (C2 communication).
3. Поведінкові індикатори: командні рядки запуску, шляхи до виконуваних файлів, спадковість процесів (ProcessGuid → ParentProcessGuid), взаємодія з API, аномальна поведінка в реєстрі.
4. Часові та контекстні індикатори: час активації шкідливого коду, обліковий запис користувача, підозріла сесія входу в систему.

Роль у Threat Hunting. На відміну від реактивного аналізу інцидентів, Threat Hunting передбачає проактивний пошук шкідливої активності. У цьому процесі IoC виступають як вихідна точка або «гіпотеза», яку дослідник перевіряє шляхом аналізу подій, зібраних із системи [3]. Наприклад, виявлення запуску певного скрипта може спонукати перевірку, які процеси породили цей скрипт, які з'єднання були встановлені тощо.

1.2 Архітектура систем виявлення загроз (IDS, EDR, SIEM): роль IoC у виявленні інцидентів

Архітектура конкретного засобу захисту визначає, де саме, та яким чином, можна «вмонтувати» індикатори компрометації (IoC) у процес виявлення інцидентів. Спрощено в інформаційній системі можна виділити три рівні спостереження: мережевий, хостовий та організаційний (або кореляційний) — кожен із власним обсягом даних, часом спрацювання й типами IoC, які найбільш ефективно використовувати в кожному конкретному рівні [4] (див. табл. 1.1).

Таблиця 1.1 – Ефективність типів IoC на рівнях IDS, EDR та SIEM

Рівень	Приклад технології	Типові джерела/IoC	Особливість, що впливає на використання IoC
Мережевий	IDS/IPS (Snort, Suricata, Cisco NGIPS)	IP-, DNS-, URL-артефакти, TLS-сертифікати, сигнатури пакета	Працює тільки з трафіком → бачить зовнішні IoC, але не знає контексту процесів.
Хостовий	EDR/NGAV (Microsoft Defender for Endpoint, ESET Inspect)	хеші файлів, шляхи EXE/DLL, поведінкові патерни (CreateRemoteThread тощо)	Має телеметрію ядра/пам'яті → може перевіряти глибокі IoC, але лише на конкретному хості. (BlueVoyant)
Кореляційний / організаційний [5]	SIEM / XDR (Splunk, Elastic Security)	агреговані IoC з TI-фідів (Threat Intelligence feeds) + дані нижчих рівнів (процеси, реєстр, мережеві логи)	Консолідує журнали з усіх джерел → дозволяє писати кореляційні правила та багатокрокові зв'язки між IoC і подіями.

Архітектура систем виявлення загроз безпосередньо впливає на ефективність використання індикаторів компрометації (IoC), оскільки визначає обсяг телеметрії, місце інтеграції правил виявлення та швидкість реакції.

- Обсяг і деталізація телеметрії. IDS працює лише з мережею, тому фіксує ІоС у вигляді ІР-адрес чи доменів. Натомість EDR має доступ до системних викликів, що дає змогу аналізувати поведінкові ознаки загрози.
- Механізм застосування правил. У мережевому шарі ІоС перетворюються на сигнатури або YARA-правила; на хості – у хеш-лок-апи й правила поведінки; у SIEM – у багатокрокові кореляційні правила, що поєднують події різних класів.
- Швидкість реагування. IDS/IPS миттєво блокує трафік, EDR—процеси, а SIEM потребує додаткового часу на отримання та обробку логів перед винесенням остаточного вердикту.
- Контекст. Чим ближче до кінцевої точки (EDR), тим більше інформації для підтвердження/спростування спрацьовування ІоС. SIEM компенсує відсутність низькорівневого контексту широтою огляду та історичною пам'яттю.

Індикатори компрометації (ІоС) проходять три основні етапи інтеграції в архітектуру системи безпеки. Спочатку вони обробляються на рівні мережевого фільтра, де перевіряються сигнатурні ознаки загроз у трафіку. Далі сенсори на хості аналізують ІоС у контексті локальних процесів, файлів та реєстру. Завершальним етапом є централізована кореляція в SIEM-системі, яка об'єднує дані з різних джерел для формування повної картини інциденту.

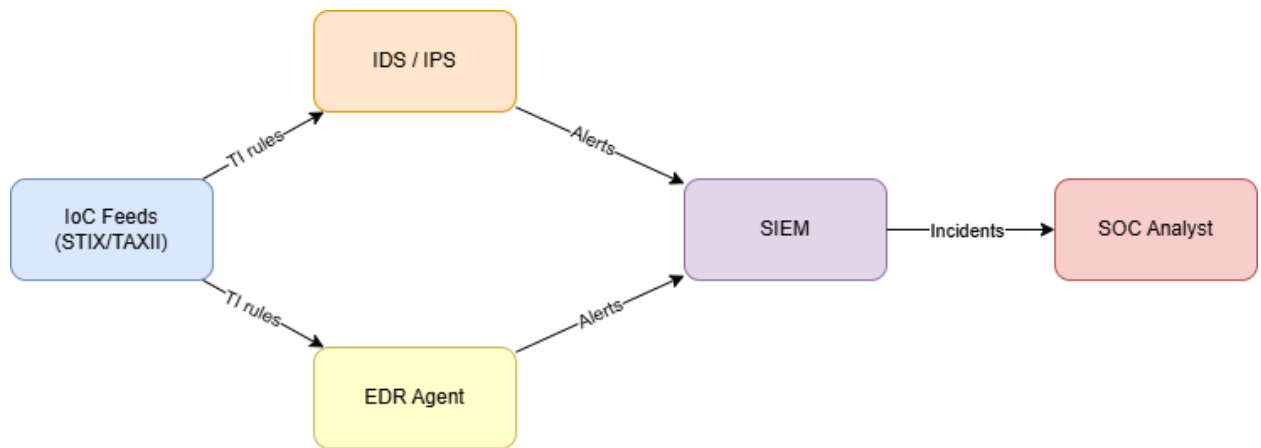


Рисунок 1.1 – Вбудовування IoC у тришарову архітектуру IDS → EDR → SIEM

Розглянемо ефективність різних типів індикаторів компрометації (IoC) на кожному рівні архітектури безпеки та визначимо механізми їх кореляції. Це дозволить перейти від окремих спрацьовувань до узагальненого аналізу, що сприятиме більш точному виявленню атак.

Мережевий рівень (IDS / IPS)

Індикатори компрометації, отримані з зовнішніх фідів, перетворюються на правила сигнатурного аналізу мережевого трафіку. Після оперативного оновлення конфігурації (hot-reload) датчик уже в найближчому сеансі контролює пакети на предмет збігу з новим атрибутом. У разі спрацювання формується негайне повідомлення до SOC і, за наявності режиму IPS, ініціюється блокування сесії. Найдоцільніше тут працюють “зовнішні” IoC: статичні IP-адреси, CIDR-діапазони, FQDN-домени, характерні URL-шаблони та JA3/JA4-фінгерпринти TLS-сеансів. Усі вони описують мережеві атрибути, які можна оцінити ще до того, як пакет потрапить на хост. Перевага – мінімальна затримка детекції й можливість негайного блокування; недолік – відсутність інформації, який саме процес або користувач ініціював трафік..

Хостовий рівень (EDR / Endpoint Sensor)

Ті самі індикатори (насамперед хеші файлів, фрагменти командного рядка, підписи сертифікатів) завантажуються до локальної бази агента на

кінцевій точці. Якщо на диску з'являється файл із відповідним хешем або запускається процес із підозрілою командою, агент фіксує подію та, згідно з політикою, блокує виконання, переносить об'єкт у карантин чи ізолює хост від мережі. Найефективніші тут “внутрішні” ІоС: хеші виконуваних і скриптових файлів, рядки командної строки, шляхи автозапуску, імена mutex-ів, параметри PowerShell. Такі атрибути неможливо відфільтрувати на мережі, зате вони точно вказують на роботу шкідливого коду усередині ОС. Перевага – багатий контекст (GUID процесу, обліковий запис, шлях файлу) та миттєве локальне стримування; обмеження – відсутність огляду на зовнішній трафік інших машин.

Кореляційний рівень (SIEM / XDR)

Журнали IDS та EDR централізуються у SIEM, де виконуються кореляційні правила «першого порядку». Платформа перевіряє, чи один і той самий індикатор зустрівся принаймні у двох незалежних джерелах, наприклад: мережева подія з IDS і локальна подія з Sysmon або EDR.

Якщо збіг підтверджується в межах встановленого часово-кореляційного вікна, формується інцидент, у якому об'єднано мережеву складову (адреса, порт, протокол) та хостовий контекст (процес, користувач, хеш файлу). Такий підхід підвищує достовірність спрацювання, але потребує уніфікованої схеми полів і здатний породжувати затримку, пов'язану з надходженням та індексацією логів.

1.3 Проблематика ізолюваних ІоС: ризик хибнопозитивних спрацювань

Ізолюваний індикатор компрометації (ІоС) – це артефакт загрози, який аналізується без урахування контексту його появи, зокрема часу, місця та

пов'язаних дій у системі. До таких IoC належать окремі IP-адреси, хеші файлів або JA3-фінгерпринти, які потрапляють до списків загроз внаслідок одноразової активності потенційного зловмисника. Відсутність контекстної інформації перетворює ізольований IoC на статичний параметр, що не дозволяє коректно оцінити реальну тактику компрометації або поточний стан загрози в інформаційній системі. [6]

Ізольовані індикатори компрометації (IoC) часто спричиняють хибнопозитивні спрацювання через низку факторів.

По перше, динамічність інфраструктури. Постачальники CDN, хмарні сервіси та VPN-мережі перевикористовують ті самі IP-адреси для легітимних клієнтів. Як наслідок, сигнатурний підхід до визначення IoC на основі IP-адреси може швидко втратити актуальність, що призводить до маркування легітимного трафіку як потенційно шкідливого.

По друге, відсутність контексту. Один і той самий SHA-256 може належати як шкідливому пакувальнику, так і легальному адміністративному інструменту – залежно від його розташування та способу використання. Без урахування додаткових параметрів, таких як шлях файлу або командний рядок, хеш-IoC може генерувати зайві спрацювання..

По третє, недостатня якість фідів. Відсутність полів confidence / severity або автоматичне агрегування з ненадійних джерел призводить до включення застарілих чи некоректних записів у систему виявлення загроз. [7]

Останнє: короткий життєвий цикл атак. Дослідження Dragos [8] показують, що частина IoC залишається актуальною лише кілька годин або днів, тоді як дані у фідах можуть зберігатися місяцями, продовжуючи генерувати сповіщення навіть після зникнення реальної загрози.

Вартість хибнопозитивів. Кожне спрацювання без реального підтвердження – це: витрата часу аналітика на з'ясування обставин;

збільшення бюджету «alert quota» у SIEM; втрата довіри до системи виявлення («alert fatigue»).

У дослідженнях Enterprise Strategy Group (ESG) та CSO Online понад 45% усіх алертів від веб- та API-захисних інструментів виявляються хибними [\[9\]](#), [\[10\]](#)

Велика кількість ізольованих індикаторів компрометації (IoC) у політиці безпеки збільшує ризик пропуску реальних атак через надмірну кількість хибнопозитивних спрацювань. Відповідно до оновлених метаданих Emerging Threats, правила з високим рівнем довіри (High confidence) – що характеризують низьку ймовірність хибнопозитивів – майже не породжують некоректних сповіщень. Натомість правила з низьким рівнем довіри (Low confidence) допускають значну кількість хибних спрацювань, що може вплинути на ефективність аналітичного процесу. [\[11\]](#)

Основними методами мінімізації ризику хибнопозитивних спрацювань IoC є:

1. Контекстуальне збагачення. Включення до індикаторів компрометації додаткових атрибутів, таких як часові характеристики (TTL), джерело походження, рівень довіри та пов'язані техніки MITRE ATT&CK, що дозволяє більш точно визначати їх релевантність.
2. Кореляція різнорівневих подій. Аналіз IoC у поєднанні з подіями на рівні хосту, зокрема перевірка, чи збіг IP-адреси супроводжується створенням підозрілого процесу. Використання мережесих та системних сигналів у комплексі значно зменшує ймовірність хибних спрацювань.
3. Автоматизований перегляд фідів. Відфільтровування записів, що не оновлювалися протягом встановленого періоду, та переведення менш релевантних IoC у категорію низького пріоритету.

4. Оцінка рівня довіри (confidence scoring). Перед імпортуванням IoC у блокувальні політики їм присвоюється ваговий коефіцієнт, що дозволяє встановлювати суворіші порогові значення лише для високодостовірних індикаторів.

Замість ізольованого аналізу IoC доцільно застосовувати графову модель кореляції, яка враховує послідовність подій та поведінкові патерни, дозволяючи формувати пов'язані взаємозв'язки між індикаторами компрометації та діями в системі. У наступному розділі детально розглянуто цю модель та її практичне застосування.

1.4 Логування подій у Windows: можливості Sysmon

У типовій інфраструктурі Windows головним сховищем інформації про те, що відбувається в операційній системі, є **Windows Event Log**. Його стандартні канали – Security, System та Application – уже багато років залишаються основним джерелом даних для адміністратора чи засобу моніторингу. Однак архітектура цих каналів формувалася ще під потреби аудиту Active Directory, а не під сучасну цифрову криміналістику. Тому записів, які вони створюють, вистачає лише для високорівневого «звіту про факт», але бракує деталей, критичних для реконструкції поведінки шкідливого коду або «білого» підозрілого скрипта.

По-перше, у подіях немає глобально унікального ідентифікатора процесу. Стандартний Event Log дає лише PID – число, актуальне поки процес живе. Щойно процес завершується, PID може бути перевикористаний, і спроба відстежити ланцюжок «батько ↔ нащадок» стає ненадійною. Для цифрової криміналістики потрібен постійний GUID, який зберігається незалежно від перезапусків.

По-друге, журнал не фіксує повний командний рядок. Часом саме один-два ключі PowerShell або wscript відділяють легітимний запуск від експлуатації вразливості; без цих ключів аналітик бачить лише ім'я файлу-інтерпретатора і змушений здогадуватися, що всередині виконувалося.

По-третє, у стандартних подіях відсутня детальна мережева та файлово-системна телеметрія: локальні й віддалені порти, IP-адреси, імена DNS-запитів, повні шляхи створених або змінених файлів. Такі атрибути (разом із десятками інших) існують всередині ядра Windows як Event Tracing for Windows (ETW) – високошвидкісного механізму трасування, який збирає інформацію без відчутних накладних витрат. Але за замовчуванням ETW події не потрапляють у класичний Event Log: їх треба або явно увімкнути, або задіяти компонент, що вміє «підняти» ці дані на поверхню.

Саме таким компонентом між ETW і оператором SOC став **Sysmon (System Monitor)** із пакета Sysinternals. Працюючи комбінацією драйвера ядра та фонові служби, Sysmon перехоплює критичні ETW-сигнали, додає до кожного нового процесу статичні хеші, GUID і атрибути безпеки, а потім записує все в окремий канал Microsoft-Windows-Sysmon/Operational – той самий Windows Event Log, але вже з телеметрією, якої бракувало. За офіційною документацією Microsoft Learn, це дозволяє «розширити видимість системи до рівня, необхідного для виявлення та розслідування сучасних загроз», не встановлюючи повноцінний комерційний EDR-агент. [\[12\]](#)

Таким чином, Sysmon закриває розрив між мінімалістичним, орієнтованим на аудит журналом Windows і потребами кіберзахисту, впроваджуючи у звичний механізм логування той обсяг контексту, який потрібен для глибокої експертизи та кореляції з індикаторами компрометації.

Після встановлення Sysmon розгортається як двокomпонентна служба «ядро + користувацький простір», доповнена факультативними блокувальними модулями, що з'явилися у версіях ≥ 15 .

На найнижчому рівні працює kernel-mode драйвер SysmonDrv.sys. Його цифровий підпис від Microsoft дозволяє безпечно інтегруватися у вразливі ядрові механізми Process Notify, Network Callout та Registry Callback. Драйвер «підслуховує» системні події раніше, ніж вони досягають користувацьких API, тому бачить створення процесів, відкриття сокетів чи модифікацію реєстру у момент фактичного виклику ядрової функції. Це мінімізує шанс, що шкідливий код устигне приховати сліди. Усі перехоплені атрибути (PID, образ, шлях, мережеві параметри) драйвер маркує унікальним Process GUID і передає в режим користувача.

У user-mode працює служба Sysmon.exe. Вона читає XML-конфігурацію, добудовує до сирих подій додаткові метадані – хеші SHA-1/256, сесійну GUID, статус цифрового підпису – і публікує результат через Event Tracing for Windows (ETW) у власний журнал Microsoft-Windows-Sysmon/Operational. Завдяки ETW продуктивність майже не страждає: трасування передається через буфери спільної пам'яті без контекстних перемикачів ядра.

Починаючи з версії 15 розробники додали блокувальні режими – FileBlockExecutable та FileBlockShredding. У них Sysmon уже не лише «спостерігає», а й активно перешкоджає зловмисникам: подія 27 блокує створення нового PE-файлу, а подія 28 зупиняє спроби інструментів на кшталт SDelete безповоротно перезаписати дані. Якщо увімкнено FileExecutableDetected (подія 29), служба логуватиме сам факт появи EXE, навіть коли блокування не застосовується. [\[12\]](#)

Уся конструкція запускається як типовий Windows service і стартує разом із системою. Поки комп'ютер увімкнений, kernel driver → user-mode service → ETW → журнал формують безперервний конвеєр телеметрії; після перезавантаження процес GUID пара циклічно починається спочатку, але зв'язок «GUID ↔ PID» відновлюється автоматично, що важливо для довготривалого моніторингу.

Таким чином, архітектура Sysmon забезпечує глибоке та малозатратне захоплення подій від ядра до журналу, а додаткові блокувальні модулі дають можливість реагувати на загрози ще до того, як вони закріпляться у системі.

Таксономія подій Sysmon. Після встановлення Sysmon кожна системна активність перетворюється на структурований запис із числовим Event ID. Усі події мають десятки підтипів, однак, на мою думку, варто зосередитись на тих, що найчастіше фігурують у кореляційних правилах IDS / EDR / SIEM.

Event ID 1 – Process Create. Це фундаментальний сигнал про запуск нового процесу. Він містить ProcessGUID (унікальний у межах домену), повний CommandLine, SHA-1/256 Hashes та посилання на ParentProcessGuid/Image. Наявність GUID дає змогу відстежувати ланцюжок батьківських і дочірніх процесів навіть після повторного використання PID.

Event ID 3 – Network Connection. Логує кожне TCP/UDP-з'єднання та прив'язує його до конкретного процесу через ProcessId/ProcessGuid. Поля Source/Destination IP і Port дозволяють безпосередньо порівнювати трафік хоста з мережевими ІоС. Подія вмикається за замовчуванням, тому її вмикають лише для вузлів під пильним моніторингом, аби не роздувати об'єм журналу.

Event ID 7 – Image Loaded. Фіксує момент підвантаження DLL чи іншого модуля у вже запущений процес. Дає хеш і статус цифрового підпису бібліотеки, допомагаючи ловити нелегітимні DLL-інжекції. Подія «дорога» в сенсі кількості записів, тому її зазвичай вмикають точково (наприклад, лише для браузерів або служб, що мають доступ до конфіденційних даних).

Event ID 11 – File Create. Реєструє створення чи перезапис файлу. Улюблений тригер для правил, що відстежують автозапуск (Startup, Run-ключі) або директорії %TEMP%, Downloads. Поле Hashes доступне, якщо у конфігурації задано hashAlgorithms="SHA256"; саме за цими хешами SIEM зіставляє запис із файловими ІоС.

Event ID 22 – DNS Query. Пише кожен DNS-запит разом із результатом (успішним чи ні) і процесом-ініціатором. Це один із небагатьох стандартних способів побачити, яке саме EXE звертався до шкідливого домену, – інформація, якої бракує мережевій IDS, що бачить лише пакет.

Event ID 25 – Process Tampering. Виявляє техніки приховування або заміни образу процесу: Process Hollowing, Herpaderping та споріднені. Подія спрямована на зупинку пост-експлуатаційних дій, коли шкідливий код намагається маскуватися під легітимний exe.

Event ID 27 – 29 – FileBlock / ExecutableDetected. Починаючи з версії 15 Sysmon може не лише логувати, а й блокувати:

- 27 (FileBlockExecutable) – забороняє створення нового PE-файлу;
- 28 (FileBlockShredding) – зупиняє спроби безповоротно перезаписати файл (наприклад, інструментом SDelete);
- 29 (FileExecutableDetected) – просто повідомляє про появу виконуваного файла без блокування.

Ці події особливо корисні для систем, де встановлення повноцінного EDR неможливе, але потрібен мінімальний рівень протидії ransomware.

[\[12\]](#)

Повний журнал Sysmon містить додаткові десятки ID (операції реєстру, WMI, Clipboard, Named Pipes тощо), але саме описані вище події найчастіше використовуються у практиці SOC і формують правила кореляції.

Оптимальне налаштування зазначених ключових ідентифікаторів (ID) забезпечує баланс між деталізацією телеметрії та керованим обсягом логів. Це є критичним фактором для подальшого імпорту даних у SIEM-систему та коректного формування графової моделі подій.

XML-конфігурація та система фільтрів Sysmon. Після інсталяції Sysmon працює за правилами, описаними у файлі sysmonconfig.xml. Це XML-

документ з чіткою ієрархією, що задає: які саме події логувати, у якому вигляді й для яких об'єктів (див. рис. 1.2).

```
▼<RuleGroup name="PowerShell" groupRelation="or">
  ▼<ProcessCreate onmatch="include">
    <Image condition="contains">\powershell.exe</Image>
    <Signature condition="is">Unsigned</Signature>
  </ProcessCreate>
</RuleGroup>
```

Рисунок 1.2 - Фрагмент XML-конфігурації Sysmon

Структурний рівень 1 – RuleGroup. Конфігурація починається з одного або кількох тегів <RuleGroup>. Кожна група визначає тему («DNS-активність», «PowerShell», «USB-носії» тощо) і прапор groupRelation="or|and", який пояснює, як комбінувати її дочірні правила. Атрибут onmatch="include|exclude" задає реакцію за промовчанням: логувати або ігнорувати все, що відповідає умовам усередині групи.

Структурний рівень 2 – EventFiltering. У межах <RuleGroup> ідуть елементи на кшталт <ProcessCreate>, <NetworkConnect>, <FileCreateTime> – один до одного з номерами Event ID. Фільтр вмикається лише тоді, коли подія відповідає хоча б одній умові. Якщо <ProcessCreate> узагалі відсутній, Sysmon продовжить логувати всі запуски процесів (поведінка "log-all"). Тож XML фактично служить «білим або чорним списком» для кожного каналу телеметрії.

Структурний рівень 3 – Condition. Усередині тегів подій розміщуються поля-умови: <Image>, <CommandLine>, <ParentImage>, <Signature> та ін. Кожне поле має атрибут condition="contains|is|begin with|end with|image|excludes". Найуживанішими залишаються:

- contains – логувати, якщо підрядок зустрічається будь-де
- is – точна відповідність

- image – спеціальний різновид is, який додатково нормалізує шлях (реєстр нечутливості до регістру, розділювачів тощо)

У виробничому середовищі Sysmon забезпечує безперервне формування потоку подій Windows-хоста, додаючи до кожного процесу глобальний GUID, обчислені хеші, мережеві атрибути та відомості про цифровий підпис. Після реєстрації подій у журналі Microsoft-Windows-Sysmon/Operational вони надходять до SIEM або XDR-системи, де здійснюється їх кореляція з наявними індикаторами компрометації, що дозволяє виявляти потенційні загрози на основі зібраної телеметрії.

Одним із ключових механізмів контролю виконуваних файлів є перевірка їхніх хешів. Адміністратор або автоматизований конектор імпортує до SIEM список SHA-256, отриманий із зовнішнього джерела загроз (TI-фіду) або внутрішнього реєстру блокувань. У разі фіксації запуску процесу (подія 1) або створення файлу (подія 11) Sysmon додає хеш до журналу подій. Кореляційне правило indicator-match порівнює його з базою загроз, а у разі збігу миттєво підвищує рівень критичності сповіщення. Це забезпечує раннє виявлення потенційно шкідливого об'єкта ще до встановлення ним мережевого каналу або внесення змін до реєстру."до того, як той устиг установити мережевий канал чи змінити реєстр.

Друга категорія – мережеві IoC. Події 3 (Network Connection) та 22 (DNS Query) несуть IP-адресу або домен разом із GUID процесу-ініціатора. Якщо ця адреса вже позначена у фіді як Command-and-Control, SIEM може накласти мережевий збіг IDS на процесний контекст Sysmon і реконструювати повний ланцюг «зовнішнє з'єднання → конкретний exe → користувач». У звичайному IDS-логові такого зв'язку бракує, тож саме поєднання двох шарів робить інцидент однозначним і зрозумілим для аналітика.

Sysmon дозволяє комбінувати поведінкові патерни з атомарними індикаторами, що значно підвищує точність виявлення. Наприклад, у разі

відкриття користувачем Excel-файлу із вбудованим шкідливим макросом цей макрос може непомітно створити виконуваний файл loader.exe у каталозі %TEMP%. Ця активність миттєво фіксується Sysmon: у журналі реєструється подія ID 11 («File Create»), яка містить обчислений SHA-256 хеш нового файлу. Якщо SIEM має актуальний список шкідливих хешів, він здійснює перевірку та, у разі збігу, маркує запис як потенційно небезпечний. Це дозволяє виявити загрозу на ранньому етапі до її подальшої активності в системі.

Через кілька секунд виконуваний файл запускається, що фіксується Sysmon як подія ID 1 («Process Create») із відповідним глобальним Process GUID та повним командним рядком. У SIEM з'являється додатковий доказ компрометації: хеш, що раніше був ідентифікований у списку індикаторів загроз, тепер підтверджує активне виконання файлу.

Після запуску шкідливий код ініціює мережеве з'єднання з сервером керування. Sysmon реєструє цю активність як подію ID 3 («Network Connect») та знову прив'язує її до Process GUID, що дозволяє точно визначити джерело з'єднання. IP-адреса призначення потрапляє до списку C2-індикаторів, що забезпечує третє підтвердження загрози у SIEM.

Поодинокі кожен із цих сигналів – підозрілий хеш, запуск файлу чи контакт із сумнівною IP-адресою – міг би згенерувати хибнопозитив. Але коли всі три події стосуються одного й того самого процесу з тим самим GUID, кореляційно можна побачити цілісний ланцюг «створення файлу → запуск → мережевий вихід». Така картина практично не залишає сумнівів: загроза реальна, і пріоритет алерта підвищується автоматично. Отже, сила Sysmon полягає не лише у великій кількості телеметрії, а у здатності об'єднати атомарні індикатори з поведінковим контекстом і подати аналітику вже перевірений, значущий інцидент.

Типові обмеження й «підводні камені» Sysmon-моніторингу. Попри ефективність Sysmon у зборі детальної телеметрії, його застосування має низку обмежень, які необхідно враховувати ще на етапі проєктування системи журналювання. Коректне налаштування дозволяє мінімізувати ці ризики та забезпечити стабільну роботу механізмів аналізу подій безпеки.

По-перше, великий обсяг даних. Готові конфігураційні файли на зразок конфігурації SwiftOnSecurity пишуть усі основні події (Process Create, Network Connect, DNS Query тощо) без суворих винятків. Для однієї активної робочої станції це може займати в середньому шість мегабайтів на годину. Якщо ж розглядати кілька сотень машин, це швидко перетворюється на десятки гігабайт добового приросту у SIEM-сховищі. [\[13\]](#) Тоді, якщо вихідні журнали зберігаються локально, без форвардингу або окремого буфера, Event Log переповнюється й старі записи можуть бути витіснені ще до того, як SOC устигне їх обробити.

По-друге, ризик хибних опозитивних спрацювань через неконкретні фільтри. Наприклад, якщо адміністратор забороняє NetworkConnect без уточнення конкретних процесів, Sysmon починає писати кожен системний вихід до мережі: служби Windows Update, телеметрія Office, синхронізація OneDrive. Обсяг логів росте в рази, а в черзі SIEM з'являються тисячі технічних повідомлень, серед яких реальну атаку легко пропустити.

Третє обмеження стосується сумісності схем XML-конфігурацій. Кожен реліз Sysmon має власну версію схеми; якщо спробувати завантажити файл, створений для старішої або новішої версії, служба зупиниться з помилкою Event 255 – “rule version incompatible” і припинить записувати події доти, доки конфігурацію не перегенерують під актуальну схему. [\[14\]](#) На практиці це означає, що автоматичне оновлення Sysmon без синхронного оновлення XML може на кілька годин або днів залишити SOC без даних для аналізу.

Найкращі практики налаштування Sysmon передбачають використання модульного підходу, що дозволяє оптимізувати обсяг журналу подій та забезпечити його керованість. На відміну від комерційних EDR-рішень, Sysmon надає адміністратору значну гнучкість у конфігурації, проте неналежне налаштування може спричинити надмірне накопичення несистематизованих даних. Щоб уникнути надмірного потоку необроблених подій, доцільно застосовувати модульну конфігурацію. Наприклад, шаблони Olaf Hartong (sysmon-modular) дозволяють вибирати лише ті групи подій, які є найбільш релевантними для аналізу. Зокрема, можна залишити фіксацію Process Create та DNS Query, водночас вимкнувши ресурсоємні події, такі як Image Loaded, що суттєво зменшує обсяг журналу та спрощує процес аналізу.

Ще на етапі імпорту індикаторів компрометації до SIEM доцільно встановлювати термін їхньої актуальності – щонайменше 30 днів. Хеші та IP-адреси можуть швидко втрачати релевантність, і у разі відсутності механізму автоматичного видалення вони можуть спричиняти значну кількість хибнопозитивних спрацювань. Оскільки більшість платформ підтримують атрибут expiration, його варто задавати безпосередньо під час імпорту, що дозволяє автоматично деактивувати правило після встановленого періоду. [\[15\]](#)

Один із ключових аспектів ефективного збору телеметрії—спосіб доставки логів. У Windows 10 та новіших версіях канал Sysmon/Operational може бути перенаправлений через Windows Event Forwarding на окремий колектор, який забезпечує централізоване збереження даних перед їхнім надходженням до SIEM-системи. Такий підхід дозволяє мінімізувати навантаження на кінцеві пристрої та запобігти втраті подій у разі вимкнення користувачем комп'ютера. [\[16\]](#)

Варто регулярно перевіряти вплив на продуктивність, особливо якщо ввімкнено SHA-256. Sysmon може рахувати одразу кілька хешів (MD5, SHA-1, SHA-256). Чим довший алгоритм, тим більше CPU-циклів і дискових звернень. На віртуальних десктопах або слабких терміналах варто заміряти час

запуску ключових застосунків до і після активації `HashAlgorithms = "SHA256"` та за потреби обмежитись SHA-1, а SHA-256 розраховувати тільки для директорій підвищеного ризику.

1.5 Існуючі підходи до кореляції IoC з подіями системи

На перший погляд, використання списку індикаторів компрометації та перевірка кожної системи або журналу на збіг здається достатньою мірою захисту. Такий метод, відомий як плоске табличне зіставлення (lookup), є відносно простим у реалізації: коли подія надходить до SIEM, внутрішній механізм порівнює її IP-адресу або хеш з наявною базою, формуючи сповіщення у разі збігу. Проте динамічне середовище корпоративної інфраструктури створює складності для ефективного використання такого підходу. Хмарні сервери змінюють IP-адреси щогодини, легітимні файли регулярно перекомпілюються з новими хешами, а CDN-мережі оновлюють доменні CNAME-ланцюжки. Внаслідок цього один і той самий ізольований індикатор компрометації—тобто одиничний атрибут, який не враховує контекст загрози—може через кілька діб вказувати на цілком легітимний ресурс, що підвищує ризик хибнопозитивних спрацювань.

Як зазначено в розділі 1.3, ізольовані IoC значно частіше спричиняють хибнопозитивні спрацювання порівняно з індикаторами, що підтверджені додатковими даними. Це пояснюється тим, що механізм lookup не враховує контекст виявлення індикатора—зокрема, час його появи, джерело виклику та пов'язані події. У результаті система може інтерпретувати IP-адресу центру керування однаково, незалежно від того, чи з'єднання ініціював службовий процес Windows Update, чи невідомий loader.exe, який нещодавно з'явився в каталозі %TEMP%.

Кореляція додає цим атрибутам просторово-часові координати. Якщо IP-збіг з'єднується з GUID-процесом, який було запущено під непривілейованим користувачем, а через хвилину той самий GUID створює файл із хешем зі списку шкідливих – SOC отримує не три незалежні «сирі» попередження, а єдиний інцидент із послідовністю «запуск → мережа → файл». Саме така історія подій різко підвищує впевненість алерта, дозволяє автоматично підняти його в пріоритеті й звільняє аналітика від ручного складання пазлу з десятків розрізнених записів.

Атомарний lookup. У більшості платформ SIEM першим і найпростішим кроком обробки телеметрії є так званий атомарний lookup: щойно подія потрапляє до індексу, рушій аналітики зіставляє її окремі поля з таблицею індикаторів. Схема майже завжди однакова. У Microsoft Sentinel це таблиця ThreatIntelligenceIndicator, у Splunk Enterprise Security – KV-lookup, у Elastic Security – data-view з полями threat.indicator.*. Механізм працює як звичайне злиття двох наборів даних: ліворуч – потік подій, праворуч – перелік ІоС; якщо значення збігаються, формується тег або новий запис у журналі. [\[17\]](#), [\[18\]](#)

Основними перевагами методу lookup є висока швидкість обчислення та простота застосування, що робить його доступним навіть для менш досвідчених користувачів. Адміністратору достатньо оновити список ІоС для актуалізації правил детекції. Завдяки цьому lookup залишається ключовим механізмом у SOC-процесах, забезпечуючи оперативне реагування на потенційні загрози. Зокрема, цей підхід дозволяє негайно заблокувати IP-адресу або хеш файлу, якщо він виявлений у списку компрометованих об'єктів, що мінімізує ризики для інформаційної системи.

Простота даного методу водночас є його недоліком. Таблиця не надає ні контексту часу, ні інформації про те, хто саме ініціював подію. Один і той самий хеш може належати і легальному інсталятору, і шкідливому ПЗ. Lookup, позбавлений поведінкового оточення, не здатен провести цю межу, тому за

великої кількості індикаторів швидко накопичує хибнопозитиви та перевантажує аналітиків малоінформативними попередженнями.

Саме усвідомлення цих обмежень підштовхнуло індустрію до складніших моделей кореляції, де атомарний збіг використовується лише як початковий етап, а остаточний вердикт виноситься з огляду на контекст, процес і послідовність дій – ці підходи й розглянемо далі.

Кореляційні правила «першого порядку». Після того як базовий lookup відмітив збіг «подія ↔ IoC», більшість SIEM-платформ роблять наступний логічний крок: вони додають до перевірки друге джерело. Ідея проста: якщо той самий атрибут з'явився у двох різних потоках телеметрії— скажімо, IP-адреса спершу зафіксована мережевою IDS, а через кілька секунд бачиться в системному журналі Sysmon, – то ймовірність помилки різко падає. Такий «подвійний збіг» і називають кореляцією першого порядку.

Splunk Enterprise Security: наприклад, у Splunk це реалізує спеціальний оператор threatmatch. Він бере регулярний потік подій, підтягує в пам'ять набір індикаторів, а потім дозволяє, всередині одного пошуку, одночасно порівняти кілька полів і відразу ж згенерувати notable event. Прикладом може бути кореляційний пошук, який об'єднує Suricata-лог і Sysmon-лог за IP-адресою: перша частина SPL вибирає мережеві алерти, друга – локальні процеси, а threatmatch зводить їх, показуючи, який саме процес вивів трафік на «чорний» хост. [\[19\]](#)

Поведінкові графи в EDR і XDR – як IoC «вшиваються» у ланцюжок дій. Класичні SIEM-джойни зводять подію й індикатор лише за однією ознакою, тоді як сучасні EDR/XDR-платформи зберігають усю життєдіяльність системи у вигляді поведінкового графа. Кожен процес, файл, мережеве з'єднання чи ключ реєстру стає вершиною, а відношення «запустив», «створив», «відкрив сокет» – ребром. Завдяки такій моделі

індикатор більше не існує «сам по собі»: він одразу лягає у контекст попередніх і наступних дій.

CrowdStrike Threat Graph. У Falcon-агента кожний endpoint-євент миттєво штовхається до хмарної бази «Threat Graph». Це, по суті, розподілена GraphDB, яка «пам'ятає» всі зв'язки процесів, файлів та мережі за роки спостережень і дозволяє шукати збіги IoC як у реальному часі, так і ретроспективно. Якщо на диску з'явився файл із хешем зі списку шкідливих, платформа одразу «підтягує» його у граф і перевіряє, чи був цей файл дочірнім до RDP-сесії, прийшов поштою чи завантажився з браузера. Таким чином, один атомарний IoC автоматично опановує повноцінну поведінкову біографію. [\[20\]](#)

Microsoft Defender XDR. У Defender усе починається з process tree, який у порталі відображається в реальному часі, а всередині системи зберігається як граф об'єктів. Коли виявлено алерт, Fusion-движок поєднує його із DNS-запитами, файлами та користувацькими діями, вибудовуючи incident graph: один запис містить і мережевий IoC, і ім'я виконуваного файлу, і ланцюг батьківських процесів [\[21\]](#).

Графові бази та SQL-less аналітика: коли події й індикатори утворюють цілісну «павутину». Поріг, до якого сягає кореляція «першого порядку», – це всього-на-всього збіг одного атрибута у двох логах. У реальному розслідуванні цього часто замало: аналітик прагне побачити ланцюжок дій – хто що створив, куди підключився, які файли залишив, і як усе це сходиться на єдиній IP-адресі чи хеші. Щоби зберегти такі послідовності, сучасні платформи заносять сирі події до графових сховищ і відкидають традиційний SQL-погляд «таблиця-рядок».

У графовій моделі вузлом стає будь-яка сутність: процес, файл, мережевий сокет, користувач або сам IoC. Ребро ж пояснює взаємозв'язок – «spawned», «wrote», «resolved», «matched IOC». На відміну від РСУБД, де

необхідна попередня схема з жорсткими зовнішніми ключами, граф приймає новий тип вершини без міграцій; тому, коли скажімо, на хості запускається новий виконуваний файл `setup123.exe`; Sysmon фіксує це як подію `Process Create` і передає хеш файлу та GUID процесу. У графовій базі одразу з'являється новий вузол `File` із властивістю `sha256` і ребро `(:Process)-[:EXECUTED]->(:File)`, що прив'язує його до процесу-ініціатора. Жодної «міграції» чи зміни схеми не потрібно: двигун просто додає ще один тип вершини та ребра до вже існуючої структури. У підсумку нова інформація про файл миттєво влітає у «павутину» попередніх процесів та мережевих з'єднань. Neo4j у своїй документації прямо називає такий підхід «digital twin» інфраструктури, бо він віддзеркалює мережу у вигляді взаємопов'язаних об'єктів та подій. [\[22\]](#)

DAG (Directed Acyclic Graph) – це орієнтований граф, у якому всі ребра мають напрямок, а цикли принципово відсутні: тобто, рухаючись по стрілках, неможливо повернутися у вже відвідану вершину. Завдяки цій властивості кожен вузол має однозначний набір попередників, а відстань (кількість кроків) від будь-якої точки до «кореня» графа легко підрахувати. Спроба реконструювати атаку як простий список подій швидко натрапляє на петлі й повтори: процес може відкривати файл десятки разів, сокет – переустановлюватися. Коли ж ті самі відносини малюються як напрямлений ациклічний граф (DAG), ланцюжок утворює чітку «стрічку часу» без зациклень: кожна подія має єдиного попередника, а відстань від IoC до точки входу легко виміряти в ребрах. Для інцидент-response це критично: DAG дає змогу одним запитом витягти цілий сценарій, скоротити ручну реконструкцію й оперативно відповісти на питання «що сталося після події X?».

У підсумку графові сховища та їхні без-SQL мови (Cypher, Gremlin) перетворюють громіздкий масив журналів на гнучку модель, де IoC одразу влітає у ланцюжок дій. Це не лише підвищує точність детекції, а й робить

розслідування швидким і візуально зрозумілим – саме таку ідею ми перенесемо до авторської реалізації у наступному розділі.

Поточні обмеження та проблеми. Попри значний прогрес у методах кореляції загроз—від простих lookup-збігів до графових моделей та алгоритмів машинного навчання—сучасні системи виявлення залишаються далекими від повної досконалості. Основні проблеми можна згрупувати у такі категорії:

- Старіння індикаторів. Навіть найзахищеніші Threat Intelligence-фіди піддаються часовій інерції. IP-адреси та домени C2-серверів можуть залишатися активними лише кілька годин або днів, проте їх записи у базах загроз зберігаються місяцями. Аналогічно, хеші шкідливих файлів швидко стають неактуальними після чергового репакування. У результаті lookup може знаходити збіги там, де загроза вже зникла, а SOC витрачає ресурси на аналіз застарілих артефактів.
- - Висока вартість графових запитів. Незважаючи на ефективність графових баз даних у моделюванні атак, їх використання потребує значних обчислювальних ресурсів. Запити на глибоку кореляцію, наприклад пошук усіх пов'язаних подій за останні 90 днів, можуть займати секунди або навіть хвилини. Для підтримання прийнятної латентності необхідно розподіляти дані на «гарячі» та «холодні» шари, кешувати підграфи або масштабувати кластери, що ускладнює підтримку та підвищує витрати.
- Відсутність уніфікованої схеми даних. Формати логів суттєво варіюються між різними джерелами: Suricata використовує src_ip, Sysmon—SourceIp, а аналітики часто застосовують client.ip. Без попередньої нормалізації (наприклад, приведення до стандартів Elastic ECS, Splunk CIM, Sentinel Standard Schema) кореляційні механізми не можуть об'єднати ці дані коректно. Вирівнювання схем вимагає значних

зусиль, а будь-яка неврахована особливість або оновлення агента може призвести до втрати частини подій у процесі обробки.

Висновки до розділу 1

У першому розділі проведено системний огляд ролі індикаторів компрометації (IoC) у сучасних технологіях виявлення загроз та окреслено обмеження, які зумовлюють потребу в більш контекстному підході до їх використання.

Класифікація та функціональне призначення IoC. З'ясовано, що файлові, мережеві, поведінкові та контекстні індикатори відрізняються за глибиною й швидкістю спрацювання, а їхня інформативність безпосередньо залежить від рівня, на якому вони збираються й аналізуються.

Архітектурна триєдність засобів виявлення. Мережевий шар (IDS/IPS) забезпечує мілісекундну реакцію на зовнішні IoC, проте позбавлений відомостей про процеси та користувачів. Хостовий шар (EDR) надає детальний контекст виконуваного коду, однак не охоплює трафік інших вузлів. Кореляційний шар (SIEM/XDR) об'єднує журнали всіх підсистем, утворюючи єдине джерело істини, але вносить затримку й вимагає уніфікації схем. Зроблено висновок, що лише взаємодоповнення цих рівнів дозволяє отримати повну й достовірну картину інциденту.

Ризики ізольованих IoC. Показано, що самотній індикатор, позбавлений часово-поведінкового контексту, у кілька разів частіше породжує хибнопозитив, ніж підтверджений багатьма ознаками. Проблему посилюють динамічність інфраструктур, старіння фідів і різна якість джерел даних.

Sysmon як джерело глибокої телеметрії Windows. Обґрунтовано доцільність використання Sysmon для отримання унікальних GUID-ів

процесів, повних командних рядків, детальної мережевої та файлової активності.

Еволюція методів кореляції. Lookup-джойни забезпечують швидке реагування, але не враховують контекст. Правила першого порядку підвищують достовірність, комбінуючи хоча б два джерела логів. В свою чергу поведінкові графи в EDR/XDR та графові БД (Neo4j, Neptune, Elastic Graph) дозволяють моделювати повний сценарій атаки у вигляді спрямованого ациклічного графа.

Відкриті проблеми. Серед головних викликів виокремлено старіння ІоС, високу вартість масштабних граф-запитів та постійну потребу у нормалізації даних.

2 ЗАПРОПОНОВАНИЙ ПІДХІД ДО КОРЕЛЯЦІЇ ІоС З ПОДІЯМИ

В даному розділі буде описано рішення для усунення цієї проблеми – побудову спрямованого ациклічного графа (DAG), у якому вузли-події та вузли-індикатори автоматично взаємопов’язані на основі логічних і часових залежностей. Такий підхід дозволяє сформувати цілісну модель компрометації та підвищити точність аналізу інцидентів.

2.1 Дані та ключові об’єкти

Для побудови кореляційного графа достатньо п’яти базових сутностей, що відображають основні дії операційної системи та самі індикатори компрометації.

Таблиця 2.1 - Основні типи вузлів графа та обов’язкові атрибути

Сутність (лейбл вузла)	Семантика графі	Мінімальний набір атрибутів
Process	Запуск або існування процесу	guid (унікальний для життєвого циклу), image (повний шлях), sha256, user, timestamp
File	Створення чи модифікація файла	path, sha256, size, timestamp
Network	Відкриття сокета, DNS-запит	src_ip, dst_ip, dst_port, protocol, timestamp
Registry	Операція з реєстром Windows	path (ключ або значення), action (add / set / delete), timestamp
Indicator	Атомарний ІоС будь-якого типу	value (IP, SHA-256, домен тощо), ioc_type, confidence, source, first_seen

Кожен вузол із цього набору уже може брати участь у кореляції:

- Process зв’язується з дочірнім Process через guid → parent_guid;

- File та Network одержують посилання на процес-ініціатор;
- Indicator приєднується ребром DETECTED_IN до вузла, у якому зафіксовано збіг.

В табл. 2.1 подано «карту домену даних» (Data object map): короткий перелік атрибутів, що демонструє, як типи об’єктів будуть співвідноситись у моделі. Ця схема буде вихідною точкою для побудови логічних зв’язків.

2.2 Графова модель та типи ребер

Попередній підпункт задає п’ять лейблів вузлів; тепер визначимо, які зв’язки (ребра) з’єднують ці вузли й за якими правилами вони формуються під час імпорту журналів. Усі ребра спрямовані; ми використовуємо існуючі атрибути журналів як готові сигнали для побудови графових зв’язків (див. табл. 2.2).

Таблиця 2.2 – Умови формування ланцюга компрометації у графі

Тип ребра	Формат (start end) →	Джерело даних / умова побудови	Приклад у графі
PARENT_OF	Process → Process	якщо у події є пара ProcessGuid / ParentProcessGuid. Створюється під час завантаження ID 1 «Process Create».	GUID {...0700} породив дочірній процес GUID {...730f}
CREATED	Process → File або Process → Process	а) подія ID 11 «File Create» → ребро до File.path; б) повторний ID 1, коли parent ≠ child.	Процес GUID {...720b} створив файл c:\temp\payload.ps1

Кінець таблиці 2.2

CONNECTED_TO	Process → Network	подія ID 3 «Network Connect» або ID 22 «DNS Query»; кінець ребра зберігає dst_ip, dst_port, protocol.	PowerShell GUID {...720b} підключився до 140.82.121.4:443
DETECTED_IN	Indicator → (Process File Network)	виникає, коли сканер або TI-фід позначив збіг: hash, cmdline, ip, domain тощо.	IoC “powershell.exe” пов’язано з процесом GUID {...720b}

Після того як модуль Normalizer упорядковує сирі Sysmon-записи у спільний словник, кожна подія проходить однакову процедуру «графового» увімкнення. Спершу система визначає, які два вузли подія пов’язує: наприклад, запуск нового процесу додає пару «батьківський → дочірній процес», а створення файлу – «процес → файл». Далі виконується команда MERGE: якщо такий вузол уже є в базі (визначається за унікальним ключем – GUID процесу, повний шлях файлу або IP-адреса), відбувається лише оновлення властивостей; якщо ні – створюється новий екземпляр. Результатом завжди стає єдине, а не дубльоване ребро між конкретними об’єктами. До ребра одразу прикріплюється часовий штамп первинної події, тому навіть при повторному імпорті його можна швидко відфільтрувати за діапазоном часу. Сукупність таких дій гарантує, що загальний граф залишається спрямованим і ациклічним: процес ніколи не виявиться власним предком, а файл – власним творцем, тому петлі принципово не виникають.

Відповідність атрибутів реальним журналам. Кожна назва властивості в схемі має прямий еквівалент у вже нормалізованих логах. Наприклад, поле ProcessGuid у Sysmon-подіях стає первинним ключем для вузла Process, а ParentProcessGuid – підставою для ребра PARENT_OF. Аналогічно DestinationIp, DestinationPort та Protocol описують вузол Network.

Якщо деякі атрибути, наприклад size у File, відсутні у конкретній події, вузол усе одно створюється: обов'язковим залишається лише ключ path. Для випадків, коли до системи надходить «голий» IoC (скажімо, з CSV-фіда), створюється лише вузол Indicator; зв'язок DETECTED_IN додається пізніше, коли в журналах уперше зафіксують відповідний збіг.

Кореляційна модель дозволяє сформувати детальну карту поведінки хоста, що охоплює всю активність – від кореневого процесу до найменшого файлу або мережевого пакета. Індикатор компрометації (IoC), щойно внесений до фіду, автоматично інтегрується у відповідний вузол графа: хеш PE-файлу зв'язується з вузлом File, а IP-адреса – з вузлом Network. Від такого IoC можна одним запитом пройти до першої батьківської програми, переглянути, які об'єкти чи з'єднання з'явилися після неї. Для аналітика це означає готовий сценарій атаки без необхідності вручну «зшивати» окремі журнали; для автоматизованого рушія – можливість миттєво оцінити глибину та ширину компрометації, на основі чого визначатиметься підсумковий бал ризику.

2.3 Алгоритм нарощування графа з журналів

Далі розглянуто процес перетворення вихідних даних—подій Sysmon та локальних IoC-збігів – у вузли та ребра графової моделі в Neo4j. Усі етапи реалізовано за допомогою Python-скриптів, що забезпечують автоматизовану інтеграцію даних у структуру графової бази.

1. **Збирання сирих подій (Collector).** Службовий скрипт читає канал Microsoft-Windows-Sysmon/Operational через API EvtQuery, бере до тисячі останніх записів і зберігає їх у events_collected.json. На цьому етапі структура ще не уніфікована: частина атрибутів міститься усередині поля Message.

2. **Нормалізація (Normalizer).** Скрипт проходить кожний запис, дістає з поля `Message` лише атрибути, потрібні для кореляції—`ProcessGuid`, `ParentProcessGuid`, `Image`, `Hashes`, `DestinationIp`, тощо – і пише плоский JSON `events_normalized.json`. Завдяки цьому всі подальші Cypher-запити працюють із єдиним словником ключів.
3. **Підвантаження ІоС-артефактів.** Паралельно модуль, відповідальний за збір ІоС, сканує тимчасову директорію, процеси та автозапуски, додає знайдені збіги до `iocs_scanner_extended.json`. Кожен запис уже містить тип (`file`, `process`), сам індикатор (`hash` або фрагмент `cmdline`) і позначку часу – цього досить, щоб згодом створити вузол `Indicator`.
4. **Завантаження у Neo4j (Importer).** Компонент-імпортер відкриває обидва JSON-файли, а далі для кожного елемента виконує Cypher-послідовність:


```
MERGE (e:Event {guid:$ProcessGuid})
SET e += $props
```

 або, для ІоС,


```
MERGE (i:Indicator {value:$hash, ioc_type:'file.hash'})
MERGE (p:Process {guid:$ProcessGuid})
MERGE (i)-[:DETECTED_IN {timestamp:$ts}]->(p)
```

 Така конструкція гарантує, що дубль вузлів не виникає: `MERGE` шукає існуючий об'єкт за ключем інакше створює новий.
5. **Нарощування зв'язків Process → Parent (Builder).** Окремий скрипт виконує один агрегований запит: він знаходить усі події, де заповнені `ProcessGuid` та `ParentProcessGuid`, і з'єднує відповідні вузли ребром `PARENT_OF`. Після виконання виводиться кількість нових зв'язків – це швидкий тест цілісності DAG.
6. **Оновлення.** Усі попередні кроки можна проганяти за розкладом; завдяки `MERGE` другої вставки того самого `GUID` просто оновлять властивості, а не створять клон вузла. Таким чином граф може розширюватись без очищення бази.

У результаті цих кроків потік журналів перетворюється на DAG, де кожен новий індикатор одразу опиняється в причинно-наслідковому контексті процесів, файлів і мережових дій. Саме на цьому ґрунті буде визначено, як відокремлювати справжні ланцюги компрометації від ізольованих, малозначущих збігів.

2.4 Правила формування ланцюга компрометації

Після того як події й індикатори потрапили до графа, потрібно відокремити серед тисяч вузлів ті підграфи, які справді ілюструють послідовну, логічно з'єднану атаку. У межах проекту така підструктура дістала назву ланцюг компрометації. Її побудова ґрунтується на трьох обов'язкових і двох допоміжних правилах.

1. **Наявність «якоря»-ІоС.** Пошук починається з будь-якого вузла Indicator. Саме індикатор – IP-адреса з фіда, шкідливий хеш або підпис shell-коду YARA – задає точку відліку. Без такого «якоря» система не витрачає ресурси на побудову ланцюжків: ізольовані події залишаються фоновими.
2. **DAG-умова.** Усі події, прилеглі до вузла ІоС і з'єднані з ним ребрами, мають утворювати спрямований ациклічний підграф. Це гарантує, що послідовність справді розгортається в часі й немає штучного «повернення» від нащадка до предка (наприклад, коли службовий процес багаторазово перевідкриває той самий файл).
3. **Часове вікно.** Щоб уникнути псевдокореляцій, система накладає обмеження на різницю міток часу між вузлами. У прототипі використано інтервал ± 120 с навколо «якоря». Якщо, скажімо, процес запустився за п'ять хвилин до мережевого ІоС, він не увійде до ланцюга: імовірність випадкового збігу занадто велика.

4. Мінімальна довжина та насиченість. Підграф уважаться релевантним лише тоді, коли містить:

- щонайменше три ребра різного типу (наприклад, DETECTED_IN, PARENT_OF, CONNECTED_TO);
- глибину не менше двох переходів від IoC до кореневого процесу або користувача.

Саме така «масивність» має різко скоротити хибнопозитиви: одиничний збіг хешу з файлом без мережових дій чи дочірніх процесів переважно виявляється шумом.

5. Закінчення ланцюга. Ланцюг обривається в момент, коли далі немає подій, що відповідають і умовам часу, і правилу DAG. Типово це відбувається на системних процесах-коренях (services.exe, wininit.exe) або на користувацькому EXE, з якого й почалась компрометація. Завдяки цьому аналітик одразу бачить «точку входу» та «точку виходу» шкідливої активності.

Сукупність описаних правил перетворює розрізнені спрацювання на цілісну, часово впорядковану траєкторію. На рис. 2.1 зображено типовий ланцюг компрометації, який утворився в нашій базі після обробки журналів та IoC-фіда. Початком слугує вузол IoC (червоний): це хеш або інший атомарний індикатор, отриманий зі стрічки загроз. IoC компонент пов'язано ребром DETECTED_IN з Process A – тим самим процесом, у якому вперше зафіксовано збіг (наприклад, у його образі або командній строці знайшовся відповідний хеш). Process B породжено з Process A; зв'язок позначено ребром PARENT_OF, сформованим за парою ParentProcessGuid / ProcessGuid. Процес B створює файл payload.ps1 (помаранчевий вузол) – це відображає ребро CREATED. Той самий процес установлює з'єднання з IP-адресою 185.199.11.13 (зелений вузол Network) – ребро CONNECTED_TO.

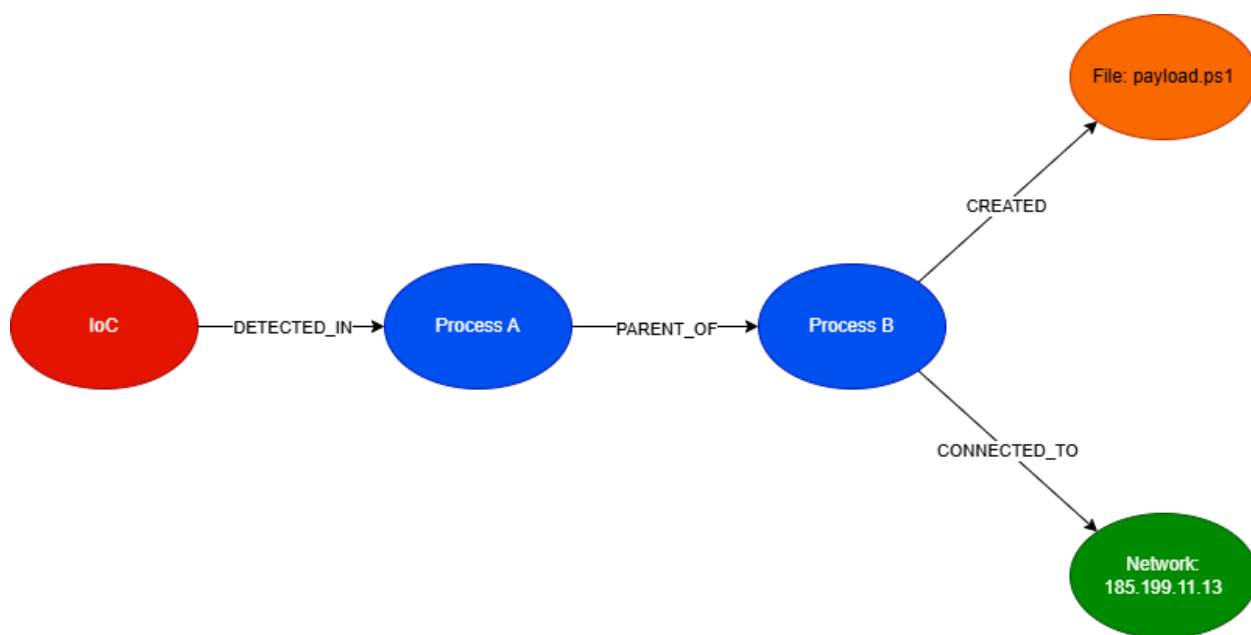


Рисунок 2.1 – Діаграма-приклад ланцюга компрометації у графі

2.5 Критерії підтвердженої компрометації

Граф-орієнтовані правила присвоєння Severity. Оскільки зовнішня оцінка довіри до індикатора не завжди доступна, остаточний рівень серйозності визначається за структурою підграфа, що утворився навколо IoC. У таблиці 2.3 наведено мінімальні умови, котрі система перевіряє послідовно – від найвищої категорії до найнижчої.

Таблиця 2.3 – Критерії High / Medium / Low для компрометації

Категорія	Обов'язкові умови (усі мають виконуватися)
High	1. Ланцюг містить не менше трьох різних типів ребер (наприклад, DETECTED_IN, PARENT_OF, CONNECTED_TO). 2. У підграфі присутні щонайменше три унікальні об'єкти-сутності, серед яких обов'язково є мережевий вузол Network або зміна «постійності» (Registry, Startup File). 3. Глибина (найдовший шлях від IoC до кореня процесів) ≥ 3 .

Кінець таблиці 2.3

Medium	1. Ланцюг містить два різні типи ребер (наприклад, DETECTED_IN + CREATED). 2. Є щонайменше два процесні рівні (процес-батько й дочірній) або поєднання «процес → файл/мережа». 3. Загальна кількість вузлів (крім IoC) ≥ 3 .
Low	Будь-який підграф, що порушує хоча б одну умову Medium

High вимагає одночасно різноманіття дій (мінімум три типи ребер) і доказ того, що активність вийшла назовні (мережевий вузол) або закріпилася в системі (автозапуск, реєстр). Часове обмеження гарантує, що це один безперервний епізод, а не артефакти різних сесій.

Medium фіксує ситуацію, коли шкідливий код уже помітно «розгорнувся» (є принаймні два переходи й кілька об'єктів), але ще не підтвердив контроль над мережевим каналом чи стійкістю.

Low охоплює ізольовані збіги: один процес і IoC-хеш, разовий мережевий пакет тощо. Такі спрацювання логуються, але ескалація не виконується без додаткових подій.

Застосування цих трьох простих правил дозволяє ранжувати підграфи без зовнішнього “confidence” та негайно відокремлювати справжні ланцюги атаки від випадкових або застарілих спостережень.

2.6 Архітектурна схема компонентів

Нижче, на рисунку 2.2, подано загальну топологію запропонованого рішення і перелік атрибутів, без яких побудова графа була б неможливою.

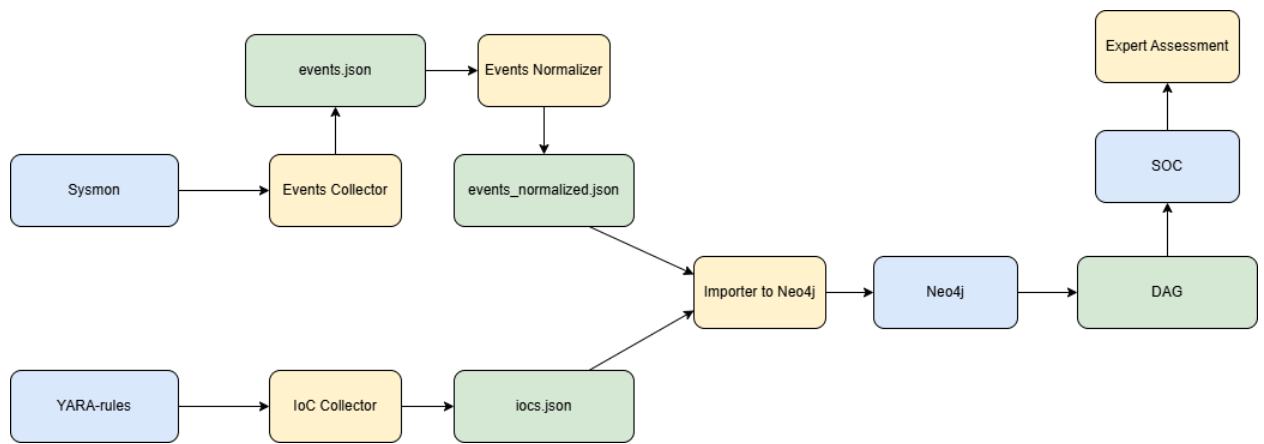


Рисунок 2.2 – Архітектура реалізованого прототипу

- **Events Collector** читає канал Microsoft-Windows-Sysmon/Operational та зберігає сирі XML-події у events.json.
- **IoC Collector** обходить файл-систему та процеси, фіксує локальні збіги з фідом і пише iocs.json.
- **Events Normalizer** приводить обидва потоки до єдиної схеми атрибутів і формує events_normalized.json.
- **Importer to Neo4j**: імпортує дані з .json файлів до Neo4j, перетворюючи їх на вузли й ребра. Наступним кроком через bolt-конектор MERGE-ить їх, дотримуючись правил й утворюючи ациклічний граф.
- **Expert Assessment (Scoring)**. Аналітик, спираючись на візуалізацію підграфа та прості правила, виділяє підграфи та присвоює їм категорію High / Medium / Low. За результатами цієї оцінки можна побудувати пріоритетність реагування на ту чи іншу потенційну компрометацію.

Висновки до розділу 2

У другому розділі було сформульовано й деталізовано концептуальну частину запропонованого рішення, яке поєднує індикатори компрометації з журнальними подіями в напрямленому ациклічному графі.

Визначено п'ять ключових сутностей – Process, File, Network, Registry, Indicator – та мінімальний набір атрибутів, достатній для кореляції без надлишкових залежностей від конкретних форматів логів.

Запропоновано чотири семантично зрозумілі зв'язки (PARENT_OF, CREATED, CONNECTED_TO, DETECTED_IN). Вони повністю покривають

причинно-наслідкові відносини, що виникають у Windows-середовищі й підтверджуються фрагментами реальних подій.

Продемонстровано правила формування ланцюга компрометації. Ланцюг починається на вузлі-IoC, залишає тільки DAG-фрагменти й вимагає мінімальної «масивності» (різні типи ребер, кілька процесних рівнів). Це забезпечує баланс між чутливістю й точністю.

Спрощена трирівнева шкала ризику. Підграфи класифікуються за власними ознаками: різноманітністю взаємодій, кількістю об'єктів і глибиною. Рівні High / Medium / Low легко читаються аналітиками й дають підставу для подальшого прийняття рішень.

Представлена pipeline-схема підтвердила, що для побудови графа досить тонкого, уніфікованого словника. Додати нове джерело означає лише створити ще один Normalizer-модуль.

Розділ доводить життєздатність концепції: від вхідних Sysmon-подій і локальних IoC до ранжованих інцидентів.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСТОСУНКУ ТА РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ

У другому розділі було викладено концептуальну схему графової кореляції ІоС із системними подіями. Далі буде розглянуто реалізацію – від збирання журналів до візуалізації ланцюгів у Neo4j та валідації методики на контрольних сценаріях.

3.1 Архітектура реалізованого рішення

Компонент збору подій (Raw Collector). Це перший модуль, він регулярно переглядає системний канал Microsoft-Windows-Sysmon/Operational, забирає найсвіжіші записи й перетворює їх на зручний для подальшої обробки формат. У кожному циклі робиться запит через win32evtlog.EvtQuery, причому події читаються у зворотному порядку – від найновішої до старіших. Щойно назбирається попередньо заданий ліміт (за замовчанням тисяча записів), запит припиняється: цього достатньо, щоб на активній робочій станції перекрити 5–10 хвилин життєдіяльності, але не створити дубль того, що вже було опрацьовано.

Сирий XML кожного Sysmon-запису конвертується у словник Python: із блоку <System> береться точний час (SystemTime) і номер події (EventID), а сам текстовий опис лишається у полі Message. Ніяких очищень чи фільтрів на цьому етапі не робиться навмисно: основне завдання Raw Collector – зібрати усі події такими, якими їх передав канал, і зберегти у файл events_collected.json. Файл ведеться у форматі JSON Lines: по-перше, це дозволяє стрімінгово писати великі масиви даних без затримки, а по-друге, спрощує подальшу обробку.

Щоб уникати повторів, перед виходом скрипт записує номер останньої прочитаної події у службовий файл `.bookmark`; під час наступного запуску Collector відновлюється саме з цього місця й бере лише нові записи. Якщо доступ до каналу тимчасово недоступний або скрипт не має прав, виняток перехоплюється, а в журналі Windows Application з'являється попередження – дані не губляться, а процес не переходить у нескінченний цикл помилок.

У тестовому середовищі (локальний SSD, Python 3.11) обробка п'ятдесяти тисяч Sysmon-подій займає менш ніж пів секунди й споживає приблизно тридцять мегабайт оперативної пам'яті. Таким чином компонент можна запускати хоч кожні п'ять хвилин навіть на малопотужних робочих місцях, не боячись помітного навантаження на ЦП чи диск.

Підсумовуючи, Raw Collector забезпечує стабільний «сирий» потік подій, який не змінюється та не фільтрується; усі подальші перетворення – розбір полів, нормалізація й кореляція – виносяться в окремі модулі.

IoC Collector зосереджений на суб'єктивних, суто локальних ознаках компрометації. Скрипт виконує кілька послідовних проходів по системі.

Сканування тимчасових директорій і завантажень. Ітерація починається з обходу `%TEMP%`, `%APPDATA%\Downloads` та інших швидкоплинних розташувань. Для кожного знайденого виконуваного файлу обчислюється SHA-256; якщо у каталозі є розгорнутий фід відомих хешів («чорний список»), скрипт одразу помічає збіг.

Перевірка процесів у реальному часі. Використовується модуль `psutil`: перебираються усі PID, і для кожного процесу збираються `exe`, `cmdline`, `user`. Командний рядок порівнюється з простими евристичними шаблонами (наприклад, `-enc`, `powershell`, `-nop`); за наявності збігу формується IoC типу `process` із позначкою «підозріла командна строка».

Аналіз автозапусків і записів реєстру. Скрипт читає ключі `Run` у гілках `HKLM` та `HKCU`, а також папку `Startup` профілю користувача. Будь-який шлях,

що веде у Temp, AppData або містить незвичне розширення (.js, .vbs), теж перетворюється на локальний індикатор.

YARA-правила (опційно). Якщо в конфігурації увімкнено, IoC Collector застосовує набір lightweight-YARA, спрацювання якого формує IoC типу `yara_match` із назвою правила.

Усі зібрані записи записуються у файл `iocs_scanner_extended.json` – так само у форматі JSON Lines. Кожен рядок має поля:

```
{
  "timestamp": "2025-04-12T09:45:03Z",
  "ioc_type": "file.hash",
  "value": "1395...8E7",
  "context": { "path": "C:\\Temp\\loader.exe" },
}
```

Сканування тимчасових папок та процесів займає $\approx 1-2$ с на середньому ноутбукі й використовує до 5 % CPU. Скрипт працює лише в режимі читання; жодних змін у файловій системі чи реєстрі не робиться, тому ризик впливу на продуктивність мінімальний.

Таким чином IoC Collector забезпечує внутрішню, «контекстну» сторону нашої моделі: він постачає атомарні індикатори, які зв'язуватимуться з подіями Sysmon у графі й допомагатимуть відрізнити реальний шкідливий процес або файл від випадкових збігів.

Event Normalizer виконує функцію перетворення зібраних Sysmon-подій в однорідний структурований формат, придатний до подальшого імпорту та кореляції.

Скрипт відкриває файл `events_collected.json`, у якому кожний об'єкт є фактичним відбитком XML-події Sysmon. Головна проблема на цьому етапі – усі значущі атрибути заховані глибоко у блоку `<Message>`, а їхній формат різниться від одного Event ID до іншого.

У модулі оголошено список `FIELDS_OF_INTEREST`: `Image`, `CommandLine`, `TargetFilename`, усі GUID-и й мережеві реквізити «`DestinationIp / DestinationPort`». Під час проходження по події `Normalizer` дістає із словника `Message` лише ці ключі. Якщо ключа немає (скажімо, у ID 1 немає `TargetFilename`) – поле просто не створюється, жодних «порожніх» значень «N/A» не додається. Завдяки цьому обсяг фінального JSON зменшується приблизно у п'ять-шість разів без втрати потрібної інформації.

Для кожної події створюється новий словник, де на верхньому рівні одразу лежать: час (`timestamp`), номер події (`event_id`), GUID-и процесу та батьківського процесу, образ, хеш, IP-адреса тощо. Весь оригінальний блок `<Message>` зберігається окремо у полі `raw_message` – за необхідності його можна буде проаналізувати пізніше, але він не заважатиме швидкій обробці.

Усі нормалізовані об'єкти складаються у `events_normalized.json`. Формат – JSON Lines, тому подальший модуль, що виконує імпорт, може читати файл потоково й одразу виконувати Cypher-операції, не завантажуючи усе у пам'ять.

Плоска структура гарантує, що скрипт `import_to_neo4j.py` завжди знайде очікуваний ключ. Зменшення розміру файлу скорочує час між збором подій та їх появою у графі, що прямо впливає на оперативність аналізу. На тестовому файлі у 10 000 записів нормалізація займає близько двох-трьох секунд на середньому CPU.

Після завершення `Normalizer` виводить у консоль повідомлення «Normalized N events → `events_normalized.json`», що слугує простим підтвердженням успішної підготовки даних.

Скрипт Importer переносить обидва набори даних (зібрані IoC та нормалізовані Sysmon-події) в Neo4j і одразу вибудовує процесний DAG.

Під час підготовки середовища відкривається bolt-сеанс neo4j://localhost:7687, перевіряються (або створюються) обмеження унікальності на Process.guid, File.path, IoC.file, Network.ip+port.

Імпорт записів, створення й оновлення вузлів IoC відбуваються першими. Для кожного шляху або PID формується вершина :IoC з полями file, pid, yara, timestamp. Робиться це попередньо, аби далі легко з'єднати індикатори з реальними процесами чи файлами.

Кожен об'єкт із нормалізованих подій опрацьовується в одному виклику tx.run:

- Process-вузол. MERGE (:Process {guid}), одразу ставляться властивості pid, image, cmd, timestamp.
- PARENT_OF. Якщо є ParentProcessGuid, виконується MATCH + MERGE, чим формується ребро «батьківський → дочірній процес». Саме ці зв'язки й роблять граф спрямованим ациклічним.
- File та CREATED. Для подій ID 11 створюється вузол :File і ребро (:Process)-[:CREATED]->(:File).
- Network та CONNECTED_TO. Для ID 3, якщо присутні IP і порт, формується вузол :Network і ребро (:Process)-[:CONNECTED_TO]->(:Network).

Після вставки процесу або файла скрипт порівнює їхні атрибути зі списком локальних IoC. Збіг шляху або PID – і виникає ребро (:IoC)-[:DETECTED_IN]->(:Process | :File). Так індикатор одразу вводиться у поведінковий контекст.

Увесь імпорт виконується в одній транзакції `execute_write`, тому граф або оновлюється повністю, або не змінюється взагалі. На локальному SSD пакет із ~500 подій і ~50 IoC імпортується за ≈ 4 с.

Після завершення скрипта в базі міститься повноцінний спрямований ациклічний граф: від IoC до кореневого процесу й далі до файлів чи мережевих виходів. Тож скрипт закриває дві задачі разом – імпорт даних та побудову процесного DAG. Уся решта ланцюжка (візуалізація, експертна оцінка High / Medium / Low) працює поверх отриманого графа, без додаткових скриптів.

3.2 Побудова та візуалізація графа в Neo4j

Після того як Importer завершив роботу, усі вузли й ребра вже зберігаються у базі. Наступне завдання аналітика – швидко знайти ланцюги й оцінити, чи відповідають вони правилам пункту 2.4. Для цього достатньо зробити Cypher-запит у Neo4j Browser або Bloom.

MATCH (n) RETURN n – відобразить всі атрибути, створені в графовій БД та зв'язки між ними.

На рис. 3.1 відображено весь вміст бази після імпорту: запит без обмеження глибини вивів кожний вузол, який наразі існує у Neo4j.

Умовна розмальовка вузлів:

- Червоний – Process
- Помаранчевий – IoC
- Пісочний – File
- Зелений – Network

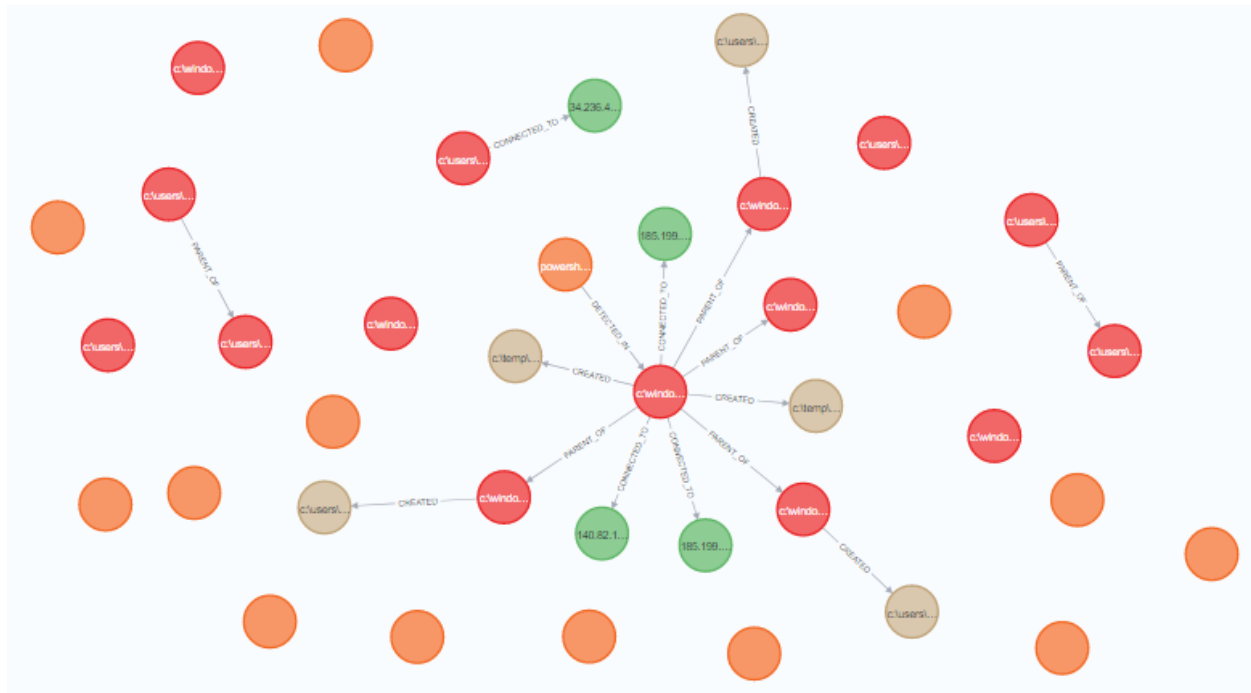


Рисунок 3.1 – Візуалізація повного графа подій у Neo4j

Тут видно одиночні помаранчеві вузли – це IoC, на які спрацювали сигнатури сканера. Чітко видно, що лише один з них прив'язаний ребром DETECTED_IN до процесу (цей граф детальніше розберем в наступному підпункті). Одиночних індикаторів завжди багато: це лог-файли, невикористані тимчасові об'єкти, артефакти попередніх сесій. Вони підпадають під категорію Low і можуть бути проігноровані, поки не з'явиться подія, що реально взаємодіє з цим шляхом чи PID.

Також у візуалізації видно групи по два-три червоних і пісочних вузли з одним ребром PARENT_OF та CREATED. Це дрібні допоміжні процеси, які створювали лог-файли чи скрипти, але не відкривали мережі і не підпадали під індикатори компрометації.

Повний вивід корисний лише на початку діагностики. Видно, що всі очікувані типи вузлів (IoC, Process, File, Network) присутні, кольори й підписи ребер правильні. Але за наявності більшої кількості даних (подій та індикаторів), візуальний аналіз і пошук необхідних графів ускладнюватиметься через велику кількість шуму. Тож для подальшої роботи

замість глобального MATCH (n) доцільніше запускати таргетовані запити, наприклад:

*MATCH path = (ioc:IoC)-[*2..5]->(n)*

RETURN path;

де:

- ioc:IoC — початкова точка;
- [*2..5] — будь-яка спрямована послідовність ребер довжиною від 2 до 5 кроків;
- RETURN path — відобразити увесь підграф.

Таким Cypher запитом можливо відкинути одиничні елементи, недостатньо глибокі та не пов'язані з IoC граfi й зосередити увагу на тих, що підпадають під категорію, що нас цікавить. Приклад виводу такого запиту показано на рисунку 3.2.

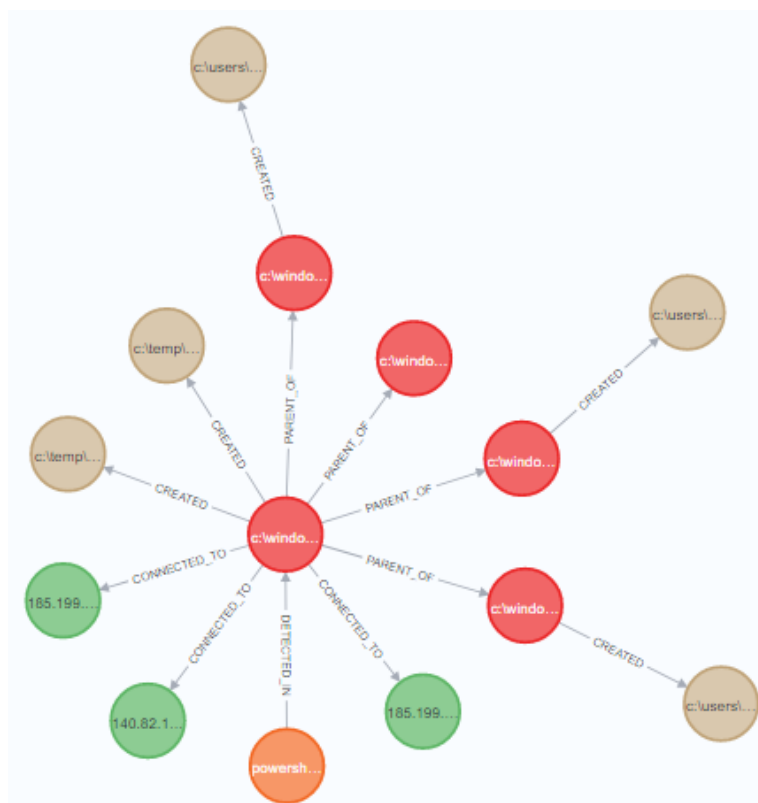


Рисунок 3.2 – Приклад глибокого підграфу зі спрямованою послідовністю ребер довжиною від 2 до 5 кроків

3.3 Тестування методики на контрольних сценаріях

Хибнопозитив: IoC без компрометації. У віртуальній машині Windows 11 із Sysmon 15.15 з базовою конфігурацією SwiftOnSecurity, у каталогі C:\Temp було створено файл evil.ps1 розміром 3 КБ. Файл жодного разу не виконувався, не був відкритий сторонніми процесами й не потрапив у механізми автозапуску. Через 30 секунд після створення стандартний збирач Sysmon зафіксував подію File Create (ID 11), а модуль IoC Collector підхопив об'єкт під час свого огляду тимчасових директорій, присвоївши йому ознаку IoC типу file.hash.

Оскільки з файлом більше нічого не відбувалося, у граф мали потрапити:

- помаранчевий вузол (:IoC) – збіг «powershell.exe» із правилом SuspiciousPowershellEncoded;
- червоний вузол (:Process powershell.exe) – скрипт копіював файл;
- пісочний вузол (:File) – безпосередньо створений за допомогою powershell

Будь-яких інших вузлів утворитися не має.

На рисунку 3.3 бачимо граф з трикутною структурою: IoC → Process → File. Дочірніх процесів (PARENT_OF) та мережевих сесій (CONNECTED_TO) немає.



Рисунок 3.3 – Підграф, відображаючий сценарій хибнопозитиву

Експертна оцінка відповідно до пункту 2.5: ланцюг містить три вузли (включно з IoC) та два різні типи ребер, має поєднання «процес → файл». Усі критерії Medium та High не виконано; отже, ланцюг подій буде класифіковано як Low Severity: будь-який підграф, що порушує хоча б одну умову Medium (порушено умову 3. Загальна кількість вузлів (крім IoC) ≥ 3)

Реальна компрометація: ланцюг з діями. У віртуальній машині Windows 11 із Sysmon 15.15 з базовою конфігурацією SwiftOnSecurity, через PowerShell було завантажено та запущено скрипт payload.ps1 із віддаленого репозиторію GitHub, а також виконано спробу зачищення слідів викликом команди Windows Event Utility, яка очищає вміст указанного журналу подій: *wevtutil.exe cl microsoft-windows-sysmon\operational*.

Послідовність подій відображено в графі на рисунку 3.4.

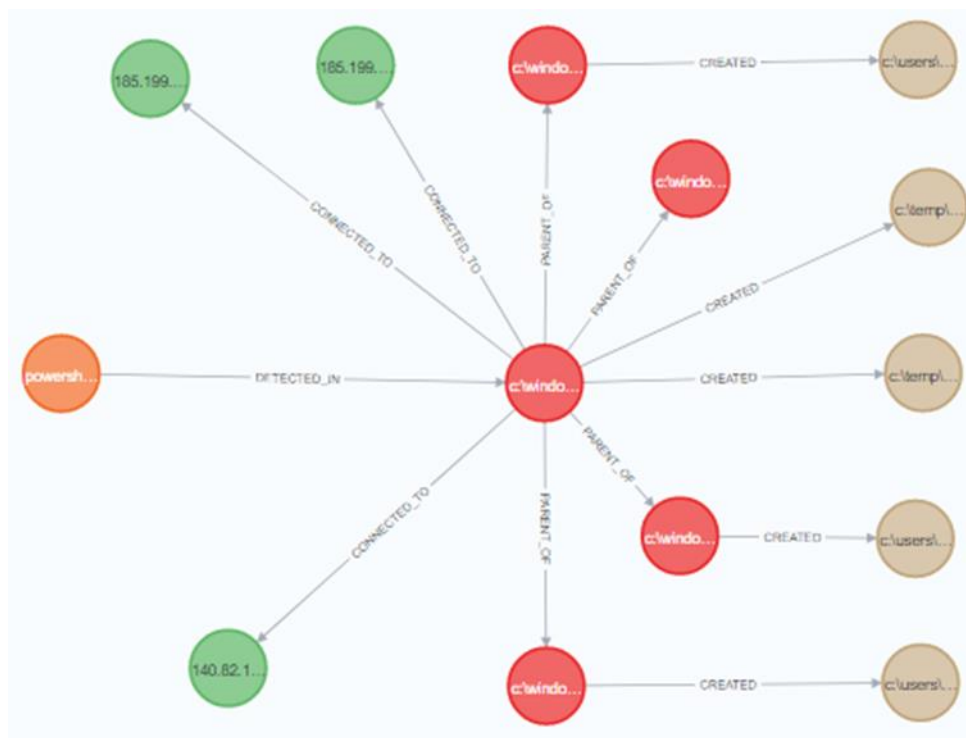


Рисунок 3.4 - Підграф, відображаючий сценарій реальної компрометації

Локальний сканер позначив процес powershell.exe (PID 7640) як IoC на підставі сигнатури SuspiciousPowershellEncoded.

Витягнуті з GUID-пари зв'язки PARENT_OF показують, що кореневий PowerShell породив три додаткові екземпляри (11844, 10416, 2572) – кожен із параметром -ExecutionPolicy Bypass -File. Таким чином підтверджено виконання зовнішнього скрипта.

Усі дочірні процеси залишили файли .ps1 у тимчасових папках; ребра CREATED пов'язують їх із відповідними процесами. Ключовим є файл C:\Temp\payload.ps1 – основний завантажувач.

Кореневий процес встановив TLS-сесії на порт 443 до трьох IP-адрес GitHub. Зв'язки CONNECTED_TO забезпечують доказ зовнішнього C2.

Окремий дочірній процес запустив wevtutil.exe cl microsoft-windows-sysmon\operational, намагаючись очистити журнал Sysmon. Це ребро PARENT_OF та вузол Process 4992 підвищують критичність події.

Експертна оцінка відповідно до пункту 2.5: ланцюг містить не менше трьох різних типів ребер, присутні щонайменше три унікальні об'єкти сутності, серед яких є мережевий вузол Network та найдовший шлях від IoC до кореня процесів = 3. Усі обов'язкові умови категорії High виконано.

3.4 Аналіз ефективності методу

Порівняння з традиційними підходами. На тій самій віртуальній машині Windows 11 розгорнуто Docker контейнер з Wazuh. До нього під'єднано Wazuh Agent, який збирає канали Microsoft-Windows-Sysmon/Operational та Security. Базові правила менеджера було доповнено двома локальними, що: фіксують запуск файлу, який входить до ТІ-фіда, виявляють пряме TCP/TLS-з'єднання на відомі IP, виявляють створення скрипта з шкідливим хешем.

В поєднанні з базовими правилами, це забезпечує спрацьовування на всі атомарні індикатори з обох тест-кейсів.

Таблиця 3.1 – Порівняльна характеристика запропонованого DAG-підходу та Wazuh 4.7

Критерій	Запропонований граф	Wazuh
К-сть інцидентів	1 High	560 окремих алертів
Кореляція	DAG: IoC → Process → File → Network	Плоский lookup + статичні OSSEC-правила
Приоритизація ключової події	Підвищена до High (всі вузли графа)	Level 5 як для події з ланцюга компрометації, так і для одиначної
Час triage SOC	1–2 хв	≥ 15 хв (ручна агрегація)
Схильність до непоміченого інциденту	Низька – ланцюг зібрано в єдиний інцидент	Вища – важливий alert губиться у фоні
Складність розгортання	Низька: Python-парсер → Neo4j, без контейнерів й великої к-ті ресурсів	Середня: Docker Desktop + nested VT-x, Wazuh Manager, агент
Платформна спрямованість	Windows	Крос-платформне рішення (Windows + Linux + macOS)

Wazuh коректно фіксує всі атомарні індикатори, однак через високе сповіщувальне навантаження (560 alerts за 20 хв) та брак процесного контексту не забезпечує оперативної ескалації. Графова модель, навпаки, автоматично поєднує хеш, запуск PowerShell, створення скриптів, мережеві виходи й подальшу чистку логів у єдиний підграф, що суттєво скорочує час реагування та мінімізує ризик пропустити реальну компрометацію (див. табл. 3.1).

Такий контраст підтверджує тезу розділу 2 – атомарне зіставлення IoC без поведінкових зв'язків не забезпечує такої повноти та точності, що й мотивує застосування графових моделей.

Переваги запропонованого рішення.

1. Узгоджений контекст «один-до-багатьох». Графове подання об'єднує Sysmon-події, мережеві алерти і локальні IoC у єдиний часо-причинний ланцюг. Аналітик отримує не фрагменти, а готову історію: хто створив файл, коли він був запущений і куди встановлено TLS-сеанс. Це зменшує когнітивне навантаження та кількість помилок у ручному зшиванні даних.
2. Зменшення шуму та F-positive. За критеріями в інциденти High/Medium потрапляють лише підграфи з кількома різнорідними ознаками (IoC + поведінка + мережа). Хибнопозитивні одиничні збіги залишаються у категорії Low і не перевантажують SOC.
3. Оперативний тріаж. На тест-сценарії один граф-інцидент обробляється < 2 хв.
4. Незалежність від Вендора і ліцензій. Neo4j Community-Edition та скрипти Python є безкоштовними; інтеграція не прив'язана до конкретної EDR або SIEM-платформи й може споживати дані будь-якого формату після нормалізації.
5. Масштабованість і візуалізація. Для розслідування складних випадків доступні можливості Cypher: пошук найкоротшого шляху, обхід у глибину, фільтрація за мітками. Готовий підграф легко показати керівництву у вигляді діаграми, не вдаючись до raw-логів.

Обмеження та ризики запропонованого рішення.

1. Залежність від якості нормалізації. Побудова правильних ребер PARENT_OF, CREATED, CONNECTED_TO можлива лише за умови, що вхідні події приведено до узгодженої. Якщо Sysmon конфігурація зміниться або поля GUID загубляться, граф може «розірватись».
2. Витрати пам'яті на великих обсягах. Neo4j утримує hot-subgraph у RAM; при постійному потоці великої кількості подій/хв

знадобиться ≥ 8 ГБ пам'яті. У промисловому середовищі потрібен кластер Neo4j Enterprise або Fabric.

3. Пасивний характер. Система не здійснює автоматичного блокування (на відміну від EDR). Рішення про реакцію приймає оператор після перегляду графа.
4. Експертна оцінка severity. Поточна реалізація застосовує правила для визначення High/Medium/Low Severity без зовнішнього Score та статистичного навчання. На нових типах атак вагові коефіцієнти доведеться корегувати вручну, або ж інтегрувати ML-модуль, що ускладнить архітектуру.
5. Затримка. Скрипт нормалізації наразі запускається пакетно; між подією та появою вузла в графі проходить ~ 60 с. Для зменшення затримки потрібно реалізувати стрімінговий конектор (Neo4j Streams / Kafka), або перенесення логіки в пам'ять EDR.
6. Обмеженість джерел. Доказовість методу прямо пропорційна різноманіттю телеметрії. Використання лише Sysmon + локальний сканер ІоС зменшує кількість типів ребер і, відповідно, різноманіття відображуваних ланцюгів подій.

3.5 Пропозиції щодо розширення функціональності

Пілотна реалізація довела концепт і успішно продемонструвала переваги графового підходу над пласким «alert flooding». Наступні кроки мають перетворити прототип на виробниче рішення, усуваючи виявлені обмеження та підвищуючи операційну готовність SOC.

Перехід від пакетної обробки до потоку подій: зменшити часовий розрив між появою події у середовищі та її відображенням у графі. Принцип роботи: події одразу публікуються у буфер обміну повідомленнями;

невеликий сервіс-конектор у режимі реального часу перетворює їх на вузли й ребра. Завдяки цьому граф оновлюється протягом секунд, що дозволяє реагувати майже миттєво.

Масштабування графа й висока доступність: забезпечити стабільність при зростанні обсягу телеметрії та відмовостійкість. Принцип роботи: дані поділяються на окремі області (наприклад, за хостами або часовими інтервалами) та зберігаються на кількох вузлах бази. До графа звертаються через шар балансування, тому навіть вихід з ладу одного вузла не зупиняє систему.

Динамічне оцінювання ризику: підвищити точність визначення High/Medium/Low без ручної оцінки та правки правил. Принцип роботи: для кожного підграфа обчислюється набір ознак (глибина, кількість типів ребер, повторюваність тощо). Модель машинного навчання присвоює вагу інциденту на основі накопиченої статистики успішних і хибних спрацювань.

Автоматизована реакція: скоротити час між виявленням та усуненням загрози.

Принцип роботи: після присвоєння високого Severity Score система викликає зовнішній інтерфейс — наприклад, блокує процес на хості або додає IP-адресу до списку блокування мережевого екрану. Дії виконуються заздалегідь підготовленими сценаріями, що мінімізує участь людини у рутинних операціях.

Кероване «старіння» індикаторів: зменшити кількість застарілих ІоС, які спричиняють зайвий шум. Принцип роботи: для кожного індикатора встановлюється строк придатності, після якого він переходить в архів, якщо не зустрічається в подіях. Актуальна база індикаторів залишається невеликою й релевантною, а отже — ефективнішою.

Висновки до розділу 3

Розроблено повний ланцюг «збирання → нормалізація → імпорт → граф-кореляція».

Чотири взаємодоповнювані скрипти (Raw Collector, IoC Collector, Event Normalizer, Importer) автоматично перетворюють Sysmon-журнали та локальні IoC у спрямований ациклічний граф Neo4j. Архітектура довела свою працездатність навіть на малопотужній робочій станції: імпорт пакета подій і індикаторів виконується без контейнерів і додаткових сервісів.

Експериментальна перевірка засвідчила коректність як для хибнопозитивного, так і для реального сценарію. У хибнопозитивному кейсі low-категорія очікувано формується лише з IoC, процесу й файлу, виконання якого не відбувається. У кейсі компрометації підграф поєднує IoC-процес, створені скрипти, зовнішні TLS-з'єднання та спробу очистити Sysmon-лог, класифікуючи інцидент як High.

Порівняльне тестування з Wazuh 4.7 підтвердило переваги підходу. Графова модель консолідує події в єдиний інцидент, скорочує triage SOC до 1–2 хв і знижує схильність до пропуску критичної активності. Wazuh генерує 560 розрізнених alert-ів, вимагаючи ручної агрегації; ключова подія губиться серед фону.

Виділено сильні сторони рішення – повний контекст, мінімальний шум, незалежність від ліцензій і великої кількості ресурсів. Разом із тим зафіксовано низку практичних обмежень: залежність від якості нормалізації та GUID-ів, RAM-витрати Neo4j при масштабуванні, пасивність (відсутність автоматичного блокування), не стрімінгова доставка подій.

Сформульовано напрямки розвитку, які мають перетворити прототип на виробничу систему: стрімінг подій, кластеризація Neo4j, ML-складова для динамічного scoring та інтеграція з EDR-API для автоматизованої реакції.

Отже, практична реалізація підтвердила теоретичні положення розділу 2: графова кореляція підвищує точність і швидкість інцидент-респонсу, залишаючись технічно доступною для впровадження у середовищах Windows без дорогих SIEM-платформ.

ВИСНОВКИ

У дипломній роботі виконано повний цикл – від теоретичного обґрунтування до прототипування та експериментальної перевірки – методу кореляції локальних індикаторів компрометації з подіями Windows-системи на основі спрямованого ациклічного графа (DAG). Отримані результати дозволяють сформулювати такі підсумкові положення.

Розв’язано проблему контекстно-незалежних ІоС. Проаналізовано недоліки традиційних lookur-підходів (п. 1.5) і обґрунтовано необхідність переходу від «плоского» зіставлення до моделі, що враховує просторово-часові зв’язки між об’єктами. Запропоновано представлення «ІоС ↔ подія» у вигляді DAG, де вершини Process, File, Network, Registry, ІоС зшиваються ребрами DETECTED_IN, PARENT_OF, CREATED, CONNECTED_TO тощо (розділ 2).

Розроблено архітектуру прикладного рішення. Побудовано чотири автономні компоненти:

1. Raw Event Collector (збір Sysmon-подій);
2. ІоС Collector (агрегація локальних та зовнішніх індикаторів);
3. Event Normalizer (приведення журналів до власної схеми ECS-подібного формату);
4. Importer (завантаження нормалізованих даних до Neo4j та формування ребер DAG).

Сформульовано формальні правила побудови та оцінки ланцюгів компрометації. Алгоритм нарощування (п. 2.3) гарантує ациклічність графа, а правила ланцюга (п. 2.4) задають умови спадковості ProcessGuid → ParentProcessGuid та транзитивні співвідношення «процес – файл – мережа». На підставі топологічних характеристик (глибина, кількість типів ребер, наявність Network/Registry-вузлів) запропоновано спрощену

триступеневу шкалу Low / Medium / High без потреби у зовнішніх score-фідах (п. 2.5).

Проведено експериментальне тестування. Хибнопозитивний сценарій («файл створено – не запущено») завершився підграфом із 2 ребрами, класифікованим експертною оцінкою як Low (п. 3.3). Реальна компрометація сформувала підграф глибиною ≥ 3 із трьома різними типами зв'язків – присвоєно оцінку High (п. 3.3). Візуалізація в Neo4j підтвердила коректність зв'язків і дала SOC-аналітику можливість «одним кліком» відстежити увесь сценарій (п. 3.2).

Порівняльний аналіз з Wazuh 4.7 підтвердив переваги підходу. При тих самих тест-кейсах запропонований граф згенерував 1 High інцидент, тоді як Wazuh створив 560 розрізнених алертів; час на аналітику скоротився з ≈ 15 хв до 1-2 хв, а схильність до непоміченого інциденту знизилася.

Практична цінність та можливості масштабування. Реалізація базується на безкоштовних компонентах (Sysmon + Neo4j Community + Python), що робить рішення придатним для малих SOC або лабораторій. Архітектура допускає горизонтальне масштабування й інтеграцію ML-скорингу як зовнішнього мікросервісу (п. 3.5).

Виявлено обмеження та напрями подальших досліджень. Ключові слабкі місця: потреба в якісній нормалізації (без якої є ризик розриву графа), різке зростання RAM при великих потоках, пасивність та ручне обслуговування Sysmon і IoC-правил. Подальший розвиток передбачає потокове завантаження, масштабований багатовузловий граф, ML-скоринг ризику та інтегровану SOAR-реакцію – усе це скорочує затримки та підвищує стійкість при моніторингу з великим обсягом даних.

Отже, поставлену мету – створити та експериментально підтвердити дієздатність прототипу, що корелює локальні IoC із подіями Windows у вигляді DAG – досягнуто повністю. Робота демонструє, що навіть без дорогих

EDR-платформ можливо значно підвищити точність виявлення інцидентів і скоротити навантаження на SOC-аналітиків, якщо доповнити атомарні індикатори поведінковою графовою логікою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Indicators of Compromise (IoC) [Електронний ресурс] – 2024 – Режим доступу: <https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/indicators-of-compromise-ioc/>
2. What Are Threat Intelligence Feeds? [Електронний ресурс] – 2024 – Режим доступу: <https://any.run/cybersecurity-blog/what-are-threat-intelligence-feeds>
3. Threat Hunting vs. Threat Detecting [Електронний ресурс] – 2024 – Режим доступу: https://www.splunk.com/en_us/blog/learn/threat-hunting-vs-threat-detecting.html
4. Graph-based Detection of Advanced Persistent Threats / Anton S. Krizhanovsky [Електронний ресурс] – 2018 – Режим доступу: <https://ltu.diva-portal.org/smash/get/diva2%3A1229171/FULLTEXT02.pdf>
5. IT Event Correlation: How It Works and Why It Matters [Електронний ресурс] – 2024 – Режим доступу: https://www.splunk.com/en_us/blog/learn/it-event-correlation.html
6. Indicators of Compromise (IoC): What They Are and How to Use Them [Електронний ресурс] – 2024 – Режим доступу: https://www.splunk.com/en_us/blog/learn/ioc-indicators-of-compromise.html
7. What is Confidence Scoring in Threat Intelligence? [Електронний ресурс] – 2024 – Режим доступу: <https://www.cyware.com/resources/security-guides/cyber-threat-intelligence/what-is-confidence-scoring-in-threat-intelligence>
8. End of Life of an Indicator of Compromise (IoC) [Електронний ресурс] – 2023 – Режим доступу: <https://www.dragos.com/blog/end-of-life-of-an-indicator-of-compromise-ioc>
9. Tool Sprawl, False Positives Hold Security Teams Back [Електронний ресурс] – 2023 – Режим доступу:

<https://www.darkreading.com/application-security/tool-sprawl-false-positives-hold-security-teams-back>

10.5 Tips for Reducing False Positive Security Alerts [Электронный ресурс] –

2022 – Режим доступа: <https://www.csoonline.com/article/571649/5-tips-for-reducing-false-positive-security-alerts.html>

11. Emerging Threats Updates Improve Metadata, Including MITRE ATT&CK Tags [Электронный ресурс] – 2024 – Режим доступа:

<https://www.proofpoint.com/us/blog/threat-insight/emerging-threats-updates-improve-metadata-including-mitre-attck-tags>

12. Sysmon – System Monitor [Электронный ресурс] – 2024 – Режим

доступу: <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>

13. Sysmon Ingest Volume [Электронный ресурс] – 2023 – Режим доступа:

<https://community.splunk.com/t5/Getting-Data-In/Sysmon-Ingest-Volume/m-p/303871>

14. Event ID 255 Error – Sysmon Error [Электронный ресурс] – 2022 – Режим

доступу: <https://learn.microsoft.com/en-us/answers/questions/1035606/event-id-255-error-sysmon-error>

15. Best Practices for Optimizing Custom Indicators [Электронный ресурс] –

2023 – Режим доступа:

<https://techcommunity.microsoft.com/blog/microsoftdefenderatpblog/best-practices-for-optimizing-custom-indicators/2670357>

16. Use Windows Event Forwarding to Assist in Intrusion Detection

[Электронный ресурс] – 2024 – Режим доступа:

<https://learn.microsoft.com/en-us/windows/security/operating-system-security/device-management/use-windows-event-forwarding-to-assist-in-intrusion-detection>

17. Work with Threat Indicators in Microsoft Sentinel [Электронный ресурс] –

2024 – Режим доступа: <https://learn.microsoft.com/en-us/azure/sentinel/work-with-threat-indicators>

18. Create a Detection Rule – Elastic Security Documentation [Электронный ресурс] – 2024 – Режим доступа:
<https://www.elastic.co/docs/solutions/security/detect-and-alert/create-detection-rule>
19. Turn On Threat Matching Searches in Splunk Enterprise Security [Электронный ресурс] – 2024 – Режим доступа:
<https://help.splunk.com/en/splunk-enterprise-security-8/administer/8.0/threat-intelligence/turn-on-threat-matching-searches-in-splunk-enterprise-security>
20. Threat Graph – CrowdStrike Platform [Электронный ресурс] – 2024 – Режим доступа: <https://www.crowdstrike.com/en-us/platform/threat-graph/>
21. Investigate Alerts in Microsoft Defender for Endpoint [Электронный ресурс] – 2024 – Режим доступа: <https://learn.microsoft.com/en-us/defender-endpoint/investigate-alerts>
22. Cybersecurity Knowledge Graph and Digital Twin with Neo4j [Электронный ресурс] – 2023 – Режим доступа:
<https://neo4j.com/blog/security/graphs-cybersecurity-knowledge-graph-digital-twin/>

ДОДАТОК А

ioc_collector.py

```

1  import os
2  import yara
3  import hashlib
4  import psutil
5  import winreg
6  import json
7  from datetime import datetime, timezone
8
9  SCAN_DIRS = [
10     os.getenv("TEMP", "C:\\Users\\Public\\Temp"),
11     os.path.expanduser("~\\AppData\\Local\\Temp")
12 ]
13
14 RULES_DIR = "rules"
15 KNOWN_BAD_HASHES_FILE = "bad_hashes.txt"
16 OUTPUT_FILE = "output/iocs_scanner_extended.json"
17
18
19 1 usage
20  def load_yara_rules():
21     rule_files = [f for f in os.listdir(RULES_DIR) if f.endswith(".yar") or f.endswith(".yara")]
22     rule_paths = {f"r{i}": os.path.join(RULES_DIR, f) for i, f in enumerate(rule_files)}
23     if not rule_paths:
24         print("[!] No YARA rule files found in ./rules/")
25         return yara.compile(source="rule dummy { condition: false }")
26     print("[*] Loaded YARA rules:", list(rule_paths.values()))
27     return yara.compile(filepaths=rule_paths)
28
29  def sha256_file(path):
30     try:
31         with open(path, "rb") as f:
32             return hashlib.sha256(f.read()).hexdigest()
33     except Exception:
34         return None
35
36
37 1 usage
38  def load_known_bad_hashes():
39     if not os.path.exists(KNOWN_BAD_HASHES_FILE):
40         return set()
41     with open(KNOWN_BAD_HASHES_FILE, "r") as f:
42         return set(line.strip().lower() for line in f if line.strip())
43
44
45 1 usage
46  def scan_files(rules, bad_hashes):
47     iocs = []
48     seen_files = set()
49     for dir_path in SCAN_DIRS:
50         if not os.path.exists(dir_path):
51             continue
52         for root, _, files in os.walk(dir_path):
53             for filename in files:
54                 file_path = os.path.join(root, filename)
55                 try:
56                     h = sha256_file(file_path)
57                     yara_matches = rules.match(filepath=file_path)

```

```

56         if yara_matches or (h and h.lower() in bad_hashes):
57             if file_path not in seen_files:
58                 seen_files.add(file_path)
59                 iocs.append({
60                     "type": "file",
61                     "file": file_path,
62                     "hash": h,
63                     "timestamp": datetime.now(timezone.utc).isoformat(),
64                     "yara": [m.rule for m in yara_matches] if yara_matches else [],
65                     "hash_match": h in bad_hashes
66                 })
67                 print(f"[!] File IoC: {file_path}")
68             except Exception:
69                 continue
70     return iocs
71
72 1 usage
73 def scan_processes():
74     iocs = []
75     for proc in psutil.process_iter(['pid', 'name', 'exe', 'cmdline']):
76         try:
77             if proc.info['cmdline']:
78                 joined = " ".join(proc.info['cmdline']).lower()
79                 if any(k in joined for k in ['-enc', 'iex', 'curl', 'wget', 'powershell']):
80                     iocs.append({
81                         "type": "process",
82                         "pid": proc.pid,
83                         "name": proc.info['name'],
84                         "cmdline": joined,
85                         "timestamp": datetime.now(timezone.utc).isoformat()
86                     })
87                     print(f"[!] Process IoC: {joined}")
88             except (psutil.AccessDenied, psutil.NoSuchProcess):
89                 continue
90     return iocs
91
92 1 usage
93 def scan_autoruns():
94     iocs = []
95     autorun_keys = [
96         (winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run"),
97         (winreg.HKEY_LOCAL_MACHINE, r"Software\Microsoft\Windows\CurrentVersion\Run")
98     ]
99     for hive, path in autorun_keys:
100         try:
101             with winreg.OpenKey(hive, path) as key:
102                 for i in range(0, winreg.QueryInfoKey(key)[1]):
103                     name, value, _ = winreg.EnumValue(key, i)
104                     if any(s in value.lower() for s in ['powershell', 'cmd', 'vbs', '.ps1']):
105                         iocs.append({
106                             "type": "autorun",
107                             "key": path,
108                             "name": name,
109                             "value": value,
110                             "timestamp": datetime.now(timezone.utc).isoformat()
111                         })
112                         print(f"[!] Autorun IoC: {name} => {value}")
113         except Exception:

```

```
114         continue
115     return iocs
116
117
118 1 usage
118 def save_iocs(iocs):
119     os.makedirs(name="output", exist_ok=True)
120     with open(OUTPUT_FILE, "w", encoding="utf-8") as f:
121         json.dump(iocs, f, indent=2, ensure_ascii=False)
122     print(f"[*] Saved {len(iocs)} IoCs to {OUTPUT_FILE}")
123
124
125 ▶ if __name__ == '__main__':
126     print("[*] Running extended IoC scanner...")
127     yara_rules = load_yara_rules()
128     known_hashes = load_known_bad_hashes()
129     iocs = []
130     iocs += scan_files(yara_rules, known_hashes)
131     iocs += scan_processes()
132     iocs += scan_autoruns()
133     save_iocs(iocs)
```

ДОДАТОК Б

sysmon_events_collector.py

```

1  import os
2  import json
3  import xml.etree.ElementTree as ET
4  import win32evtlog
5
6  OUTPUT_FILE = "output/events_collected.json"
7  LOG_CHANNEL = "Microsoft-Windows-Sysmon/Operational"
8  MAX_EVENTS = 1000
9
10 1 usage
11 def read_sysmon_events(log_channel=LOG_CHANNEL, max_events=MAX_EVENTS):
12     query = win32evtlog.EvtQuery(*args: log_channel,
13                                 win32evtlog.EvtQueryChannelPath | win32evtlog.EvtQueryReverseDirection,
14                                 None)
15     events = []
16     count = 0
17
18     while count < max_events:
19         try:
20             evt_list = win32evtlog.EvtNext(*args: query, 10)
21         except Exception:
22             break
23         if not evt_list:
24             break
25
26         for evt in evt_list:
27             try:
28                 xml = win32evtlog.EvtRender(*args: evt, win32evtlog.EvtRenderEventXml)
29                 parsed = parse_event_xml(xml)
30                 if parsed:
31                     events.append(parsed)
32
33                 count += 1
34             except:
35                 continue
36
37     return events
38
39 1 usage
40 def parse_event_xml(xml_str):
41     ns = {'e': 'http://schemas.microsoft.com/win/2004/08/events/event'}
42     try:
43         root = ET.fromstring(xml_str)
44         system = root.find(path: 'e:System', ns)
45         eventdata = root.find(path: 'e:EventData', ns)
46
47         evt = {
48             "TimeCreated": system.find(path: 'e:TimeCreated', ns).attrib.get('SystemTime'),
49             "Source": system.find(path: 'e:Provider', ns).attrib.get('Name'),
50             "EventID": int(system.find(path: 'e:EventID', ns).text),
51             "RecordNumber": int(system.find(path: 'e:EventRecordID', ns).text),
52             "Message": {}
53         }
54
55         if eventdata is not None:
56             for data in eventdata.findall(path: 'e:Data', ns):
57                 key = data.attrib.get('Name')
58                 if key:
59                     evt["Message"][key] = data.text
60
61         return evt
62     except:
63         return None

```

```
60 def save_events(events, output_path=OUTPUT_FILE):
61     os.makedirs(os.path.dirname(output_path), exist_ok=True)
62     with open(output_path, 'w', encoding='utf-8') as f:
63         json.dump(events, f, indent=2, ensure_ascii=False)
64     print(f"[*] Saved {len(events)} events to {output_path}")
65
66 ▶ if __name__ == '__main__':
67     print("[*] Collecting Sysmon events...")
68     events = read_sysmon_events()
69     save_events(events)
70
```

ДОДАТОК В

sysmon_events_normalizer.py

```

1  import json
2  import os
3
4  INPUT_FILE = "output/events_collected.json"
5  OUTPUT_FILE = "output/events_normalized.json"
6
7  FIELDS_OF_INTEREST = [
8      "Image",
9      "CommandLine",
10     "TargetFilename",
11     "Hashes",
12     "ParentImage",
13     "ParentCommandLine",
14     "User",
15     "DestinationIp",
16     "DestinationPort",
17     "ProcessGuid",
18     "ParentProcessGuid",
19     "ProcessId",
20     "ParentProcessId"
21 ]
22
23 1 usage
24  def normalize_event(evt):
25      raw_msg = evt.get("Message", {})
26      norm = {
27          "timestamp": evt.get("TimeCreated"),
28          "event_id": evt.get("EventID"),
29          "record_id": evt.get("RecordNumber"),
30          "source": evt.get("Source"),
31          "raw_message": raw_msg if isinstance(raw_msg, str) else json.dumps(raw_msg, ensure_ascii=False)
32      }
33
34      if isinstance(raw_msg, dict):
35          for key in FIELDS_OF_INTEREST:
36              if key in raw_msg:
37                  norm[key] = raw_msg[key]
38
39      return norm
40
41 1 usage
42  def normalize_all():
43      with open(INPUT_FILE, 'r', encoding='utf-8') as f:
44          raw_events = json.load(f)
45
46      normalized = [normalize_event(ev) for ev in raw_events]
47
48      os.makedirs(os.path.dirname(OUTPUT_FILE), exist_ok=True)
49      with open(OUTPUT_FILE, 'w', encoding='utf-8') as f:
50          json.dump(normalized, f, indent=2, ensure_ascii=False)
51
52      print(f"[*] Normalized {len(normalized)} events → {OUTPUT_FILE}")
53
54  if __name__ == '__main__':
55      normalize_all()
56

```

ДОДАТОК Г

import_to_neo4j.py

```

1  import json
2  from neo4j import GraphDatabase
3  import os
4
5  IOC_FILE = "output/iocs_scanner_extended.json"
6  EVENT_FILE = "output/events_normalized.json"
7  NEO4J_URI = "bolt://localhost:7687"
8  NEO4J_USER = "neo4j"
9
10 with open("neo4j_pwd", "r") as f:
11     NEO4J_PASSWORD = f.read().strip()
12
13 driver = GraphDatabase.driver(NEO4J_URI, auth=(NEO4J_USER, NEO4J_PASSWORD))
14
15 2 usages
16 def load_json(path):
17     with open(path, "r", encoding="utf-8") as f:
18         return json.load(f)
19
20 4 usages
21 def normalize_path(p):
22     return os.path.normpath(p).lower() if p else ""
23
24 1 usage
25 def create_graph(tx, iocs, events):
26     for ioc in iocs:
27         file_path = ioc.get("file")
28         if not file_path:
29             continue
30         yara = ioc.get("yara", [])
31         timestamp = ioc.get("timestamp")

```

```

29         tx.run("MERGE (i:IoC {file: $file}) SET i.yara = $yara, i.timestamp = $ts",
30                file=file_path.lower(), yara=yara, ts=timestamp)
31
32     for event in events:
33         eid = event.get("event_id")
34         pguid = event.get("ProcessGuid")
35         pid = event.get("ProcessId")
36         img = event.get("Image", "")
37         cmd = event.get("CommandLine", "")
38         file = event.get("TargetFilename", "")
39         parent_guid = event.get("ParentProcessGuid")
40         timestamp = event.get("timestamp")
41
42         img = normalize_path(img)
43         cmd = normalize_path(cmd)
44         file = normalize_path(file)
45
46         if pguid:
47             tx.run("""
48                 MERGE (p:Process {guid: $pguid})
49                 SET p.pid = $pid, p.image = $img, p.cmd = $cmd, p.timestamp = $ts
50                 """, pguid=pguid, pid=pid, img=img, cmd=cmd, ts=timestamp)
51
52         if parent_guid:
53             tx.run("""
54                 MATCH (parent:Process {guid: $parent}), (child:Process {guid: $child})
55                 MERGE (parent)-[:PARENT_OF]->(child)
56                 """, parent=parent_guid, child=pguid)
57
58         if eid == 11 and file:
59             tx.run("""
60                 MERGE (f:File {path: $file})
61                 WITH f
62                 MATCH (p:Process {guid: $pguid})
63                 MERGE (p)-[:CREATED]->(f)
64                 """, file=file, pguid=pguid)
65
66         if eid == 3:
67             ip = event.get("DestinationIp")
68             port = event.get("DestinationPort")
69             if ip and port:
70                 tx.run("""
71                     MERGE (n:Network {ip: $ip, port: $port})
72                     WITH n
73                     MATCH (p:Process {guid: $pguid})
74                     MERGE (p)-[:CONNECTED_TO]->(n)
75                     """, ip=ip, port=port, pguid=pguid)
76
77         for ioc in iocs:
78             ioc_type = ioc.get("type")
79             if ioc_type == "file":
80                 ioc_file = normalize_path(ioc.get("file", ""))
81                 if ioc_file and (ioc_file in img or ioc_file in cmd or ioc_file in file):
82                     tx.run("""
83                         MERGE (i:IoC {file: $ioc_file})
84                         WITH i
85                         MATCH (p:Process {guid: $pguid})
86                         MERGE (i)-[:DETECTED_IN]->(p)
87                         """, ioc_file=ioc_file, pguid=pguid)
88

```

```

89         elif ioc_type == "process":
90             ioc_pid = ioc.get("pid")
91             ioc_name = ioc.get("name", "").lower()
92             ioc_cmd = ioc.get("cmdline", "").lower()
93             if str(pid) == str(ioc_pid) and ioc_name in img:
94                 tx.run("""
95                     MERGE (i:IoC {pid: $pid, name: $name})
96                     WITH i
97                     MATCH (p:Process {guid: $pguid})
98                     MERGE (i)-[:DETECTED_IN]->(p)
99                 """, pid=ioc_pid, name=ioc_name, pguid=pguid)
100
101     1 usage
102     def main():
103         iocs = load_json(IOC_FILE)
104         events = load_json(EVENT_FILE)
105         with driver.session() as session:
106             session.execute_write(create_graph, iocs, events)
107         driver.close()
108
109     if __name__ == "__main__":
110         main()

```