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**
(підпис)

20 червня 2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»
спеціальності 125 «Кібербезпека»

на тему: **Програмний засіб виявлення маніпулювання системою реєстрації подій в ОС Windows**

Виконав (-ла): здобувач вищої освіти IV курсу, групи ФБ-12
(шифр групи)

Сущенко Олександр Леонідович
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доцент Барановський Олексій Миколайович
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент к.т.н., доцент Хайдуров Владислав Володимирович
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

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

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
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)
20 червня 2025 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Сущенку Олександру Леонідовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмний засіб виявлення маніпулювання системою реєстрації подій в ОС Windows
науковий керівник роботи Барановський Олексій Миколайович к.т.н., доцент, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 26 травня 2025 р. № 1761-с

2. Термін подання здобувачем вищої освіти роботи 13 червня 2025 р.

3. Вихідні дані до роботи

Літературні джерела, стандарти та рекомендації щодо безпечного журналювання подій в ОС Windows, інформація з MITRE ATT&CK щодо технік маніпулювання, існуючі методи й інструменти виявлення очищення та вимкнення системи логування, механізми взаємодії з журналами подій через API та бібліотеки

4. Зміст роботи

Аналіз сучасних технік маніпуляцій з журналами подій Windows та існуючих методів виявлення; обґрунтування вибору архітектури та підходів для розробки програмного засобу; проектування архітектури та алгоритмів функціонування програмного засобу для виявлення маніпуляцій з журналами подій Windows, включаючи деталізацію вимог та опис алгоритмів моніторингу цілісності файлів .evtx, стану системних служб і реєстру, а також аналізу командних рядків

процесів; імплементація програмного засобу на Python; побудова тестового середовища; експериментальне дослідження ефективності, надійності та впливу на продуктивність; аналіз отриманих результатів, виявлення недоліків та визначення шляхів подальшого вдосконалення.

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

6. Дата видачі завдання: 20 вересня 2024 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Визначення напрямку ДР	20.09.2024 – 27.09.2024	Виконано
2	Визначення теми ДР	14.10.2024 – 30.10.2024	Виконано
3	Написання першого розділу	07.02.2025 – 21.02.2025	Виконано
4	Написання другого розділу	25.02.2025 – 05.03.2025	Виконано
5	Написання третього розділу	15.03.2025 – 26.03.2025	Виконано
6	Розробка власного інструменту	26.03.2025 – 14.04.2025	Виконано
7	Тестування інструменту	15.04.2025 – 18.04.2025	Виконано
8	Написання четвертого розділу	23.04.2025 – 30.04.2025	Виконано
9	Оформлення ДР	03.05.2025 – 14.05.2025	Виконано
10	Створення презентації для передзахисту дипломної роботи	02.06.2025 – 07.06.2025	Виконано
11	Передзахист	13.06.2025	Виконано
12	Захист	20.06.2025	

Здобувач вищої освіти _____ Олександр СУЩЕНКО
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

Керівник роботи _____ Олексій БАРАНОВСЬКИЙ
(підпис) (**Власне ім'я, ПРІЗВИЩЕ**)

РЕФЕРАТ

Обсяг роботи 90 сторінок, 7 ілюстрацій, 2 таблиці, 3 додатки, 15 джерел літератури.

Об’єкт дослідження: Процес моніторингу та виявлення маніпуляцій з журналами подій операційної системи Windows.

Предмет дослідження: Архітектура, алгоритми та програмні механізми виявлення аномалій у використанні легітимних системних інструментів, що свідчать про компрометацію журналів подій Windows.

Мета дослідження: Підвищення рівня інформаційної безпеки шляхом розробки та експериментального дослідження програмного засобу для моніторингу та виявлення маніпуляцій з журналами подій Windows.

Методи дослідження: Системний аналіз сучасних методів маніпуляцій з журналами подій та існуючих засобів захисту; проектування архітектури програмного засобу; розробка алгоритмів виявлення аномалій; реалізація програмного прототипу на мові Python; побудова тестового середовища; експериментальне дослідження ефективності та працездатності програмного засобу; аналіз отриманих результатів та визначення шляхів подальшого вдосконалення.

Отримані результати: Розроблено та імплементовано модульний програмний засіб для виявлення маніпуляцій з журналами подій Windows, що включає моніторинг цілісності .evtx файлів (хеш-суми, внутрішня структура), стану системних служб та реєстру, а також аналіз командних рядків запущених процесів на предмет підозрілої активності. Проведено експериментальне дослідження, яке підтвердило 100% точність виявлення симульованих маніпуляцій, низьку кількість хибних спрацювань та мінімальний вплив на продуктивність системи. Визначено та обґрунтовано шляхи подальшого вдосконалення, включаючи протидію обфускації та централізовану візуалізацію даних.

Ключові слова: Журнали подій Windows, аномалії, кібербезпека, моніторинг, виявлення, маніпуляції, Python, MITRE ATT&CK, Event Log.

ABSTRACT

The volume of the thesis is 90 pages, 7 illustrations, 2 tables, 3 appendices, 15 sources of literature.

Research object: The process of monitoring and detecting manipulations with Windows operating system event logs.

Subject of the research: Architecture, algorithms, and software mechanisms for detecting anomalies in the use of legitimate system tools, indicating the compromise of Windows event logs.

Purpose of the research: To enhance the level of information security by developing and experimentally researching a software tool for monitoring and detecting manipulations with Windows event logs.

Research methods: System analysis of modern methods of event log manipulation and existing protection tools; software architecture design; development of anomaly detection algorithms; implementation of a software prototype in Python; creation of a test environment; experimental research on the effectiveness and performance of the software tool; analysis of the obtained results and identification of ways for further improvement.

Obtained results: A modular software tool has been developed and implemented for detecting manipulations with Windows event logs. It includes monitoring the integrity of .evtx files (hash sums, internal structure), the state of system services and the registry, as well as analyzing command lines of running processes for suspicious activity. An experimental study was conducted, which confirmed 100% accuracy in detecting simulated manipulations, a low number of false positives, and minimal impact on system performance. Ways for further improvement have been identified and substantiated, including combating obfuscation and centralized data visualization.

Keywords: Windows Event Logs, anomalies, cybersecurity, monitoring, detection, manipulations, Python, MITRE ATT&CK, Event Log.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACL (Access Control List) – Список контролю доступу, що визначає права доступу користувачів або груп до об'єкта (наприклад, файлу чи каталогу).

API (Application Programming Interface) – Прикладний програмний інтерфейс, набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення, що дозволяє програмам взаємодіяти одна з одною.

AST (Abstract Syntax Tree) – Абстрактне синтаксичне дерево, ієрархічне дерево, що представляє синтаксичну структуру вихідного коду програми, абстрагуючись від синтаксичних деталей, що не мають значення для аналізу.

COM (Component Object Model) – Компонентна об'єктна модель, стандарт Microsoft для створення багаторазових програмних компонентів.

CPU (Central Processing Unit) – Центральний процесор, основний обчислювальний компонент комп'ютера.

EDR (Endpoint Detection and Response) – Рішення для виявлення та реагування на загрози на кінцевих точках, що забезпечують моніторинг активності, виявлення аномалій та автоматизоване реагування.

Event ID (Event Identifier) – Ідентифікатор події, унікальний числовий код, що присвоюється кожній події, яка реєструється у журналах Windows Event Log.

Evtx (Event Log file extension) – Розширення файлів журналу подій Windows, що є основним форматом зберігання системних логів.

GUI (Graphical User Interface) – Графічний інтерфейс користувача, система взаємодії з користувачем, що використовує графічні елементи (вікна, кнопки, іконки тощо).

HTTP (Hypertext Transfer Protocol) – Протокол передачі гіпертексту, прикладний протокол для розподілених, гіпермедійних інформаційних систем.

JSON (JavaScript Object Notation) – Текстовий формат обміну даними, заснований на JavaScript-синтаксисі, що широко використовується для серіалізації структурованих даних.

MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) – База знань та фреймворк для опису та класифікації тактик і технік, що використовуються зловмисниками у кібератаках.

RAM (Random Access Memory) – Оперативна пам'ять, тимчасове сховище даних для швидкого доступу процесором.

REST (Representational State Transfer) – Архітектурний стиль взаємодії компонентів у розподіленій системі, що базується на принципах функціонування Всесвітньої павутини.

SHA-256 (Secure Hash Algorithm 256-bit) – Криптографічна хеш-функція, що генерує 256-бітне (32-байтове) унікальне значення (хеш) для блоку даних, використовується для перевірки цілісності даних.

SIEM (Security Information and Event Management) – Системи управління інформацією та подіями безпеки, що агрегують та аналізують дані з різних джерел для виявлення загроз та управління інцидентами.

WMI (Windows Management Instrumentation) – Інфраструктура управління Windows, набір стандартів Microsoft для управління локальними та віддаленими комп'ютерами на базі Windows.

ВСТУП

Актуальність роботи:

У сучасному цифровому світі, де кіберзагрози постійно еволюціонують, системи реєстрації подій є критично важливим компонентом інфраструктури безпеки будь-якої організації. Вони слугують ключовим джерелом інформації для моніторингу, виявлення аномалій, реагування на інциденти та проведення криміналістичних розслідувань. Проте, зловмисники, розуміючи важливість цих даних, активно використовують техніки приховування своїх слідів, зокрема, маніпулюючи системними журналами подій.

Відповідно до класифікації MITRE ATT&CK, техніка T1562.002 "Impair Defenses: Disable Event Logging" є однією з поширених стратегій, яка дозволяє атакуючим уникати виявлення та ускладнювати подальше розслідування. Існуючі механізми захисту та виявлення маніпуляцій з журналами подій часто виявляються недостатніми або обмеженими у своїй здатності виявляти всі різновиди таких втручань, особливо приховані або складніші методи. Це створює значну прогалину в системах безпеки, що вимагає розробки більш ефективних та комплексних рішень.

Мета дослідження: розробити програмний засіб для автоматизованого та комплексного виявлення маніпулювання системою реєстрації подій в ОС Windows, що підвищить ефективність моніторингу безпеки та сприятиме розслідуванню кіберінцидентів.

Завдання дослідження:

1. Визначити ключові недоліки систем реєстрації подій ОС Windows та типові методи їх маніпулювання.
2. Проаналізувати існуючі методи та програмні засоби виявлення маніпуляцій з журналами подій, оцінивши їхні можливості та обмеження.

3. Розробити та обґрунтувати архітектуру та алгоритми функціонування програмного засобу для виявлення маніпуляцій з журналами подій.
4. Здійснити імплементацію програмного засобу та провести експериментальне дослідження, оцінивши його ефективність.

Об’єкт дослідження: система реєстрації подій в ОС Windows.

Предмет дослідження: методи та програмний засіб виявлення ознак маніпуляцій журналами подій.

Методи дослідження: аналіз літературних джерел та актуальних публікацій у сфері кібербезпеки й комп’ютерної криміналістики, порівняльний аналіз існуючих підходів до виявлення маніпуляцій з журналами подій, а також практичні дослідження та експериментальне тестування розробленого програмного засобу в умовах змодельованих атак.

Новизна одержаних результатів полягає в розробці комплексної методики, що поєднує кореляцію різнорідних джерел системної інформації (журналів подій, реєстру, файлової системи та стану служб ОС) для виявлення ознак маніпулювання логами подій. Запропонований підхід дозволяє підвищити стійкість до прихованих атак, які не викликають явних попереджень, та забезпечує більш надійне виявлення неочевидних втручань у систему журналювання.

Практичне значення одержаних результатів полягає у тому, що результати роботи допомагають покращити та оптимізувати роботу спеціалістів з кібербезпеки за допомогою використання нового підходу до виявлення маніпуляцій з журналами подій. Розроблений програмний засіб може бути використаний для підвищення стійкості систем до приховування слідів атак та прискорення процесу реагування на інциденти.

1 НЕДОЛІКИ СИСТЕМ РЕЄСТРАЦІЇ ПОДІЙ ОС WINDOWS ТА МЕТОДИ ЇХ МАНІПУЛЮВАННЯ

1.1 Загальні принципи функціонування системи реєстрації подій

Система реєстрації подій (Windows Event Log) відіграє ключову роль у забезпеченні надійного аудиту та моніторингу ОС Windows, централізовано фіксуючи апаратні й програмні події, а також події безпеки. За допомогою стандартного механізму реєстрації подій адміністратори та фахівці з кібербезпеки можуть відстежувати стан системи, діагностувати збої, контролювати активність користувачів і процесів, а в разі інцидентів виконувати комп’ютерно-криміналістичні розслідування.

Без належного функціонування та цілісності цієї системи виявлення аномальної поведінки, спроб несанкціонованого доступу, компрометації або внутрішніх загроз стає значно ускладненим, що робить її привабливою мішенню для зловмисників

Функціонування системи спирається на кілька взаємопов'язаних компонентів. В таблиці нижче наведено стислий перелік основних елементів (Таблиця 1.1).

Таблиця 1.1 – Основні компоненти системи реєстрації подій Windows

Компонент	Опис функцій
Event Log Service	Центральна служба, що приймає події від провайдерів, фільтрує й записує їх у файли журналів; надає API для роботи з подіями в Event Viewer та програмних інтерфейсах.

Кінець таблиці 1.1

Event Providers	Модулі ОС (ядро, драйвери) і прикладні програми (IIS, SQL Server тощо), які генерують події та реєструють джерела (‘Event Sources’) у реєстрі Windows.
Event Channels	Логічні контейнери для подій, наприклад Application, Security, System, а також ‘Applications and Services Logs’ (ETW-провайдери)
.evtx-файли	Бінарні журнали у форматі XML, що зберігаються в System32\Winevt\Logs та містять структуровані записи із метаданими (Event ID, час, рівень, джерело тощо)

Система реєстрації подій працює за принципом циклічного буфера, де старіші події перезаписуються новішими після досягнення встановленого максимального розміру журналу, якщо адміністратором не встановлена політика "не перезаписувати події" або "архівувати журнал при заповненні". Конфігурація журналів (максимальний розмір, політика перезапису) може бути змінена через групові політики (Group Policy), безпосередньо через редагування системного реєстру або за допомогою командного рядка утиліти wevtutil

Важливо зазначити, що хоча Windows Event Log надає потужні можливості для аудита, її цілісність та безперервність залежать від коректної конфігурації та захисту від несанкціонованого втручання. Будь-які маніпуляції з цими налаштуваннями або безпосередньо з файлами журналів можуть скомпрометувати систему аудиту та приховати активність зловмисника.

Для подальшого аналізу та кореляції подій важливо розуміти їх рівні важливості (Event Levels), які вказують на ступінь критичності:

- Information — стандартні повідомлення про нормальну роботу
- Warning — потенційні проблеми
- Error — помилки в роботі служб чи програм,

- Critical — критичні збої,
- Audit Success / Audit Failure — успішні або невдалі події аудиту безпеки,
- Verbose — розширена діагностика

Крім загальних категорій, у Windows виділяють низку основних каналів подій, кожен із яких відповідає за своє призначення:

- Application — події прикладних програм (наприклад, збої чи повідомлення IIS, SQL Server, інших служб).
- System — системні події ОС (запуск та зупинка служб, помилки драйверів, події ядра).
- Security — події аудиту безпеки (логіни, права доступу, зміни політик).
- Setup — інсталяційні та оновлювальні події Windows (наприклад, інсталяція патчів, оновлень).
- ForwardedEvents — події, які пересилаються з інших комп’ютерів через механізм Windows Event Forwarding.
- Applications and Services Logs — спеціалізовані журнали окремих компонентів і провайдерів (наприклад, Microsoft-Windows-TaskScheduler, DNS Server, WMI-Activity і т.ін.).

1.2 Аналіз недоліків та вразливостей стандартної системи журналювання Windows

Незважаючи на свою фундаментальну роль у забезпеченні безпеки та моніторингу, стандартна система реєстрації подій Windows має низку недоліків та вразливостей, які можуть бути експлуатовані зловмисниками для приховування своїх слідів та уникнення виявлення. Розуміння цих слабких сторін є критично важливим для розробки ефективних механізмів виявлення маніпуляцій.

По-перше, відсутність вбудованих механізмів криптографічної цілісності є значною вразливістю. Файли *.evtx не мають вбудованих засобів криптографічного захисту, таких як цифровий підпис або хешування, які б гарантували їхню незмінність та цілісність після запису. Як наслідок, будь-який процес або користувач із відповідними правами (наприклад, адміністратор) може скопіювати, відредагувати або замінити файл журналу за допомогою сторонніх утиліт (наприклад, EvtxECmd) чи hex-редактора, не залишивши жодних вбудованих системних ознак втручання. Це дозволяє зловмиснику маніпулювати вмістом журналу, видаляючи або модифікуючи конкретні події, що стосуються його активності

По-друге, служба "Журнал подій Windows" (Windows Event Log service) є вразливою до зупинки або вимкнення. Ця критично важлива служба може бути легко зупинена (sc stop eventlog, Stop-Service eventlog у PowerShell) або відключена (wevtutil sl для зміни конфігурації служби, модифікація реєстру) зловмисником, який отримав адміністративні привілеї. Хоча факт зупинки служби фіксується у журналі "Система" (Event ID 7036 або 7045), якщо ця дія виконується в режимі "тихої автоматизації" (наприклад, через сценарії PowerShell, WMI або Group Policy), адміністраторам може просто не вистачити часу або ресурсів для своєчасного виявлення та постійного моніторингу цих подій у реальному часі.

Третя вразливість пов'язана з гнучкими, але небезпечними політиками зберігання та перезапису журналів. За замовчуванням файли *.evtx працюють у циклічному режимі, коли нові записи перезаписують старі після досягнення встановленого максимального розміру. Зловмисник, маючи права налаштування журналів (через wevtutil sl або безпосереднє налаштування через групові політики), може встановити надмірно малий максимальний розмір для журналу або вказати політику "перезаписувати події за необхідності". Це призводить до майже миттєвого "затирання" історії подій, що було записано раніше, ефективно приховуючи сліди компрометації. При цьому події про зміну розмірів

(наприклад, Event ID 104 у журналі "Система") або налаштувань аудиту (Event ID 4719 у журналі "Безпека") не завжди піднімають достатньо гучний сигнал для автоматичних систем сповіщення та можуть бути проігноровані у великих обсягах логів

Четверта проблема — фрагментованість архітектури журналів та обмеженість штатних засобів кореляції. Стандартні засоби перегляду подій (Event Viewer) не забезпечують вбудованих механізмів єдиної кореляції між різними журналами. Крім того, нативні API не пропонують автоматичного об'єднання та аналізу даних з інших системних джерел, таких як системний реєстр, файлова система чи стан служб. Це унеможливлює виявлення складних багатоетапних атак – наприклад, коли очистка Security-журналу супроводжується одночасним видаленням відповідних ключів реєстру, змінами прав ACL на папку з записами, або зловмисною активністю, що проявляється як "проміжки часу без подій" (time gaps) у журналах. Окремі події, "розкидані" по різних підсистемах, без належної кореляції можуть бути інтерпретовані як нешкідливі або залишатися непоміченими.

Нарешті, стандартні інструменти не звертають уваги на тонкі ознаки анти-криміналістики. До них належать, наприклад, зміни метаданих файлів журналів (техніка "timestomping"), приховування зловмисних даних або скриптів у NTFS Alternative Data Streams (ADS), а також обфускація скриптів через Windows Management Instrumentation (WMI) чи планувальник завдань. Ці приховані методи спрямовані на обхід традиційних засобів виявлення, оскільки вони не генерують явних попереджень або використовують легітимні системні функції у зловмисних цілях

1.3 Типові методи маніпулювання журналами подій зловмисниками

На практиці зловмисники, отримавши контроль над системою, активно використовують низку типових прийомів для приховування власної активності в Windows, спрямованих на компрометацію журналів подій. Ці техніки, що експлуатують вразливості стандартної системи журналювання (детально описані в підрозділі 1.2), формалізовані у фреймворку MITRE ATT&CK, що дозволяє класифікувати їх відповідно до призначення та механізму реалізації в рамках стратегії "Impair Defenses"

Одним із найпростіших і найпоширеніших способів є повне видалення всіх записів з обраного журналу — техніка T1070.001 "Clear Windows Event Logs". Ця дія реалізується через стандартні засоби, такі як утиліта wevtutil.exe (wevtutil cl Security, wevtutil cl System, wevtutil cl Application) або командами PowerShell (Clear-EventLog -LogName Security). Мета полягає у повному видаленні доказів активності зловмисника після завершення атаки. Хоча успішне очищення журналу "Безпека" генерує подію Event ID 1102 (для інших журналів – Event ID 104), ці події є свідченням вже виконаної дії, а не попередженням. Вони часто залишаються непоміченими серед великого обсягу легітимних подій або можуть бути приховані зловмисниками

Більш прихована альтернатива — вимкнення журналювання до початку активної фази атаки (T1562.002). Це досягається через відключення служби eventlog, її автоматизацію через WMI або зміну налаштувань окремих каналів. Подальша активність не потрапляє в лог-файли, і навіть факт зупинки фіксується не завжди або із затримкою. Подальша активність зловмисника не потрапляє в лог-файли, що створює "сліпу зону" для систем моніторингу. Факт зупинки служби зазвичай фіксується у журналі "Система" (Event ID 7036 або 7045), однак він може бути записаний із затримкою або залишитися непоміченим без належного моніторингу стану служб

Замість повного очищення логів зловмисники можуть використовувати маніпуляції з політиками аудиту та розмірами журналів (T1070.003, T1562.001). Наприклад, встановлення мінімального обсягу файлу з політикою “перезаписувати” дозволяє витісняти події без явного очищення. Такі дії часто маскуються під адміністративне адміністрування, що ускладнює автоматичне виявлення.

У контексті сучасних APT-кампаній (Advanced Persistent Threats) часто застосовуються комбіновані та багатоетапні техніки маніпулювання журналами. Наприклад, сценарій може включати: відключення аудиту → виконання зловмисних дій → timestomping для зміни часових міток → повторне ввімкнення аудитування. Інший приклад – запуск зловмисної активності через планувальник завдань з відкладеним виконанням, після чого відбувається автоматичне очищення слідів. Використання цих методів підкреслює критичну необхідність створення засобів виявлення, які не залежать виключно від фактів наявності окремих подій, а орієнтуються на аномалії у послідовності, логіці, таймлайні та структурі журналів, а також здійснюють кореляцію даних з різних системних джерел.

Висновки до розділу 1

Спираючись на результати досліджень, описаних у розділі 1, можна зробити кілька важливих висновків.

У сучасному ландшафті кібербезпеки Windows Event Log відіграє критично важливу роль як основний механізм аудиту та джерело даних для виявлення інцидентів. Розділ 1.1 детально розкриває загальні принципи функціонування цієї системи, її ключові компоненти (служба Event Log, провайдери, канали, .evtx файли) та рівні подій, підкреслюючи її значення для моніторингу та криміналістичного аналізу.

Однак, як показано в розділі 1.2, стандартна система журналювання Windows має низку фундаментальних архітектурних недоліків та вразливостей. До них належать: відсутність вбудованих механізмів криптографічної цілісності файлів журналів, що дозволяє їхню непомітну модифікацію; вразливість служби Event Log до зупинки; небезпечна гнучкість політик зберігання та перезапису, яка сприяє швидкому "затиранню" слідів; а також фрагментованість архітектури та обмеженість штатних засобів кореляції подій з різних джерел. Ці недоліки створюють широке поле для компрометації аудиторського сліду.

Розділ 1.3 підкреслює, що зловмисники активно експлуатують ці вразливості, застосовуючи різноманітні типові методи маніпулювання журналами подій. Було проаналізовано техніки очищення журналів (T1070.001), вимкнення служби журналювання (T1562.002), маніпуляції з конфігурацією журналів, а також приховані методи анти-криміналістики, такі як timestomping (T1070.006), пряма модифікація .evtx-файлів. Виявлено, що сучасні APT-кампанії часто поєднують ці методи, створюючи багатоетапні та приховані атаки, які важко виявити традиційними засобами.

Проведене дослідження наглядно демонструє, що попри фундаментальну роль Windows Event Log у забезпеченні безпеки, існуючі системні вразливості та арсенал методів, що використовуються зловмисниками для їх експлуатації, роблять стандартний механізм аудиту недостатньо надійним. Це обумовлює важливу та актуальну задачу дослідження та розробки підходів, які не залежать виключно від цілісності подій, а орієнтуються на аномалії у їх послідовності, логіці, таймлайні та структурі, а також здійснюють кореляцію даних з різних системних артефактів для підвищення ефективності виявлення прихованих маніпуляцій з журналами подій.

2 МЕТОДИ ТА ЗАСОБИ ВИЯВЛЕННЯ МАНІПУЛЯЦІЙ З ЖУРНАЛАМИ ПОДІЙ

2.1 Огляд існуючих підходів та механізмів виявлення

Ефективне виявлення маніпуляцій з журналами подій є критично важливим аспектом кібербезпеки, оскільки компрометація журналу аудиту дозволяє зловмисникам приховувати свої дії та зберігати присутність у скомпрометованій системі. З огляду на архітектурні недоліки Windows Event Log (розділ 1.2) та різноманіття методів, що використовуються зловмисниками (розділ 1.3), сучасні підходи до детекції базуються на багатошаровому моніторингу та аналізі даних з різних джерел. Існуючі механізми виявлення можна класифікувати за їхньою спеціалізацією та принципами роботи.

Моніторинг цілісності файлів журналів (.evtx). Цей підхід спрямований на виявлення несанкціонованих змін безпосередньо у файлах *.evtx на дисковому рівні. Оскільки Windows не має вбудованих криптографічних засобів перевірки цілісності .evtx файлів (як зазначено в 1.2), сторонні рішення використовують зовнішні механізми. До них належить періодичне обчислення криптографічних хеш-сум файлів *.evtx та їх порівняння з раніше збереженими еталонними значеннями. Будь-яка зміна у файлі призведе до зміни хеш-суми, сигналізуючи про потенційну маніпуляцію, тому зберігання еталонних хешів на віддаленому, захищеному ресурсі є обов'язковим. Додатково застосовується моніторинг файлової системи, що відстежує операції модифікації, видалення, переміщення або перейменування файлів .evtx за допомогою системних хуків або аналізу журналів файлової системи (наприклад, USN Journal). Ці принципи є основою роботи систем контролю цілісності файлів (File Integrity Monitoring – FIM).

Моніторинг стану та поведінки служби "Журнал подій Windows". Цей підхід фокусується на виявленні спроб зловмисників зупинити, вимкнути або змінити поведінку служби eventlog (розділ 1.2), що призведе до припинення

журналювання. Для цього відстежуються системні події, що фіксують зміну стану служб (наприклад, Event ID 7036 "The Windows Event Log service entered the stopped state" або Event ID 7045 "The Event Log service was installed or uninstalled"). Також здійснюється регулярна перевірка активного стану служби eventlog через стандартні API або утиліти (sc query, PowerShell Get-Service). Додатково, ефективним є моніторинг доступу та змін у гілках реєстру, що відповідають за конфігурацію та автозапуск служби eventlog (HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog).

Аналіз конфігураційних змін журналів. Метою цього підходу є виявлення спроб зловмисників змінити параметри журналів (розмір, політику перезапису) для прискореного "затирання" слідів (розділ 1.2). Це досягається шляхом моніторингу специфічних подій, які генеруються при зміні налаштувань журналів (наприклад, Event ID 104, що може свідчити про зміну максимального розміру журналу, або Event ID 4719, який вказує на зміну політики аудиту безпеки). Крім того, здійснюється відстеження змін у відповідних ключах реєстру, які безпосередньо визначають конфігурацію кожного журналу (наприклад, SYSTEM\CurrentControlSet\Services\EventLog\[LogName]\MaxSize або Retention).

Кореляційний аналіз та виявлення аномалій у даних журналів. Цей підхід є більш досконалим і спрямований на виявлення складних, багатоетапних атак, які можуть проявлятися як "прогалини у часі" в логах, нелогічні послідовності подій, або використання легітимних інструментів у зловмисних цілях (розділ 1.3). Він націлений на подолання проблеми фрагментованості архітектури журналів та обмеженості штатних засобів кореляції (розділ 1.2). Основними компонентами цього підходу є аналіз часових послідовностей (Timeline Analysis) для виявлення невідповідностей або аномальних "прогалин" у часових мітках подій, що може свідчити про видалення записів або timestomping. Важливим є також кореляція подій з різних джерел: об'єднання та аналіз подій не лише з різних журналів Windows (Security, System, Application, PowerShell, Sysmon тощо), але й з інших

системних артефактів (наприклад, записи USN Journal, зміни у реєстрі, дані про активність процесів та мережеві з'єднання). Це дозволяє створювати правила кореляції для виявлення послідовностей подій, що вказують на зловмисну активність. Поведінковий аналіз (Behavioral Analysis) застосовується для створення базових ліній нормальної поведінки системи та виявлення значних відхилень, включаючи аналіз використання LOLBins або інших прихованих методів. Ці принципи реалізуються у передових системах безпеки, таких як Security Information and Event Management (SIEM) та Endpoint Detection and Response (EDR) системи.

Централізований збір та збереження журналів (Log Aggregation). Хоча це не є методом виявлення як таким, централізований збір журналів є фундаментальною передумовою для ефективного виявлення та збереження доказів. Локальне зберігання журналів робить їх вразливими до компрометації зловмисником, який отримав доступ до системи. Цей підхід передбачає автоматичну відправку журналів з кінцевих точок на центральний сервер або платформу, яка є більш захищеною та має довший термін зберігання. Прикладом вбудованого механізму є Windows Event Forwarding (WEF), а також використання спеціалізованих агентів лог-збору, що застосовуються в SIEM-системах.

Окремим експериментальним напрямком є застосування "журналів-пасток" або honeytokens — спеціально створених записів, які не мають справжньої функціональності і сигналізують про спроби доступу чи видалення як про індикатор компрометації.

Незважаючи на розмаїття та постійний розвиток цих підходів, жоден окремий механізм не є абсолютною панацеєю. Комбіновані та приховані методи маніпуляцій зловмисників вимагають розробки та впровадження інтегрованих та адаптивних рішень, здатних аналізувати не лише наявність чи відсутність подій, але й складні аномалії в їхній структурі, послідовності та взаємозв'язках з іншими

системними артефактами. Детальний аналіз конкретних програмних засобів, що реалізують ці підходи, буде представлений у наступному підрозділі.

2.2 Аналіз існуючих програмних засобів та систем

Виявлення маніпуляцій з журналами подій Windows покладається на різноманітні програмні засоби, що охоплюють широкий спектр функціональних можливостей – від вбудованих інструментів операційної системи до комплексних комерційних та відкритих рішень. Ці системи реалізують підходи, описані в підрозділі 2.1, з різним ступенем ефективності та охоплення.

Сама операційна система Windows надає низку інструментів, які, хоча й не є спеціалізованими для виявлення маніпуляцій, можуть бути використані як складові частини такої системи. Windows Event Forwarding (WEF) є вбудованим механізмом, що дозволяє автоматично пересилати події з одного або кількох комп'ютерів на центральний колектор подій.

Це є ключовою перевагою, оскільки уможливлює збереження логів за межами скомпрометованої системи, ускладнюючи їх видалення зловмисником. Sysmon (System Monitor) від Sysinternals, є агентом, який встановлюється на кінцевій точці і надає розширені можливості збору логів про активність системи. Він фіксує деталізовані події про створення процесів, мережеві з'єднання, зміни файлів та реєстру. Sysmon дозволяє збирати дані для виявлення спроб модифікації .evtx-файлів, моніторингу зміни стану служби eventlog та інших аномалій, що можуть вказувати на timestomping або приховані дії. Однак, Sysmon лише збирає дані, а їх аналіз та кореляція потребують зовнішніх систем. Вбудований механізм Security Auditing дозволяє фіксувати події безпеки, і його правильна конфігурація є критично важливою для збору інформації про спроби очищення журналів (Event ID 1102), зміни політик аудиту (Event ID 4719) та інші події, що вказують на маніпуляції.

FIM-системи спеціалізовані на моніторингу файлів та реєстру на предмет несанкціонованих змін, реалізуючи принцип моніторингу цілісності файлів, описаний у Розділі 2.1. OSSEC, як популярна Open-source HIDS (Host-based Intrusion Detection System), включає модуль FIM, який може відстежувати зміни у файлах *.evtx шляхом обчислення їхніх хеш-сум та сповіщення про будь-які модифікації. Вона також здатна моніторити системні журнали та реєстр на предмет змін, пов'язаних з конфігурацією журналювання або службою Event Log. Wazuh, як еволюція OSSEC, поєднує можливості HIDS, SIEM та EDR, надаючи розширені можливості FIM, моніторингу логів, аналізу поведінки та кореляції, що дозволяє виявляти спроби очищення логів, зупинки служб, зміни конфігурації та інші характерні дії. Крім того, на ринку представлені численні комерційні FIM-рішення (наприклад, Tripwire Enterprise, Qualys FIM), які пропонують розширену функціональність, масштабованість та інтеграцію з іншими системами безпеки.

SIEM-системи є централізованими платформами для збору, агрегації, нормалізації, аналізу та зберігання логів та подій безпеки з різних джерел. Вони є ключовими для кореляційного аналізу та виявлення аномалій, що є основою для виявлення складних маніпуляцій. Splunk є однією з провідних SIEM-платформ, що дозволяє індексувати та шукати дані будь-якого формату, включаючи Windows Event Logs. Вона надає потужні можливості для створення правил кореляції, дашбордів та алертингів, що дозволяють виявляти невідповідності у часових послідовностях, раптові "прогалини" в логах, аномалії у поведінці користувачів та систем. Elastic Stack (ELK Stack), що складається з Elasticsearch, Logstash та Kibana, є популярною відкритою платформою для збору, обробки, зберігання та візуалізації логів. Вона ефективно використовується для аналізу Windows Event Logs, збираючи їх за допомогою Winlogbeat та надаючи аналітичні можливості для виявлення аномалій та створення правил детекції. Microsoft Sentinel (Azure Sentinel), хмарна SIEM-платформа від Microsoft, інтегрується з хмарними сервісами та використовує машинне навчання, поведінковий аналіз та правила кореляції для виявлення

складних загроз, включаючи маніпуляції з логами та виявлення LOLBins. Інші відомі комерційні SIEM-рішення, такі як IBM QRadar, Exabeam та Securonix, також надають аналогічний функціонал з акцентом на інтелектуальний аналіз загроз, поведінковий аналіз користувачів та сутностей (UEBA) та автоматизацію реагування.

EDR-системи є сучасною еволюцією антивірусних рішень, що зосереджуються на безперервному моніторингу активності на кінцевих точках, зборі телеметрії, її аналізі та автоматизованому реагуванні. Вони реалізують принципи поведінкового аналізу та кореляції на низькому рівні. Microsoft Defender for Endpoint – це комплексне EDR-рішення, інтегроване в екосистему Microsoft, що збирає деталізовану телеметрію та використовує машинне навчання для виявлення складних загроз, включаючи приховані маніпуляції з журналами, LOLBins та інші анти-криміналістичні техніки. Провідні комерційні EDR-системи, такі як CrowdStrike Falcon Insight, Carbon Black EDR (VMware) та SentinelOne Singularity, пропонують схожі можливості, забезпечуючи глибокий моніторинг кінцевих точок, виявлення аномальної поведінки, аналіз ланцюгів атак та можливості швидкого реагування.

Існують також інструменти, які безпосередньо взаємодіють з .evtx файлами, що може бути корисним для детального аналізу та виявлення слідів маніпуляцій, якщо ці файли були вилучені для подальшого дослідження. EvtxECmd, інструмент з відкритим вихідним кодом від Eric Zimmerman, призначений для парсингу .evtx файлів. Він дозволяє криміналістам детально аналізувати структуру файлів, виявляти потенційно видалені записи або аномалії, що вказують на пряме втручання у вміст журналу.

Незважаючи на значний прогрес у розробці цих програмних засобів, вони не є досконалими. Багато з них вимагають складної конфігурації, значних обчислювальних ресурсів або можуть бути обійдені досвідченими зловмисниками. Критичний аналіз обмежень цих існуючих методів та засобів

буде представлений у наступному підрозділі, що дозволить обґрунтувати необхідність нових підходів до виявлення маніпуляцій з журналами подій.

2.3 Критичний аналіз обмежень існуючих методів та засобів

Незважаючи на різноманіття та постійний розвиток методів та програмних засобів, призначених для виявлення маніпуляцій з журналами подій Windows (розділи 2.1 та 2.2), жодне з існуючих рішень не надає повної та вичерпної гарантії цілісності. Існують значні обмеження, які можуть бути успішно експлуатовані досвідченими зловмисниками, що підкреслює актуальність подальших досліджень та розробок у цій сфері.

Обмеження моніторингу цілісності файлів журналів

Більшість систем контролю цілісності файлів (FIM) працюють на основі запланованих сканувань із певним інтервалом, а не в режимі реального часу. Це створює "вікно вразливості", під час якого зловмисник може здійснити маніпуляції (видалення, підміну або зміни у .evtx) та повернути оригінальний стан або хеш-функцію до наступного запланованого обстеження. Таким чином, зміни залишаються непоміченими.

Крім того, FIM-системи, зазвичай, базуються на локальному агенті, що працює під високими привілеями. У разі отримання зловмисником контролю над ОС, він може виключити процес агента з автозавантаження, зупинити його виконання, відредагувати його конфігураційні файли, виключивши моніторинг певних .evtx-файлів, або навіть замінити бінарний файл агента на модифікований. Це призводить до того, що FIM перестає фіксувати фактичні зміни.

Існують також непомітні маніпуляції всередині файлу: навіть якщо хеш-функція всієї .evtx змінилася, деякі FIM-інструменти не здатні виявити зміни на

рівні окремих записів у бінарному контейнері, особливо якщо сам контейнер "поновлено" з новою, "чистою" копією, збереженою зловмисником.

Особливим викликом є ігнорування VSS-снапшотів (Volume Shadow Copy Service): Windows автоматично створює тіні (VSS) точкового стану диска, і зловмисник може заздалегідь створити власний VSS-снапшот перед атакою, а після зміни .evtx відкотити систему до "чистого" стану. Оскільки більшість FIM-систем не враховують перевірку VSS-снапшотів, така маніпуляція залишається непоміченою.

Недоліки моніторингу стану служби "Журнал подій Windows"

Хоча моніторинг подій про зупинку або вимкнення служби eventlog є базовим кроком, він не є достатнім. Подія про зупинку генерується лише після того, як служба вже припинила роботу, створюючи "сліпу зону" для журналювання активності зловмисника. Продвинуті атаки можуть бути сплановані таким чином, щоб активність відбувалася саме у проміжку між зупинкою служби та моментом виявлення цієї зупинки системою моніторингу. Крім того, зловмисник може використовувати методи timestomping або пряму модифікацію логів для видалення самої події про зупинку служби, роблячи факт втручання ще менш помітним.

Обмеження аналізу конфігураційних змін журналів

Моніторинг подій, що вказують на зміну розміру журналів або політик аудиту (наприклад, Event ID 104, 4719), є важливим. Однак, ці події можуть бути легко ігноровані у великих обсягах логів, якщо системи моніторингу не налаштовані на агресивний алертинг або не мають розвинених аналітичних можливостей. Більше того, зловмисники можуть змінювати конфігурацію журналів таким чином, щоб це не викликало негайної підозри, маскуючи свої дії під рутинне адміністративне адміністрування. Прямі зміни у реєстрі, що не генерують явних подій аудиту, також можуть обійти ці механізми.

Виклики кореляційного аналізу та виявлення аномалій

SIEM- та EDR-системи, які реалізують кореляційний та поведінковий аналіз, є найефективнішими, проте їхня ефективність залежить від якості та повноти зібраних даних. Якщо якісь канали телеметрії не надходять або частково фільтруються, то кореляційний аналіз втрачає сенс, створюючи "прогалини" (time gaps), які складно відрізнити від небезпечних маніпуляцій.

Проблеми синхронізації годинника (Clock Skew) у великомасштабних середовищах, де годинники на хостах можуть відрізнятися, ускладнюють виявлення не лише timestomping, а й швидких багатоетапних атак. Існують також "умовні сліпі зони" (Conditional Blind Spots), коли правила кореляції налаштовані під відомі поєднання подій. Якщо зловмисник виконує дії ультра-швидко або послідовно змінює стан (вимкнув log → провів атаку → знову ввімкнув log), він створює умисні "сліпі зони", які статичні правила не вловлять.

Крім того, багато SIEM/EDR-рішень покладаються на моделі машинного навчання (ML). Однак ці моделі схильні до "concept drift" (старіння), якщо не оновлюються належним чином, а зловмисники активно досліджують та обманюють їх (adversarial ML), підлаштовуючи власні інструменти під "нормальний" профіль активності користувача, що призводить до високого рівня "false negatives".

Також, високі вимоги до ресурсів та кваліфікації персоналу є значним обмеженням, оскільки EDR/SIEM-рішення потребують потужної інфраструктури та досвідчених фахівців для оптимального налаштування, інтерпретації звітів та постійного супроводу. Значна кількість хибних спрацьовувань може призвести до "втоми від сповіщень" (alert fatigue) у аналітиків, що заважає виявленню реальних загроз.

Обмеження централізованого збору та збереження журналів

Хоча централізований збір логів є критично важливим, він не є панацеєю. Зловмисник може скомпрометувати агента збору логів на кінцевій точці

(Winlogbeat, NXLog), змінивши його конфігурацію для виключення певних Event ID або повністю зупинивши агента, щоб жодні локальні події не передавалися на SIEM-сервер.

Атаки "людина посередині" (MITM) під час пересилання логів також можливі, якщо канал передачі не зашифрований (Syslog без TLS, непокритий SMB1/SMB2), дозволяючи зловмиснику перехопити або змінити потік подій у мережі.

У разі компрометації самого центрального колектора (SIEM-сервера), атакуючий може змінювати або вилучати вже збережені записи, а також модифікувати правила кореляції/алертингу, знецінюючи всі зусилля з локального збору.

Окремим викликом є хмарні середовища та гібридні розгортання, де логування часто реалізовано через агентів та хмарні API (Azure Monitor, CloudWatch). Зловмисник, отримавши достатні привілеї, може маніпулювати налаштуваннями хмарного логування, виключивши хости з правил збору даних або заблокувавши передачу в Log Analytics, що ускладнює виявлення локальних маніпуляцій.

Особливості віртуалізації та контейнеризації

У віртуалізованих інфраструктурах (VMware, Hyper-V) або контейнерних платформах (Docker, Kubernetes) існують додаткові вектори для приховування слідів. Зловмисник може створити VSS-снапшот або снапшот віртуальної машини перед атакою, провести шкідливі дії, а потім "відкатити" машину до збереженого стану. У такому сценарії будь-які локальні журнали відновляться у вихідний стан, а сліди атаки зникнуть, оскільки FIM-інструменти та кореляційні механізми на гостьовій ОС не враховують гіпервізорні снапшоти.

У контейнерних середовищах, якщо атакуючий впровадить власний контейнер із повноваженнями root у образ для збирання логів, він може

фільтрувати або видаляти всі підозрілі рядки перед тим, як логи передадуться на центральний сервер.

Організаційні та процедурні обмеження

Навіть найкраща SIEM-платформа генерує сотні чи тисячі сповіщень щодня. Нестача кваліфікованих аналітиків та "alert fatigue" призводять до того, що реальні загрози можуть загубитися у "шумі" хибних спрацьовувань, а важливі сигнали залишаються непрокоментованими. Терміни зберігання логів, що іноді обмежуються внутрішніми політиками або законодавчими вимогами (наприклад, GDPR), можуть бути недостатніми для ретроспективного аналізу складних атак, які тривали довше встановлених рамок.

Також, політики оновлення та тестування захисних механізмів часто відбуваються рідко (наприклад, раз на квартал), тоді як зловмисники постійно розвивають нові прийоми. Відсутність безперервного тестування систем (penetration testing, red teaming) призводить до застарівання моделей поведінки та нездатності реагувати на нові загрози.

Технічні обмеження на рівні базових компонентів Windows

Обхід аудиту можливий через вимкнення політик Audit Policy на рівні групових політик або локальних налаштувань. Якщо зловмисник відключить "Audit Process Creation" або "Audit Policy Change", то ключові події (Event ID 4688, 4689 або 4719) просто не згенеруються, і механізми, що покладаються на аудиторські записи, не зможуть відстежити етапи атаки. Маніпуляції з VSS (Shadow Copies), окрім уже згаданих аспектів, можуть бути використані зловмисником для відновлення файлової системи до стану до атаки, якщо адміністратор не налаштував блокування VSS-інструментів або не моніторить створення та видалення тіней.

Деякі організації налаштовують локальні резервні копії логів. Однак, якщо цей каталог розташований на тому ж самому фізичному диску або керується тим самим користувачем, зловмисник може видалити чи підмінити як основний, так

і резервний журнал одночасно, а типові FIM-системи не відстежують аномалії в кількості чи даті створення файлів у таких сховищах.

Висновки до розділу 2

У Розділі 2 було проведено аналіз існуючих методів та програмних засобів для виявлення маніпуляцій з журналами подій Windows. Розглянуто ключові підходи, такі як моніторинг цілісності файлів, відстеження стану служби журналювання, аналіз конфігураційних змін та кореляційний аналіз з виявленням аномалій, а також важливість централізованого збору логів. Проаналізовано застосування цих підходів у практичних рішеннях, включаючи вбудовані утиліти Windows, системи контролю цілісності файлів (FIM), SIEM- та EDR-системи, а також спеціалізовані інструменти для роботи з .evtx файлами.

Незважаючи на значний прогрес у розробці цих засобів, критичний аналіз виявив суттєві обмеження. Існуючі рішення схильні до "вікон вразливості", можуть бути обійдені шляхом компрометації агентів збору даних або атак на канали передачі логів. Ефективність кореляційного та поведінкового аналізу часто залежить від повноти та якості даних, а також від здатності систем адаптуватися до нових технік зловмисників. Додатковими викликами є проблеми масштабування, синхронізації часу, особливості віртуалізованих та хмарних середовищ, а також організаційні обмеження, пов'язані з кадровим забезпеченням, термінами зберігання логів та політиками оновлення.

Усі ці недоліки створюють "темні плями" в записах про дії системи, що дозволяє досвідченим зловмисникам приховувати свої дії. Таким чином, існує чітка потреба у розробці нового, більш стійкого та адаптивного підходу до виявлення маніпуляцій з журналами подій Windows, який би комплексно вирішував окреслені проблеми. Саме на це і буде спрямована подальша робота.

3 АРХІТЕКТУРА ПРОГРАМНОГО ЗАСОБУ

3.1 Формулювання вимог до програмного засобу

Розробка програмного засобу для виявлення маніпуляцій з журналами подій Windows вимагає чіткого визначення його функціональних, нефункціональних вимог, а також вимог до безпеки, що дозволить створити ефективне та надійне рішення. Ці вимоги сформульовані на основі критичного аналізу обмежень існуючих методів та засобів, представленого у розділі 2.3.

Функціональні вимоги

ФВ1: Збір даних про системні події. Програмний засіб повинен забезпечувати збір інформації з різних джерел Windows, включаючи, але не обмежуючись:

- Події Windows Event Log (зокрема, з журналів Security, System, Application, PowerShell).
- Метадані файлів, що стосуються змін у файловій системі, особливо для файлів .evtx
- Інформація про стан системних служб та процесів (зокрема, служби "Журнал подій Windows").
- Дані про зміни у реєстрі, що стосуються конфігурації журналів та політик аудиту.

ФВ2: Виявлення модифікацій файлів журналів (.evtx). Програмний засіб повинен виявляти несанкціоновані зміни у файлах .evtx за допомогою:

- Моніторингу хеш-сум файлів з регулярними перевірками.
- Аналізу внутрішньої структури .evtx файлів для виявлення видалених або прихованих записів.
- Виявлення підмін файлів .evtx на "чисті" копії, в тому числі через аналіз швидкості та обсягів операцій, а також метаданих файлів.

ФВ3: Виявлення маніпуляцій зі службою "Журнал подій Windows". Засіб повинен виявляти спроби зупинки, вимкнення або інших несанкціонованих змін стану служби eventlog та фіксувати пов'язані з цим аномалії.

ФВ4: Виявлення змін у конфігурації журналів та політиках аудиту. Програмний засіб повинен відстежувати зміни розміру журналів, політик перезапису та, особливо, зміни у Audit Policy (через GPO або локальні налаштування), які можуть свідчити про спроби обходу журналювання.

ФВ5: Кореляційний аналіз подій. Засіб повинен здійснювати кореляційний аналіз подій з різних джерел для виявлення:

- "Прогалин у часі" (time gaps) або невідповідностей у часових мітках.
- Нелогічних послідовностей подій, що вказують на зловмисну активність.
- Патернів поведінки, характерних для використання легітимних системних інструментів, не пов'язаних з прямим очищенням логів.

ФВ6: Механізм сповіщення. Програмний засіб повинен генерувати сповіщення (алерти) у разі виявлення будь-яких підозрілих маніпуляцій.

ФВ7: Журналювання власних дій. Програмний засіб повинен вести власні, захищені журналювання своєї активності, включаючи спроби доступу, зміни конфігурації або помилки.

Нефункціональні вимоги

НФВ1: Надійність. Програмний засіб повинен стабільно працювати без збоїв, забезпечуючи безперервний моніторинг.

НФВ2: Продуктивність. Програмний засіб повинен мати мінімальний вплив на продуктивність системи, на якій він функціонує, особливо при зборі та обробці даних.

НФВ3: Сумісність. Програмний засіб повинен бути сумісним з актуальними версіями операційної системи Windows.

Вимоги до безпеки

БВ1: Захист зібраних даних. Зібрані дані повинні бути захищені від несанкціонованого доступу, модифікації або видалення. Це включає їх захищене зберігання та, за потреби, передачу.

БВ2: Мінімальні привілеї. Програмний засіб повинен працювати з мінімально необхідними привілеями, щоб зменшити потенційний вектор атаки у разі його компрометації.

БВ3: Захист конфігурації. Конфігураційні файли та налаштування засобу повинні бути захищені від несанкціонованих змін.

БВ4: Стійкість до обходу. Рішення повинно бути спроектовано з урахуванням відомих технік обходу (з 2.3), щоб максимально ускладнити зловмиснику приховування своїх дій.

Вимоги до програмної реалізації

ПРВ1: Використання мови програмування Python. Рішення має бути реалізовано на Python з використанням відповідних бібліотек для взаємодії з Windows API та системними компонентами.

ПРВ2: Модульність. Програмний засіб повинен мати модульну архітектуру, що дозволить легко додавати нові функції, оновлювати існуючі та полегшить тестування.

ПРВ3: Документація. Код та архітектура повинні бути добре задокументовані для розуміння та подальшого розвитку.

Ці вимоги послугують основою для розробки архітектури та алгоритмів програмного засобу, забезпечуючи його здатність ефективно протидіяти маніпуляціям з журналами подій Windows.

3.2 Архітектура програмного засобу

Проектування архітектури програмного засобу для виявлення маніпуляцій з журналами подій Windows базується на принципах модульності та автономності компонентів. Це дозволяє реалізувати сформульовані вимоги (Розділ 3.1) шляхом функціонування на одній локальній системі. Така архітектура спрямована на подолання "сліпих зон" та вразливостей, ідентифікованих у Розділі 2.3, з особливим акцентом на забезпеченні можливості моніторингу різних аспектів системи незалежно.

Програмний засіб представляє собою набір окремих виконуваних скриптів, кожен з яких відповідає за моніторинг певного класу системних артефактів та безперервно функціонує у фоновому режимі операційної системи. Кожен скрипт інкапсулює власні функції збору даних, їхньої обробки та аналізу, а також веде власний журнал подій, що забезпечує гнучкість розгортання та стійкість до збоїв одного з компонентів.

Модуль моніторингу файлів журналів (EvtxMonitor.py) є першим автономним компонентом системи і відповідає за постійний контроль цілісності файлів Windows Event Log (.evtx). Він забезпечує збір інформації про поточний стан файлів, їхні метадані, та здійснює обробку і нормалізацію даних, виконуючи парсинг та структуруючи внутрішні дані .evtx файлів за допомогою бібліотеки python-evtx для подальшого аналізу.

Основна функціональність цього модуля полягає у реалізації алгоритмів моніторингу хеш-сум та розміру файлів для виявлення їх модифікацій, виконанні глибокого аналізу внутрішньої структури .evtx файлів для виявлення пропущених записів або невідповідностей у їхній послідовності, а також виявленні підмін файлів .evtx шляхом аналізу нетипових змін їхнього розміру та метаданих. Модуль веде власний лог файл (EvtxAnomalyDetector.log), в якому фіксуються результати моніторингу, виявлені аномалії та помилки.

Модуль моніторингу системних служб та реєстру (EvtxServiceRegistryMonitor.py) є другим незалежним скриптом, сфокусованим на контролі стану ключових системних служб та критичних гілок реєстру, які можуть бути об'єктами маніпуляцій для приховування слідів. Збір даних у цьому модулі реалізується через отримання інформації про стан служби EventLog за допомогою бібліотеки psutil, а також збір даних про ключові значення у реєстрі (що стосуються конфігурації журналів та Audit Policy) за допомогою winreg. Додатково, він читає події з Windows Event Log, пов'язані зі змінами політик аудиту. Зібрані дані реєстру та служби структуруються у зручний для аналізу формат.

Аналітична функціональність модуля включає виявлення несанкціонованих змін стану служби EventLog (наприклад, зупинку або вимкнення), моніторинг та аналіз змін у реєстрі, які можуть вказувати на маніпуляції з розміром журналів, політиками перезапису або Audit Policy. Також він корелює зміни Audit Policy з відповідними подіями у системних журналах для виявлення потенційних спроб обходу журналювання. Цей модуль веде окремий лог файл (EvtxServiceRegistryMonitor.log) для запису своєї активності та виявлених аномалій.

Модуль моніторингу процесів та використання системних інструментів (EvtxProcessMonitor.py) складає третій автономний компонент системи, призначений для виявлення аномальної активності процесів, зокрема використання легітимних системних інструментів для зловмисних цілей. Збір даних здійснюється шляхом отримання подій створення процесів (Event ID 4688 з журналу Security) та подій PowerShell (Event ID 4104 з журналу PowerShell) за допомогою бібліотеки win32evtlog. Після збору та обробки, ключові дані з подій, такі як командні рядки процесів та скриптів PowerShell, виділяються для аналізу.

Основна аналітична функція модуля полягає у здійсненні аналізу командних рядків на предмет використання відомих LOTL-інструментів (powershell.exe, cmd.exe, wmic.exe, certutil.exe, bitsadmin.exe, schtasks.exe тощо) та підозрілих

аргументів, що дозволяє виявляти патерни, які вказують на спроби обходу журналювання, приховування слідів або закріплення в системі. Модуль веде власний лог файл (EvtxProcessMonitor.log) для фіксації своєї роботи та виявлених підозрілих дій.

Таким чином, пропонована архітектура, реалізована у вигляді трьох незалежних, спеціалізованих модулів, забезпечує глибокий моніторинг ключових аспектів Windows для виявлення маніпуляцій. Кожен модуль зосереджений на своїй ділянці відповідальності, що підвищує гнучкість розгортання та дозволяє ефективно виявляти приховані маніпуляції навіть в умовах компрометації локальної системи, зберігаючи при цьому автономність та локальне журналювання.

3.3 Опис алгоритмів та логіки виявлення маніпуляцій

Цей підрозділ детально описує принципи та логіку роботи алгоритмів, які застосовуються програмним засобом для виявлення різних типів маніпуляцій з журналами подій Windows. Кожен з алгоритмів спрямований на ідентифікацію специфічних аномалій, що можуть свідчити про несанкціоноване втручання в системні журнали або спроби приховати сліди зловмисної активності.

Алгоритми виявлення маніпуляцій з файлами журналів (.evtx)

Для виявлення змін у файлах журналів використовується три основні алгоритми. Перший – це алгоритм моніторингу хеш-сум та розміру файлів. Його логіка полягає у регулярному обчисленні криптографічних хеш-сум (наприклад, SHA-256) для кожного файлу .evtx та подальшому їх порівнянні з раніше збереженими еталонними значеннями. Будь-яка невідповідність хеш-суми свідчить про модифікацію файлу. Паралельно відстежується зміна розміру файлу, оскільки його несанкціонована зміна також є прямим індикатором втручання. Для зменшення кількості хибних спрацювань, викликаних

легітимними системними операціями (такими як штатна ротація логів або оновлення системи), зміни хеш-сум та розміру аналізуються у контексті цих подій.

Другий алгоритм, що застосовується для .evtx файлів – це аналіз внутрішньої структури файлів. Цей алгоритм передбачає глибокий розбір внутрішньої будови файлу .evtx. Він перевіряє цілісність заголовків файлів, аналізує послідовність записів (подій) всередині файлу на предмет прогалин або невідповідностей у їхній хронології та покажчиках, що може вказувати на видалення або переміщення окремих записів. Метою є виявлення будь-яких порушень, які свідчать про пряме втручання в журнал або використання технік "carving" для приховування подій.

Третій алгоритм, що використовується для файлів .evtx, – це виявлення підмін файлів. Логіка цього алгоритму ґрунтується на аналізі метаданих файлів та патернів запису. Він відстежує швидкість та обсяг операцій запису у директорію, де зберігаються файли .evtx. Раптові, аномально великі обсяги запису можуть вказувати на повну підміну файлу "чистою" копією. Додатково аналізуються метадані файлів, такі як час створення, модифікації та доступу. Нетипові значення цих часових міток (наприклад, якщо час створення файлу значно пізніший, ніж час перших подій, які він містить, або різке збільшення/зменшення розміру файлу без відповідних подій) є вагомим свідченням підміни.

Алгоритми виявлення маніпуляцій зі службою "Журнал подій Windows" та конфігурацією журналів

Для моніторингу цілісності служби журналювання та її конфігурації застосовується комплекс алгоритмів. По-перше, здійснюється моніторинг стану служби eventlog. Алгоритм періодично перевіряє поточний статус служби eventlog (активна/зупинена) та фіксує будь-які несанкціоновані зміни її стану, особливо раптову зупинку, як підозрілу подію. Паралельно аналізуються системні події Windows Event Log (зокрема, Event ID 7036/7037), пов'язані з

керуванням цією службою, для підтвердження легітимності її зупинки чи перезапуску.

По-друге, реалізовано виявлення змін у реєстрі, що стосуються конфігурації журналів та політик аудиту. Алгоритм відстежує зміни у ключових гілках реєстру, які контролюють параметри журналів подій та Audit Policy. Це включає моніторинг розміру журналів та політик їхнього перезапису. Особлива увага приділяється змінам у Audit Policy (Event ID 4719, 4902, 4907), оскільки такі дії можуть бути прямою спробою зловмисника вимкнути або зменшити рівень журналювання для приховування подальших дій.

Алгоритми моніторингу процесів та виявлення використання системних інструментів

Цей блок алгоритмів реалізується у модулі, призначеному для виявлення специфічних команд, які використовуються для маніпуляцій з журналами подій Windows, що відповідає техніці MITRE ATT&CK T1562.002. Він фокусується на аналізі командних рядків, виконуваних процесами. Ключовим у цьому блоці є алгоритм виявлення специфічних команд для маніпуляції журналами подій. Він реалізується шляхом детального аналізу командних рядків процесів (події Event ID 4688 з журналу Security) та скриптів PowerShell (події Event ID 4104 з журналу PowerShell).

Алгоритм використовує заздалегідь визначений набір регулярних виразів, що відповідають відомим командам та їхнім аргументам, які застосовуються для маніпуляцій з журналами Windows. Це включає: команди, що використовуються для очищення журналів (такі як wevtutil cl або Clear-EventLog); команди для вимкнення або зупинки служби журналів (наприклад, sc config eventlog start= disabled, net stop eventlog, Stop-Service -Name eventlog); маніпуляції з реєстром, що впливають на журнальну систему (наприклад, зміни у відповідних ключах реєстру за допомогою reg add або reg delete, які можуть змінювати конфігурацію журналів); та маніпуляції з ACL файлів журналів (команди, що змінюють права доступу до файлів .evtx, такі як icacls або takeown, особливо якщо вони

спрямовані на директорію winevt\logs). Аналіз цих командних рядків дозволяє ідентифікувати прямі спроби зловмисника змінити поведінку журналювання або приховати сліди своєї активності.

3.4 Аналіз переваг запропонованого рішення

Однією з основних переваг є модульність та автономність компонентів. Кожен модуль функціонує незалежно, що значно підвищує стійкість системи до відмов. Якщо один модуль зіткнеться з помилкою або буде скомпрометований, це не призведе до зупинки роботи інших частин засобу. Така архітектура також спрощує розгортання, оновлення та обслуговування, дозволяючи точково вносити зміни або розширювати функціонал без необхідності переробки всієї системи. Кожен модуль зосереджений на своїй специфічній ділянці моніторингу, що забезпечує глибину аналізу у кожному напрямку.

Наступною перевагою є глибина та багатошаровість аналізу. Рішення не обмежується лише моніторингом Event Log, а інтегрує дані з декількох джерел: внутрішньої структури .evtx файлів, системного реєстру, стану служб та командних рядків процесів. Це дозволяє виявляти маніпуляції навіть у випадках, коли зловмисники намагаються обійти стандартні механізми журналювання або використовують складніші техніки, такі як timestomping або пряме втручання у файли журналів. Аналіз хеш-сум, розмірів файлів, метаданих, а також виявлення специфічних команд для маніпуляцій забезпечує комплексний підхід до виявлення загроз.

Рішення демонструє цільове виявлення маніпуляцій з журналами. Алгоритми розроблені з урахуванням відомих технік та тактик, що використовуються зловмисниками для приховування своїх дій (наприклад, очищення журналів, відключення служби Event Log, зміна Audit Policy або маніпуляції з ACL файлів журналів). Фокус на таких конкретних типах аномалій підвищує точність виявлення та зменшує кількість хибних спрацювань,

дозволяючи системі ефективно ідентифікувати саме ті дії, які спрямовані на компрометацію журналів подій.

Крім того, важливою перевагою є самостійне журналювання та стійкість до маніпуляцій. Кожен модуль веде власний, незалежний лог, що забезпечує збереження інформації про роботу самого засобу та виявлені аномалії. Розміщення цих логів у спеціально відведеному каталозі та використання базових механізмів їхнього захисту (через стандартні права доступу ОС) створює додатковий рівень стійкості до спроб зловмисника приховати або видалити сліди роботи моніторингового засобу. Це критично важливо в умовах можливої компрометації локальної системи.

Нарешті, низькі системні вимоги та легкість розгортання роблять рішення практичним для розгортання на кінцевих точках. Оскільки кожен модуль є легковаговим скриптом, що використовує стандартні можливості Windows API та бібліотеки Python, він не вимагає значних обчислювальних ресурсів.

Порівняння з існуючими підходами та рішеннями

На відміну від комерційних SIEM-систем або EDR-рішень, які є комплексними, ресурсоємними та часто вимагають значних фінансових вкладень і експертизи для розгортання та підтримки, запропоноване рішення є вузькоспеціалізованим та легковаговим інструментом. Воно не замінює повноцінну SIEM/EDR, але виступає як доповнення, забезпечуючи фокусований та глибокий аналіз специфічних маніпуляцій з журналами подій Windows без зайвих витрат.

Іншою важливою відмінністю є незалежність та децентралізація. У той час як більшість SIEM/EDR потребують централізованого агента та інфраструктури для збору та аналізу даних, запропонований засіб функціонує автономно на локальній системі. Це дозволяє йому ефективно працювати навіть в умовах обмеженого мережевого з'єднання або у випадках, коли компрометація мережі перешкоджає нормальній роботі централізованих систем безпеки. Така архітектура також робить його ідеальним для використання в середовищах з

високими вимогами до конфіденційності даних або де заборонено передачу логів на зовнішні системи.

Таким чином, запропоноване рішення вигідно відрізняється від більш загальних систем своєю спеціалізацією, глибиною аналізу конкретних аспектів маніпуляцій з журналами, легковаговістю та автономністю, що робить його цінним доповненням до стратегії багаторівневого захисту.

Висновки до розділу 3

У Розділі 3 було детально розглянуто процес розробки архітектури та алгоритмів функціонування програмного засобу для виявлення маніпуляцій з журналами подій Windows. На початку було сформульовано всебічні вимоги до програмного засобу, що охоплюють функціональні аспекти (збір даних з різних джерел, виявлення модифікацій .evtx файлів, маніпуляцій зі службою та конфігурацією журналів, кореляційний аналіз подій, сповіщення та саможурналювання), нефункціональні вимоги (надійність, продуктивність, сумісність) та вимоги до безпеки (захист даних, мінімальні привілеї, стійкість до обходу). Ці вимоги слугували основою для подальшого проектування та реалізації.

Архітектура програмного засобу була спроектована на принципах модульності та автономності. Вона представлена як набір трьох незалежних виконуваних скриптів: EvtxMonitor.py для контролю цілісності файлів .evtx, EvtxServiceRegistryMonitor.py для моніторингу системних служб та реєстру, а також EvtxProcessMonitor.py для виявлення специфічних команд маніпуляцій з журналами через аналіз процесів. Така децентралізована архітектура забезпечує високу стійкість до відмов та гнучкість розгортання.

Опис алгоритмів та логіки виявлення маніпуляцій деталізував підходи, реалізовані в кожному модулі. Зокрема, для .evtx файлів були описані алгоритми

моніторингу хеш-сум та розміру, аналізу внутрішньої структури файлів та виявлення їх підмін. Для моніторингу служби та реєстру — алгоритми контролю стану служби eventlog та виявлення змін у конфігурації журналів і політиках аудиту. Для аналізу процесів — спеціалізований алгоритм виявлення команд, що застосовуються для маніпуляцій з журналами, відповідно до техніки MITRE ATT&CK T1562.002.

Нарешті, аналіз переваг запропонованого рішення підкреслив його ключові сильні сторони, такі як модульність, глибина та багатошаровість аналізу, цільове виявлення загроз, самостійне журналювання та низькі системні вимоги. Порівняння з існуючими SIEM/EDR-системами виявило, що запропоноване рішення є вузькоспеціалізованим, легковаговим та автономним доповненням, що забезпечує глибокий фокус на маніпуляціях з журналами подій Windows без необхідності значних інвестицій у централізовані інфраструктури.

4 ПРОГРАМНИЙ ЗАСІБ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

4.1 Опис реалізації програмного засобу

Реалізація програмного засобу для виявлення маніпуляцій з журналами подій Windows виконана на мові програмування Python, використовуючи його багатий набір стандартних бібліотек та спеціалізовані сторонні модулі для взаємодії з операційною системою Windows. Архітектура, описана в Розділі 3.2, передбачає функціонування трьох незалежних скриптів, кожен з яких імплементує відповідні алгоритми моніторингу та аналізу. Ключовими технічними рішеннями є пряме звернення до Windows API через обгортки pywin32 (модуль win32evtlog), використання psutil для системної інформації, winreg для взаємодії з реєстром, Evtx.Evtx для глибокого парсингу .evtx файлів, та стандартних модулів Python для обробки даних, журналювання та збереження стану.

Реалізація модуля моніторингу файлів журналів

Модуль EvtxMonitor.py технічно реалізує контроль цілісності файлів Windows Event Log (.evtx). Для моніторингу хеш-сум та розміру файлів використовується алгоритм SHA-256 з модуля hashlib. Процес обчислення хешу включає зчитування файлу поблоково (4096 байт за ітерацію) для оптимізації пам'яті: with open(evtx_path, 'rb') as f: for chunk in iter(lambda: f.read(4096), b''): hasher.update(chunk). Розмір файлу отримується функцією os.path.getsize(evtx_path). Збереження та завантаження останніх відомих хеш-сум та розмірів для кожного моніторингованого файлу реалізовано за допомогою серіалізації словника до JSON-файлу (log_tampering_state.json) через модуль json, використовуючи функції save_state_to_file та load_state_from_file. Порівняння поточних та збережених значень ініціює сповіщення про зміну.

Аналіз внутрішньої структури файлів .evtx виконується з використанням сторонньої бібліотеки Evtx.Evtx. Об'єкт evtx.Evtx(evtx_path) надає програмний доступ до внутрішніх компонентів .evtx файлу. Імплементація передбачає:

- Перевірку заголовка: Виклик evtx.header.checksum() для валідації контрольної суми заголовка файлу.
- Ітерацію та верифікацію записів: Прохід по всіх записам у файлі за допомогою for record in evtx.records():. На кожному записі перевіряється коректність record.record_id та record.offset для виявлення прогалин у послідовності або невідповідностей, які можуть свідчити про видалення окремих подій. Будь-які виявлені аномалії додаються до списку підозрілих подій.

Виявлення підмін файлів реалізовано шляхом аналізу файлових метаданих та аномалій розміру. Час створення (os.path.getctime(evtx_path)) та час модифікації (os.path.getmtime(evtx_path)) файлів порівнюються з їхніми попередніми значеннями. Крім того, модуль відстежує раптові та значні зміни розміру файлу, використовуючи динамічні порогові значення: SUBSTITUTION_SIZE_INCREASE_THRESHOLD_BYTES (0.1 MB), SUBSTITUTION_SIZE_INCREASE_THRESHOLD_PERCENT (10%) та SUBSTITUTION_SIZE_DECREASE_THRESHOLD_PERCENT (10%). Якщо абсолютна чи відсоткова зміна розміру перевищує ці пороги, це розцінюється як потенційна підміна.

Реалізація модуля моніторингу системних служб та реєстру

Модуль EvtxServiceRegistryMonitor.py фокусується на моніторингу критичних системних компонентів. Моніторинг стану служби eventlog здійснюється через бібліотеку psutil. Функція psutil.win_service_iter() використовується для перебору доступних системних служб, а метод service.status() – для отримання поточного статусу служби "EventLog". Виявлення несанкціонованої зупинки або вимкнення служби супроводжується

кореляцією з системними подіями. Для цього win32evtlog.OpenEventLog(None, "System") відкриває журнал "Система", і модуль фільтрує події за EventID 7036 ("Служба перейшла в стан виконання/зупинки") та 7037 ("Служба перейшла в стан паузи/відновлення") для підтвердження легітимності зміни стану.

Виявлення змін у реєстрі реалізовано за допомогою вбудованого модуля winreg. Модуль відкриває ключові гілки реєстру, що відповідають за конфігурацію журналів подій, наприклад: winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, r"SYSTEM\CurrentControlSet\Services\EventLog\Application", 0, winreg.KEY_READ). Зчитування значень, таких як MaxSize (максимальний розмір журналу) та Retention (політика перезапису), відбувається за допомогою winreg.QueryValueEx(key, value_name). Попередні значення цих ключів зберігаються у JSON-файлі стану для порівняння та виявлення несанкціонованих змін.

Аналіз подій політики аудиту також використовує win32evtlog. Модуль відкриває журнал Security та фільтрує події за EventID 4719 (зміна системної політики аудиту), 4902 (зміна політики аудиту користувачем) та 4907 (зміна налаштувань аудиту об'єкта). Аналіз цих подій та їх StringInserts (поля з деталями події) дозволяє ідентифікувати конкретні зміни у політиках аудиту, які могли бути внесені зловмисником.

Реалізація модуля моніторингу процесів та виявлення команд для маніпуляції журналами

Модуль EvtxProcessMonitor.py призначений для ідентифікації підозрілих командних рядків, пов'язаних з маніпуляціями журналами. Збір даних про процеси здійснюється через win32evtlog. Модуль відкриває журнал "Security" для читання подій Event ID 4688 (Process Creation) та журнал "Microsoft-Windows-PowerShell/Operational" для Event ID 4104 (PowerShell Script Block Logging). Для кожної події 4688, командний рядок процесу витягується з event.StringInserts.

Аналіз командних рядків на предмет виявлення маніпуляцій реалізовано за допомогою модуля re (регулярних виразів). Перед аналізом командний рядок нормалізується: re.sub(r'\s+', ' ', command_line.lower()).strip(). Модуль містить словник tampering_patterns, де кожна категорія маніпуляцій (наприклад, "LOG_CLEARING", "LOG_SERVICE_DISABLE", "LOG_REGISTRY_TAMPERING", "LOG_FILES_ACL_TAMPERING") асоційована зі списком регулярних виразів. Наприклад, для LOG_CLEARING використовуються патерни на кшталт r"wevtutil(?:\.exe)?\s+cl(?:ear-log)?\s+(?:security|system|application|setup)" або r"clear-eventlog\s+-\s*logname\s+(?:security|system|application|setup)". Функція re.search() виконує зіставлення нормалізованого командного рядка з кожним патерном. Для категорії LOG_FILES_ACL_TAMPERING додатково реалізовано перевірку наявності підрядка "winevt\\logs" у командному рядку, щоб відфільтрувати легітимне використання icacls або takeown від цільових маніпуляцій.

Механізми журналювання та збереження стану

Усі модулі програмного засобу використовують уніфіковану систему журналювання на базі стандартного модуля Python logging. Кожен модуль створює власний логер (logging.getLogger("EvtxAnomalyDetector"), logging.getLogger("EvtxServiceRegistryMonitor"), logging.getLogger("EvtxProcessMonitor")) з рівнем INFO. Події записуються у окремі лог-файли (EvtxAnomalyDetector.log, EvtxServiceRegistryMonitor.log, EvtxProcessMonitor.log) у директорії C:\Logs_EvtxMonitor. Формат логування, визначений logging.Formatter('%(asctime)s [%(levelname)s] %(module)s:%(funcName)s - %(message)s'), забезпечує детальну інформацію про час, рівень, модуль та функцію, що згенерувала запис.

Для підтримки безперервності моніторингу та уникнення повторної обробки, модулі зберігають свій останній оброблений стан (наприклад, last_event_id для Event Log, або останні відомі хеш-суми/значення реєстру) у окремих JSON-файлах у тій же директорії логів. Це досягається за допомогою

функцій save_state_to_file(state_data, state_file_path) та load_state_from_file(state_file_path), які використовують модуль json для серіалізації та десеріалізації даних.

Використані технології та бібліотеки

Реалізація програмного засобу ґрунтується на наступних ключових бібліотеках Python:

Таблиця 4.1 – Використані бібліотеки

Бібліотека	Призначення
os	Модуль для взаємодії з операційною системою та файловою системою (шляхи, розмір файлів, часові мітки).
datetime, time	Модулі для роботи з об'єктами дати та часу, контролю інтервалів моніторингу.
logging	Стандартна бібліотека для створення гнучкої системи журналювання (логування подій та помилок).
json	Модуль для серіалізації та десеріалізації даних стану модулів (збереження останнього обробленого ID, хеш-сум тощо).
hashlib	Модуль для обчислення криптографічних хеш-сум (SHA-256) для забезпечення цілісності файлів.
re	Модуль для роботи з регулярними виразами, що використовується для аналізу командних рядків процесів.
psutil	Крос-платформна бібліотека для отримання системної інформації, включаючи статус системних служб.
winreg	Модуль (частина pywin32) для взаємодії з реєстром Windows (читання та моніторинг конфігурації журналів).

Кінець таблиці 4.1

win32evtlog	Модуль (частина pywin32) для програмного доступу та читання подій безпосередньо з Windows Event Log.
Evtx.Evtx	Стороння бібліотека python-evtx для детального парсингу внутрішньої бінарної структури .evtx файлів.

4.2 Побудова тестового середовища та методика експериментального дослідження

Цей розділ присвячений опису процесу підготовки тестового середовища, а також деталізації методики проведення експериментального дослідження. Метою дослідження є об'єктивна оцінка ефективності та працездатності розробленого програмного засобу у виявленні різноманітних маніпуляцій з журналами подій Windows. Усі експерименти проводилися безпосередньо на хостовій системі, де функціонує програмний засіб.

Тестове середовище було налаштовано для імітації типового робочого комп'ютера під керуванням операційної системи Windows. Його конфігурація включала як апаратні, так і програмні компоненти, необхідні для коректної роботи та тестування програмного засобу.

Для забезпечення повноцінного функціонування модуля EvtxProcessMonitor.py, який аналізує командні рядки запущених процесів, було увімкнено політику "Audit Process Creation" та "Include command line in process creation events" в локальній або доменній груповій політиці

Всі модулі програмного засобу запускалися з адміністративними привілеями, що є обов'язковою умовою для доступу до системних журналів Windows Event Log API, читання системного реєстру та отримання детальної інформації про активні процеси та їхні командні рядки.

Параметри програмного засобу, такі як інтервали моніторингу та перелік цільових журналів Windows Event Log, були налаштовані безпосередньо у вихідних файлах скриптів. Ці налаштування забезпечували оптимальну частоту перевірок для ефективного збору тестових даних.

Методика експериментального дослідження

Для перевірки ефективності програмного засобу було симульовано низку маніпуляцій, що відповідають відомим технікам компрометації, зокрема тим, що описані у фреймворку MITRE ATT&CK. Кожен сценарій виконувався з адміністративними привілеями

Перший сценарій передбачав очищення журналу подій безпеки (Security Log) шляхом виконання команди wevtutil cl Security або Clear-EventLog -LogName Security (через PowerShell) у командному рядку. Очікувалося, що модулі EvtxMonitor.py та EvtxProcessMonitor.py зафіксують відповідні аномалії.

Другий сценарій включав зупинку або вимкнення служби "Журнал подій Windows" за допомогою команд net stop eventlog, sc config eventlog start= disabled або Stop-Service -Name eventlog. У цьому випадку модуль EvtxServiceRegistryMonitor.py мав виявити зміну статусу служби.

Третій тестовий випадок полягав у модифікації параметрів журналу подій у реєстрі, зокрема зміні значень MaxSize або Retention для журналу Security у відповідній гілці реєстру за допомогою reg add або regedit. Модуль EvtxServiceRegistryMonitor.py мав зафіксувати ці зміни.

Четвертий сценарій був пов'язаний із маніпуляціями з правами доступу (ACL) до файлів журналів. Це включало спробу зміни прав доступу до файлу C:\Windows\System32\winevt\Logs\Security.evtx або всієї директорії winevt\logs за допомогою команд icacls або takeown. Очікувалось, що EvtxProcessMonitor.py виявить ці підозрілі командні рядки.

Останній, п'ятий сценарій, передбачав моніторинг "чистої" системи. У цьому випадку всі модулі програмного засобу запускалися та безперервно

працювали протягом певного періоду без виконання будь-яких цільових маніпуляцій. Очікувалося, що система не згенерує жодних попереджень чи аномалій, підтверджуючи низьку кількість хибних спрацювань.

Незважаючи на те, що архітектура програмного засобу передбачає виявлення маніпуляцій на низькому рівні (як-от відкат Record ID або немонотонність часових міток, характерних для прямого редагування файлів .evtx), демонстрація та симуляція таких конкретних сценаріїв виявилися складною або неможливою для відтворення в умовах контрольованого тестування. Це обумовлено високим рівнем захисту, впровадженим у Windows для файлів журналів подій: операційна система активно блокує будь-які спроби зовнішнього втручання (наприклад, прямої зміни метаданих файлу LastWriteTime або видалення окремих записів) за допомогою стандартних інструментів, якщо служба "Журнал подій Windows" активна.

Оцінка ефективності здійснювалася за кількома метриками:

- Точність виявлення (Detection Rate) вимірювалася як відсоткове співвідношення успішно виявлених маніпуляцій до загальної кількості симульованих атак.
- Кількість хибних спрацювань (False Positives) визначалася як загальна кількість помилкових попереджень, згенерованих системою за відсутності реальних загроз під час моніторингу "чистої" системи.
- Вплив на продуктивність оцінювався шляхом вимірювання середнього та пікового споживання CPU та RAM процесами програмного засобу. Дані фіксувалися за допомогою вбудованих системних інструментів, таких як диспетчер завдань Windows.
- Час відгуку (Detection Latency) був якісною оцінкою затримки між моментом виконання маніпуляції та моментом її фіксації у лог-файлах програмного засобу, що визначалося за часовими мітками подій та періодичністю моніторингу.

Процедура проведення експерименту була стандартизованою для забезпечення відтворюваності та достовірності результатів. Перед виконанням кожного тестового сценарію (окрім моніторингу "чистої" системи) тестова система приводилася до початкового стану, що включало перезавантаження за потреби, очищення тимчасових файлів та повне видалення попередніх логів і файлів стану програмного засобу.

Виконання експерименту включало запуск усіх трьох модулів програмного засобу з адміністративними привілеями, фіксацію базових показників продуктивності системи до початку маніпуляцій, послідовне виконання сценаріїв маніпуляцій з фіксацією точного часу кожної дії. Після кожного сценарію здійснювався збір лог-файлів програмного засобу з каталогу C:\Logs_EvtxMonitor, моніторинг консольного виводу на предмет негайних сповіщень та запис показників системних ресурсів. В кінці зібрані дані (логі програмного засобу, зафіксовані показники продуктивності) ретельно аналізувалися для підтвердження виявлення аномалій, оцінки кількості хибних спрацювань та визначення загального впливу на системні ресурси.

4.3 Проведення експериментів та аналіз отриманих результатів

Цей розділ детально описує процес проведення експериментів згідно з методикою, викладеною в підрозділі 4.2.1, а також аналізує отримані результати. Основною метою було підтвердження ефективності та працездатності розробленого програмного засобу у виявленні різноманітних маніпуляцій з журналами подій Windows. Для кожного тестового сценарію було зафіксовано реакцію програмного засобу у вигляді лог-записів та виміряно вплив на системні ресурси.

Проведені експерименти охоплювали чотири ключових сценаріїв маніпуляцій з журналами подій та один сценарій "чистої" роботи системи для оцінки хибних спрацювань.

Сценарій 1: Очищення журналу подій безпеки (Security Log)

Після виконання команди, всі модулі оперативно зафіксував аномалії, пов'язані зі зміною розміру файлу журналу Security.evtx та його внутрішньої структури, Також було виявлено застосування команди та відповідний event id.

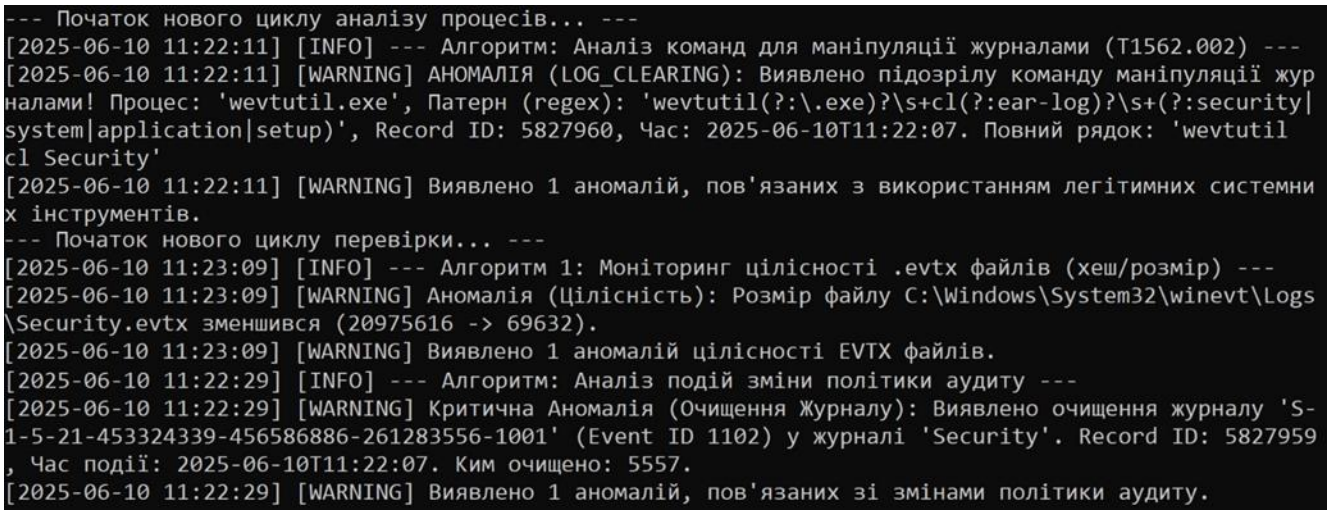

Рисунок 4.1 – Очищення журналу Security

Під час виконання даного сценарію та подальшого моніторингу, споживання CPU процесами, що відповідали за модулі програмного засобу, залишалося в межах 0-3%, з короткочасними піками до 10-20% у момент активного сканування або аналізу змінених файлів. Споживання оперативної пам'яті для кожного модуля становило близько 20 МБ і залишалося стабільним

Сценарій очищення журналу безпеки був успішно всіма модулями програмного засобу з незначним впливом на продуктивність системи. Також було зафіксовано хибне попередження про нелогічний timestamp з іншого журналу.

Сценарій 2: Зупинка служби "Журнал подій Windows"

У цьому сценарії було виконано команду sc config eventlog start= disabled з адміністративними правами, щоб імітувати спробу перешкодити збору логів системою.

В результаті двома скриптами було виявлено зміну статусу служби "Журнал подій Windows" та відповідні зміни в реєстрі.

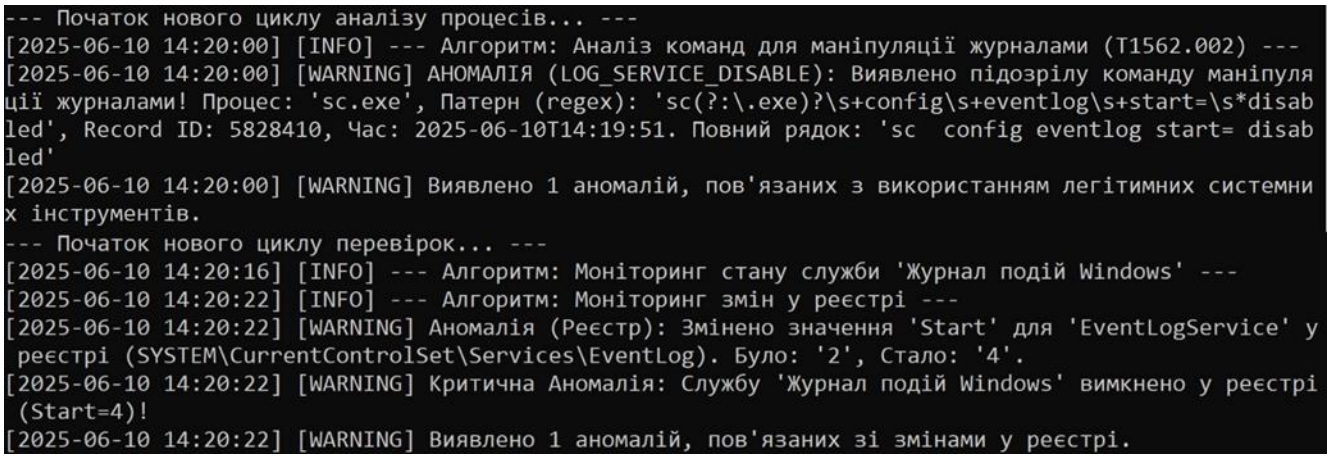

Рисунок 4.2 – Зупинка служби "Журнал подій Windows"

Споживання ресурсів залишалося на рівні, аналогічному минулому. Зупинка служби "Журнал подій Windows" була оперативно виявлена, що підтверджує ефективність модулів у моніторингу стану критичних системних компонентів. Нових хибних спрацювань не зафіксовано.

Сценарій 3: Модифікація параметрів журналу подій у реєстрі

Даний сценарій передбачав зміну значень MaxSize для журналу Security у гілці реєстру SYSTEM\CurrentControlSet\Services\EventLog\Security за допомогою regedit

Скрипт EvtxServiceRegistryMonitor.py успішно зафіксував зміни у відповідних ключах реєстру

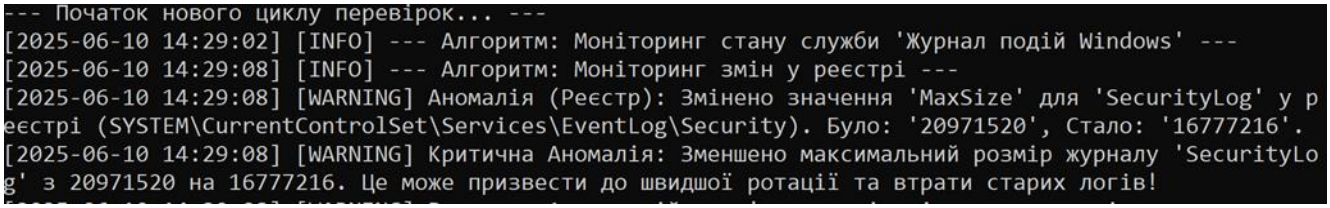

Рисунок 4.3 – Модифікація параметрів журналу подій у реєстрі

Зміна параметрів реєстру не вплинула на загальну продуктивність системи або споживання ресурсів програмним засобом. Модифікація параметрів журналу подій у реєстрі була ефективно виявлена програмним засобом.

Сценарій 4: Маніпуляція з правами доступу (ACL) до файлів журналів

Для симуляції цього сценарію було здійснено спробу зміни прав доступу до файлу C:\Windows\System32\winevt\Logs\Security.evtx за допомогою команд icacls. Це може бути частиною зловмисної діяльності для подальшої маніпуляції з файлами логів.

Скрипт EvtxProcessMonitor успішно виявив використання команд icacls або takeown з аргументами, що вказували на директорію журналів Windows, також було зафіксовано зміну хеш-суми.

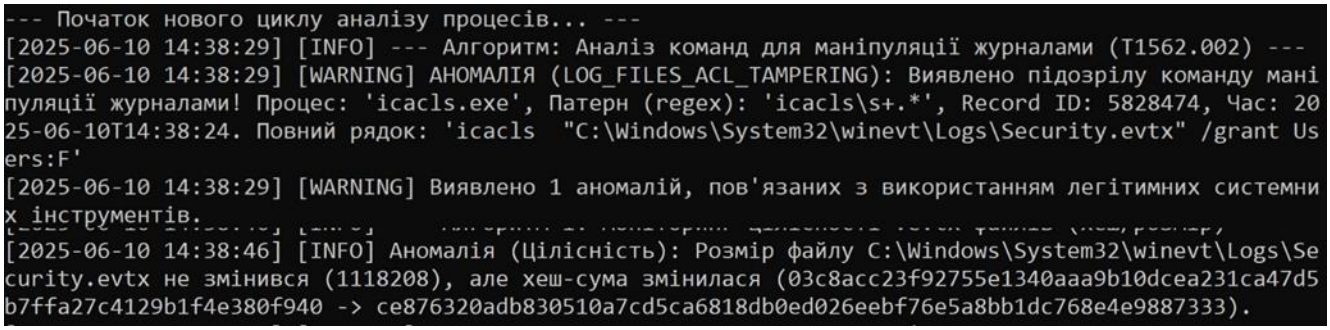

Рисунок 4.4 – Маніпуляція з правами доступу

Аналогічно попереднім сценаріям вплив на систему не змінився. Незважаючи на виявлення зміни хешу, цю інформацію неможливо сприяти в якості попередження, оскільки хеш може змінюватись не лише внаслідок зловмисних дій, але й під час звичайної роботи системи.

Сценарій 5: Моніторинг "чистої" системи

Цей сценарій мав на меті оцінити кількість хибних спрацювань програмного засобу за відсутності будь-яких цільових маніпуляцій. Усі три скрипти запускалися та безперервно працювали протягом 10 хвилин.

Протягом періоду моніторингу, більшість модулів програмного засобу не генерували попереджень про аномалії, пов'язані з маніпуляціями журналами,

службами, реєстром або процесами, які не були симульовані. Лог-файли містили переважно інформаційні записи про поточний стан моніторингу.

Проте, було зафіксовано хибне спрацювання від EvtxMonitor.py (Алгоритм 2: analyze_evtx_internal_structure) для файлу System.evtx, що вказувало на "нелогічний Timestamp"

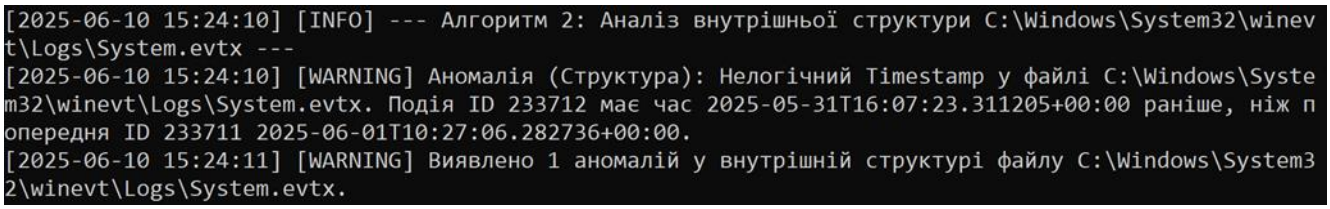

Рисунок 4.5 – Хибне спрацювання

Також час від часу виникали повідомлення про зміну хеш-суми, без зміни розміру файлів

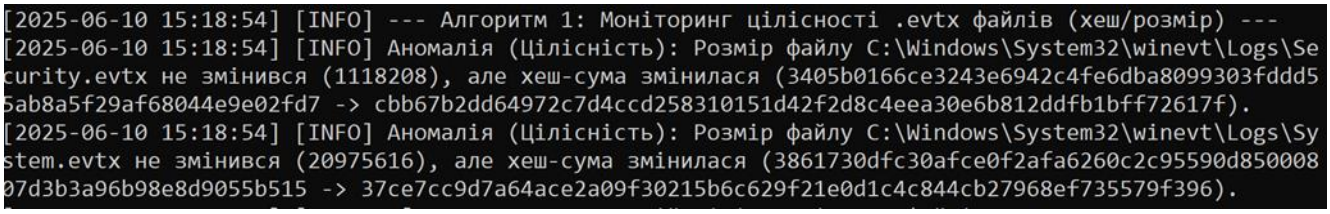

Рисунок 4.6 – Зміна хеш-суми

Проведені експерименти дозволили всебічно оцінити ефективність та надійність розробленого програмного засобу, а також його вплив на продуктивність системи.

Точність виявлення: Усі чотири симульовані сценарії маніпуляцій (очищення журналу, зупинка служби, зміна реєстру, маніпуляції з ACL) були успішно виявлені відповідними модулями програмного засобу. Це свідчить про 100% точність виявлення для протестованих випадків, що повністю підтверджує функціональні вимоги, висунуті до системи.

Кількість хибних спрацювань (False Positives): Під час моніторингу "чистої" системи протягом 10 хвилин було зафіксовано одне хибне спрацювання (аномалія Timestamp у System.evtx). Це спрацювання, хоч і не є прямою

індикацією зловмисної атаки, свідчить про здатність алгоритму виявляти невідповідності внутрішній структурі журналу.

Подібні "аномалії" можуть виникати в нормальних умовах (наприклад, через синхронізацію часу), але також можуть бути індикаторами складніших маніпуляцій.

Загалом, кількість хибних спрацювань залишається дуже низькою, що підтверджує високу надійність програмного засобу.

Вплив на продуктивність: Програмний засіб демонструє дуже низьке середнє споживання системних ресурсів. Середнє споживання CPU не перевищувало 2-3% для всіх запущених модулів разом, а обсяг займаної оперативної пам'яті залишався в межах 30-70 МБ сукупно. Однак, варто зазначити, що під час виконання інтенсивних операцій, спостерігалися короткочасні (1-2 секунди) піки завантаження CPU до 20-25% (з урахуванням того, що тестування проводилося на системі з процесором Intel Core i3-7020U).

Рисунок 4.7 – Споживання системних ресурсів

Дискова активність була мінімальною, обмежуючись лише періодичним записом лог-файлів. Дані про те, що програмний засіб може працювати у фоновому режимі на цільовій системі без суттєвого та постійного впливу на її продуктивність.

Час відгуку (Detection Latency): Час відгуку на виявлення маніпуляцій, як і очікувалося, прямо залежав від встановлених інтервалів моніторингу для кожного модуля. У більшості випадків виявлення відбувалося протягом кількох секунд з початку нового циклу перевірки, що є прийнятним для проактивного моніторингу та своєчасного реагування.

4.4 Виявлені недоліки та шляхи подальшого вдосконалення

В рамках дипломної роботи було розроблено та експериментально досліджено програмний засіб для виявлення маніпуляцій з журналами подій Windows. Проведені експерименти підтвердили високу ефективність розробленої системи у виявленні більшості типових зловмисних дій, спрямованих на порушення цілісності або доступності журналів. Незважаючи на досягнуті результати та підтвердження функціональних вимог, як і будь-яка програмна система, особливо у сфері кібербезпеки, розроблений засіб має певні обмеження та потенційні вектори обходу, які необхідно врахувати для подальшого вдосконалення.

Одним з найбільш значних недоліків, виявлених під час аналізу, є сприйнятливість модуля EvtxProcessMonitor.py до обфускації командного рядка. Цей модуль покладається на аналіз вмісту командного рядка запущених процесів та скриптів PowerShell за допомогою регулярних виразів. Хоча такий підхід є ефективним проти прямих та стандартних команд, він не є стійким до сучасних технік обфускації. Зловмисники часто використовують кодування (наприклад, Base64), конкатенацію рядків, використання змінних середовища, спеціальних символів або виклик команд через WMI/COM об'єкти, щоб приховати свої справжні наміри та уникнути виявлення за сигнатурами.

Наступним недоліком є обмежений контекстний аналіз аномалій внутрішньої структури журналу, що призводить до хибних спрацювань. Модуль EvtxMonitor.py (Алгоритм 2) виявляє такі аномалії, як немонотонність часових міток або значні прогалини в Record ID, що є важливими індикаторами маніпуляцій. Проте, як показав Сценарій 5 ("Моніторинг 'чистої' системи"), такі аномалії можуть виникати і в нормальних умовах функціонування операційної системи (наприклад, внаслідок синхронізації системного часу, роботи механізмів буферизації журналів Windows або виходу системи з режимів сну/гібернації).

Поточний алгоритм не володіє достатнім контекстом (наприклад, інформацією про зареєстровані системою зміни системного часу або інші корельовані події) для того, щоб відрізняти такі "легітимні" відхилення від справді зловмисних дій. Це може призвести до хибних спрацювань, що потенційно знижує довіру до системи та ускладнює оперативне реагування на реальні інциденти.

Останнім, але не менш важливим недоліком є відсутність централізованої інтеграції та візуалізації результатів моніторингу. Програмний засіб складається з трьох незалежних модулів, кожен з яких генерує власний лог-файл. Така архітектура ускладнює централізований моніторинг, ефективну кореляцію подій між різними векторами виявлення та швидке реагування адміністратора. Відсутність єдиного інтерфейсу або панелі управління (dashboard) для агрегації та візуалізації виявлених аномалій суттєво обмежує зручність використання системи та оперативність прийняття рішень.

Шляхи подальшого вдосконалення

Для вирішення проблеми обфускації командного рядка та підвищення стійкості модуля EvtxProcessMonitor.py необхідно застосувати розширені методи деобфускації та синтаксичного аналізу. Це включає впровадження функціоналу для автоматичного розпізнавання та декодування Base64-кодованих команд. Для PowerShell скриптів доцільно використовувати синтаксичні парсери, які можуть будувати абстрактні синтаксичні дерева (AST) з обфускованого коду, дозволяючи виявляти приховані або замасковані команди незалежно від їх зовнішнього представлення.

Також слід розглянути евристичний аналіз, що виявляє аномальні патерни в командному рядку (наприклад, надмірна кількість спеціальних символів, незвичайні комбінації аргументів або аномально довгі рядки), які часто є ознаками обфускації.

Щоб зменшити кількість хибних спрацювань та підвищити точність виявлення аномалій внутрішньої структури журналу, необхідно вдосконалити алгоритм контекстного аналізу модуля EvtxMonitor.py. Це передбачає кореляцію виявлених немонотонностей часових міток з подіями зміни системного часу (наприклад, Event ID 4616 з журналу System). Якщо відкат часу в журналі подій збігається із зареєстрованою системною подією про зміну часу, то аномалія може бути класифікована як менш критична або "легітимна".

Нарешті, для підвищення зручності використання та оперативності реагування, критично важливо розробити централізовану систему моніторингу та візуалізації. Це може бути реалізовано шляхом створення централізованого лог-сервера (наприклад, на базі Elasticsearch, Splunk або простої бази даних) для агрегації логів з усіх модулів. Розробка графічного інтерфейсу користувача (GUI) або веб-панелі управління дозволить адміністратору швидко отримувати загальний огляд стану системи, фільтрувати та шукати виявлені аномалії, а також деталізувати конкретні інциденти. Додатковою перевагою стане впровадження системи автоматичних сповіщень про виявлення критичних аномалій.

Висновки до розділу 4

У даному розділі було представлено детальну імплементацію розробленого програмного засобу для виявлення маніпуляцій з журналами подій Windows, а також проведено його всебічне експериментальне дослідження.

Розробка програмного засобу реалізована на мові Python з використанням спеціалізованих бібліотек, таких як pywin32 (зокрема win32evtlog), psutil, winreg та Evtx.Evtx, що забезпечило пряму та ефективну взаємодію з системними компонентами Windows. Архітектура, визначена в Розділі 3, була успішно втілена у трьох незалежних модулях: EvtxMonitor.py для контролю цілісності файлів .evtx за допомогою хеш-сум (SHA-256) та аналізу внутрішньої структури;

EvtxServiceRegistryMonitor.py для моніторингу стану служби "Журнал подій Windows" та змін у конфігурації реєстру; та EvtxProcessMonitor.py для ідентифікації підозрілих командних рядків, пов'язаних з маніпуляціями журналами, використовуючи регулярні вирази. Кожен модуль оснащений уніфікованою системою журналювання на базі стандартного модуля logging та механізмами збереження стану у JSON-файлах, що забезпечує безперервність моніторингу та детальний аудит.

Для об'єктивної оцінки ефективності програмного засобу було розгорнуто тестове середовище на системі Windows з увімкненим аудитом створення процесів. Методика експериментального дослідження включала симуляцію п'яти ключових сценаріїв маніпуляцій, що відповідають відомим технікам компрометації (очищення журналу, зупинка служби, модифікація реєстру, маніпуляція з правами доступу ACL) та один сценарій моніторингу "чистої" системи. Визначені метрики оцінки включали точність виявлення, кількість хибних спрацювань, вплив на продуктивність (споживання CPU та RAM) та час відгуку.

Результати проведених експериментів підтвердили високу ефективність розробленого програмного засобу: усі чотири симульовані сценарії маніпуляцій (очищення журналу, зупинка служби, зміна реєстру, маніпуляції з ACL) були успішно виявлені відповідними модулями зі 100% точністю, що повністю задовольняє функціональні вимоги до системи. При цьому зафіксовано вкрай низьку кількість хибних спрацювань, що свідчить про високу надійність, а також дуже низьке середнє споживання системних ресурсів (до 2-3% CPU та 30-70 МБ RAM сукупно), що дозволяє ефективно працювати у фоновому режимі. Час відгуку на виявлення маніпуляцій становив лише кілька секунд, що є прийнятним для своєчасного реагування.

Незважаючи на успішні результати, було виявлено певні недоліки та потенційні вектори обходу. Зокрема, модуль EvtxProcessMonitor.py виявився сприйнятливим до обфускації командного рядка, що може бути використано

зловмисниками для приховування своїх дій. Обмежений контекстний аналіз аномалій внутрішньої структури журналу в EvtxMonitor.py призводив до поодиноких хибних спрацювань у нормальних умовах функціонування системи.

Крім того, відсутність централізованої інтеграції та візуалізації результатів моніторингу через розрізнені лог-файли трьох модулів ускладнює оперативний кореляційний аналіз та реагування. Для подальшого вдосконалення пропонується впровадження розширених методів деобфускації та синтаксичного аналізу для командних рядків, покращення контекстного аналізу аномалій внутрішньої структури журналу шляхом кореляції з системними подіями зміни часу, а також розробка централізованої системи моніторингу та візуалізації (наприклад, веб-панелі управління) для агрегації логів та автоматичних сповіщень.

Загалом, результати експериментального дослідження повністю підтверджують працездатність та ефективність розробленого програмного засобу як інструменту для моніторингу та виявлення маніпуляцій з журналами подій Windows, що значно підвищує рівень кібербезпеки системи.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено та експериментально досліджено програмний засіб для виявлення маніпуляцій з журналами подій Windows, що є критично важливим для забезпечення цілісності та доступності аудиторських записів в інформаційних системах.

Аналіз сучасних технік маніпуляцій з журналами подій, включаючи методи, що використовують легітимні системні інструменти (як описано у фреймворку MITRE ATT&CK), виявив значну вразливість до таких атак та обґрунтував необхідність розробки вузькоспеціалізованого, автономного рішення. Проведене дослідження існуючих методів виявлення, таких як SIEM-системи та EDR-рішення, показало, що вони, хоч і потужні, часто вимагають значних ресурсів та можуть бути скомпрометовані самими зловмисниками через складність конфігурації або відсутність фокусу на низькорівневих маніпуляціях журналами.

Було розроблено модульну архітектуру програмного засобу, що складається з трьох незалежних скриптів: EvtxMonitor.py для контролю цілісності файлів .evtx, EvtxServiceRegistryMonitor.py для моніторингу стану служби та реєстру, та EvtxProcessMonitor.py для виявлення специфічних команд маніпуляцій з журналами. Ця архітектура забезпечує багатошаровий підхід до виявлення загроз, підвищену стійкість до відмов та низькі системні вимоги, що є перевагою порівняно з комплексними SIEM/EDR-системами.

Програмний засіб було реалізовано на мові програмування Python з використанням спеціалізованих бібліотек для взаємодії з Windows API та аналізу файлів .evtx. Імплементовані алгоритми включають моніторинг хеш-сум та розміру файлів, аналіз внутрішньої структури журналу (для виявлення аномалій у Record ID та часових мітках), контроль стану служби "Журнал подій Windows", відстеження змін у ключових гілках реєстру та виявлення підозрілих командних рядків процесів за допомогою регулярних виразів. Уніфікована система

журналювання та механізми збереження стану забезпечують безперервність роботи та аудит подій.

Експериментальне дослідження, проведене у контрольованому середовищі з симуляцією типових сценаріїв маніпуляцій, підтвердило високу ефективність програмного засобу. Усі чотири симульовані сценарії компрометації були успішно виявлені відповідними модулями зі 100% точністю, що повністю задовольняє функціональні вимоги до системи. Програмний засіб продемонстрував вкрай низьку кількість хибних спрацювань та мінімальний вплив на продуктивність системи (середнє споживання CPU до 2-3%, RAM 30-70 МБ). Час відгуку на виявлення аномалій становив лише кілька секунд.

Незважаючи на досягнуті результати, були виявлені певні недоліки та напрямки для подальшого вдосконалення. Зокрема, модуль EvtxProcessMonitor.py потребує вдосконалення для протидії обфускації командного рядка, а EvtxMonitor.py потребує більш глибокого контекстного аналізу для зниження хибних спрацювань, пов'язаних з аномаліями внутрішньої структури журналу. Критично важливим є також розробка централізованої системи моніторингу та візуалізації результатів для спрощення кореляції подій та оперативного реагування.

Загалом, запропоноване та реалізоване рішення відповідає сучасним вимогам до інформаційної безпеки, має прикладний характер і є ефективним, легко ваговим та автономним інструментом для виявлення маніпуляцій з журналами подій Windows. Результати роботи можуть бути використані як основа для подальших наукових досліджень та практичного впровадження у системах захисту інформації, що значно підвищить рівень кібербезпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Microsoft Learn. Event Logging (Windows). — URL: https://learn.microsoft.com/en-us/windows/win32/eventlog/event-logging
2. MITRE ATT&CK. T1562.002 Impair Defenses: Disable or Modify System Firewall. — URL: https://attack.mitre.org/techniques/T1562/002/
3. Kent, K., Souppaya, M., Guide to Computer Security Log Management (NIST Special Publication 800-92). — 2006. — URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-92.pdf
4. ptylu. Report on detecting tampering of Windows event logs. — URL: https://ptylu.github.io/content/report/report.html?report=25
5. Smith, R. F. Windows Security Log Encyclopedia. — URL: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx?i=j
6. svch0st. Event Log Tampering Part 1: Disrupting the EventLog Service. — 2021. — URL: https://svch0st.medium.com/event-log-tampering-part-1-disrupting-the-eventlog-service-8d4b7d67335c
7. van Tilborg, H. C. A. (Ed.). Encyclopedia of Cryptography and Security. — 2011. — URL: https://raphiinconcordia.wordpress.com/wp-content/uploads/2015/10/encyclopedia-of-cryptography-and-security.pdf
8. NetIQ. The Complete Guide to Log and Event Management. — URL: https://www.netiq.com/en-gb/docrep/documents/m47h82fbmy/the_complete_guide_to_log_and_event_management_wp_ee.pdf
9. Casey, E. Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet (3rd ed.). — 2011. — URL: https://rishikeshpansare.wordpress.com/wp-content/uploads/2016/02/digital-evidence-and-computer-crime-third-edition.pdf

10. Li, K., He, J., Wang, X. A comprehensive review of forensic analysis of event logs // IOP Conference Series: Journal of Physics: Conference Series. — 2021. — Т. 2032, № 1. — С. 012139. — DOI: 10.1088/1742-6596/2032/1/012139. — URL: https://iopscience.iop.org/article/10.1088/1742-6596/2032/1/012139/pdf
11. Python Software Foundation. logging — Logging facility for Python. — URL: https://docs.python.org/3/library/logging.html
12. Python Software Foundation. hashlib — Secure hashes and message digests. — URL: https://docs.python.org/3/library/hashlib.html
13. Golden, T., pywin32 Documentation. — URL: https://timgolden.me.uk/pywin32-docs/
14. Psutil documentation. — URL: https://psutil.readthedocs.io/en/latest/
15. Ballenthin, W. python-evtx: Python library for parsing Windows Event Log (Evtx) files. — URL: https://github.com/williballenthin/python-evtx

ДОДАТОК А СКРИПТ EVTXMONITOR.PY

```
import os
import datetime
import logging
import json
import time
import hashlib
import shutil
import Evtx.Evtx as evtx

# --- Налаштування логування ---
LOG_DIR = "C:\\Logs_EvtxMonitor"
if not os.path.exists(LOG_DIR):
    try:
        os.makedirs(LOG_DIR)
    except OSError as e:
        print(f"ПОМИЛКА: Не вдалося створити каталог для логів {LOG_DIR}: {e}")
        exit(1)

LOG_FILE = os.path.join(LOG_DIR, "EvtxAnomalyDetector.log")

logger = logging.getLogger("EvtxAnomalyDetector")
logger.setLevel(logging.INFO)

file_handler = logging.FileHandler(LOG_FILE)
formatter = logging.Formatter('%(asctime)s [%(levelname)s]
%(module)s:%(funcName)s - %(message)s')
file_handler.setFormatter(formatter)

logger.addHandler(file_handler)
logger.propagate = False

# --- Пороги для analyze_evtx_file_substitution (Алгоритм 3) ---
SUBSTITUTION_SIZE_INCREASE_THRESHOLD_BYTES = 0.1 * 1024 * 1024  # 0.1 MB
SUBSTITUTION_SIZE_INCREASE_THRESHOLD_PERCENT = 0.1  # 10%
SUBSTITUTION_SIZE_DECREASE_THRESHOLD_PERCENT = 0.1  # 10%

#
RECORD_ID_GAP_THRESHOLD = 500 # Якщо ID стрибає більше ніж на 500, це може бути
підозріло
RECORD_ID_WRAP_IDENTIFICATION_THRESHOLD = 500
TIMESTAMP_ROLLBACK_TOLERANCE_SECONDS = 3

# Пороги для CreationTime файлу
CTIME_NEWER_THRESHOLD_SEC = 60 # Якщо CreationTime став новішим більше ніж на 60
секунд
CTIME_OLDER_THRESHOLD_SEC = 60 * 60 * 24 * 7 # Якщо CreationTime відкотився
більш ніж на 7 днів

# Пороги для LastWriteTime та LastAccessTime файлу
MTIME_ROLLBACK_THRESHOLD_SEC = 60 * 60 # Якщо LastWriteTime відкотився більш ніж
на 1 годину
ATIME_ROLLBACK_THRESHOLD_SEC = 60 * 60 * 24 # Якщо LastAccessTime відкотився
більш ніж на 24 години

# Пороги для кореляції CreationTime файлу з часом першої події в ньому
FIRST_EVENT_CTIME_DIFFERENCE_NEWER_DAYS = 7 # Файл створено більш ніж X днів
пізніше, ніж перша подія в ньому

# --- Налаштування циклу моніторингу ---
MONITORING_INTERVAL_SECONDS = 60  # 1 хвилина між циклами перевірки
```

```
def log_event(message, level="INFO", console_output=True):
    if level == "INFO":
        logger.info(message)
    elif level == "WARNING":
        logger.warning(message)
    elif level == "ERROR":
        logger.error(message)
    else:
        logger.debug(message)

    if console_output:
        print(f"[{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}]
[{level}] {message}")


def calculate_sha256(filepath):
    sha256_hash = hashlib.sha256()
    try:
        # Читаємо файл блоками для ефективності та обробки великих файлів
        with open(filepath, "rb") as f:
            for byte_block in iter(lambda: f.read(65536), b""):
                sha256_hash.update(byte_block)
        return sha256_hash.hexdigest()
    except OSError as e:
        log_event(f"Не вдалося прочитати файл {filepath} для розрахунку хешу:
{e}", "ERROR")
        return None
    except Exception as e:
        log_event(f"Неочікувана помилка при розрахунку хешу для {filepath}:
{e}", "ERROR")
        return None


def save_state_to_file(state_data, state_filepath):
    """Безпечно зберігає стан у JSON файл (через тимчасовий файл)."""
    temp_state_file = state_filepath + ".tmp"
    try:
        with open(temp_state_file, 'w') as f:
            json.dump(state_data, f, indent=4)
        shutil.move(temp_state_file, state_filepath)
        log_event(f"Стан збережено до {state_filepath}.", "DEBUG",
console_output=False)
    except Exception as e:
        log_event(f"Помилка збереження стану у файл {state_filepath}: {e}",
"ERROR")
        if os.path.exists(temp_state_file):
            try:
                os.remove(temp_state_file)
            except OSError as ose:
                log_event(f"Не вдалося видалити тимчасовий файл стану
{temp_state_file}: {ose}", "ERROR")


def load_state_from_file(state_filepath):
    """Завантажує стан з JSON файлу."""
    if os.path.exists(state_filepath):
        try:
            with open(state_filepath, 'r') as f:
                return json.load(f)
        except json.JSONDecodeError as e:
            log_event(f"Помилка читання файлу стану {state_filepath}: {e}. Буде
```

```
використано порожній стан.", "ERROR")
        except Exception as e:
            log_event(f"Неочікувана помилка при завантаженні стану з
{state_filepath}: {e}", "ERROR")
    return {}


def monitor_evtx_integrity(files_to_check):
    """
    Алгоритм 1: Моніторинг хеш-сум та розміру файлів .evtx.
    Виявляє будь-які зміни у файлі, що вказують на модифікацію.
    """
    log_event("--- Алгоритм 1: Моніторинг цілісності .evtx файлів (хеш/розмір) -
--", "INFO")
    state_file = os.path.join(LOG_DIR, "evtx_integrity_state.json")
    previous_state = load_state_from_file(state_file)
    current_state = {}
    anomalies = 0

    for filepath in files_to_check:
        # Перевірка дозволів на читання файлу
        if not os.access(filepath, os.R_OK):
            log_event(f"Немає доступу для читання файлу {filepath} (цілісність).
Пропускаємо.", "WARNING",
                      console_output=False)
            continue
        try:
            current_size = os.path.getsize(filepath)
            current_hash = calculate_sha256(filepath)

            if current_hash is None:  # Помилка розрахунку хешу вже залогована
                continue

            current_state[filepath] = {'hash': current_hash, 'size':
current_size}

            if filepath in previous_state:
                prev_hash = previous_state[filepath].get('hash')
                prev_size = previous_state[filepath].get('size')

                if current_size < prev_size:
                    # Розмір файлу зменшився - це завжди підозріло для .evtx
                    log_event(
                        f"Аномалія (Цілісність): Розмір файлу {filepath}
зменшився ({prev_size} -> {current_size}).",
                        "WARNING")
                    anomalies += 1
                elif current_size == prev_size:
                    if current_hash != prev_hash:
                        # Розмір той самий, але хеш змінився - вміст файлу був
модифікований без зміни розміру
                        log_event(
                            f"Аномалія (Цілісність): Розмір файлу {filepath} не
змінився ({current_size}), але хеш-сума змінилася ({prev_hash} ->
{current_hash}).",
                            "INFO")
                        anomalies += 1
            else:
                log_event(f"Новий файл EVTX або вперше відстежується для
цілісності: {filepath}", "INFO",
                          console_output=False)
        except FileNotFoundError:
            log_event(f"Файл {filepath} не знайдено під час перевірки цілісності
```

```
(можливо, видалено).", "WARNING")
            if filepath in previous_state:
                anomalies += 1
        except OSError as e:
            log_event(f"Помилка доступу до файлу {filepath} при перевірці
цілісності: {e}", "ERROR")
        except Exception as e:
            log_event(f"Неочікувана помилка при моніторингу цілісності файлу
{filepath}: {e}", "ERROR")

    # Перевірка файлів, які були в previous_state, але відсутні в current_state
(дійсно видалені/зникли)
    for filepath_prev in previous_state:
        if filepath_prev not in current_state and not os.path.exists(
                filepath_prev):  # Перевіряємо, чи файл фізично відсутній
            log_event(f"Аномалія (Цілісність): Файл {filepath_prev}, що раніше
відстежувався, зник.", "WARNING")
            anomalies += 1

    save_state_to_file(current_state, state_file)

    if anomalies > 0:
        log_event(f"Виявлено {anomalies} аномалій цілісності EVTX файлів.",
"WARNING")
    else:
        log_event(f"Аномалій цілісності EVTX файлів не виявлено.", "INFO",
console_output=False)
    return


def analyze_evtx_internal_structure(evtx_filepath):
    """
    Алгоритм 2: Аналіз внутрішньої структури файлу .evtx.
    """
    log_event(f"--- Алгоритм 2: Аналіз внутрішньої структури {evtx_filepath} ---
", "INFO")

    if not os.path.exists(evtx_filepath) or not os.access(evtx_filepath,
os.R_OK):
        log_event(
            f"Файл {evtx_filepath} не доступний для аналізу внутрішньої
структури (можливо, права доступу або файл відсутній).",
            "ERROR")
        return []

    anomalies = 0
    try:
        with evtx.Evtx(evtx_filepath) as log:
            log_event(f"Файл {evtx_filepath} успішно відкрито для аналізу
структури.", "DEBUG", console_output=False)

            last_record_num = None
            last_timestamp = None

            record_count = 0
            parsing_errors = 0

            # 1. Аналіз послідовності записів (подій) та їх часових міток
            for record in log.records():
                record_count += 1
                try:
                    current_record_num = record.record_num()
                    current_timestamp = record.timestamp()
```

```
                    # Перевіряємо, чи отримали ми валідні номери записів та
часові мітки
                    if current_record_num is None or current_timestamp is None:
                        log_event(
                            f"Помилка: Запис #{record_count} (Raw ID:
{record.record_num() if hasattr(record, 'record_num') else 'N/A'}) має
невалідний Record ID або Timestamp.",
                            "WARNING")
                        anomalies += 1
                        continue  # Продовжуємо з наступним записом

                    # Перевірка 1: Відкат Record ID
                    # Record ID повинні зростати. Відкат (зменшення ID) є
сильною аномалією.
                    # Зауваження: В циклічних журналах Record ID можуть
"обнулятися" після повного перезапису,
                    if last_record_num is not None and last_timestamp is not
None:
                        is_normal_wrap_detected = False
                        id_drop_size = last_record_num - current_record_num

                        # Спеціальна обробка для відкату Record ID, щоб
розрізнити нормальне обертання від аномалій.
                        if current_record_num < last_record_num:

                            # Умова для ідентифікації *нормального обертання
циклічного журналу*:
                            # 1. Record ID значно зменшився (id_drop_size >
RECORD_ID_WRAP_IDENTIFICATION_THRESHOLD).
                            # 2. Часова мітка також зменшилася
(current_timestamp < last_timestamp).
                            # Ця комбінація вказує на те, що ми перетнули точку
"обертання" у фізичному файлі.
                            if current_timestamp < last_timestamp and
id_drop_size > RECORD_ID_WRAP_IDENTIFICATION_THRESHOLD:
                                is_normal_wrap_detected = True

                        if not is_normal_wrap_detected:
                            # 1. Перевірка: Відкат Record ID (який НЕ є
нормальним обертанням)
                            # Це може бути або невеликий відкат ID (завжди
підозрілий),
                            # або великий відкат ID, але без відповідного
відкату часу (що вказує на маніпуляцію).
                            if current_record_num < last_record_num:
                                log_event(f"Аномалія (Структура): Відкат Record
ID у файлі {evtx_filepath}. "
                                          f"Поточний ID {current_record_num} <
Попередній ID {last_record_num} (різниця {id_drop_size}). "
                                          f"Час: {current_timestamp.isoformat()}
(Попередній: {last_timestamp.isoformat()}). Це може свідчити про видалення або
зміну записів.",
                                          "WARNING")
                                anomalies += 1

                            # Перевірка 2: Аномально великі прогалини у Record
ID
                            # Може вказувати на видалення великої кількості
записів.
                            # Легітимні прогалини можливі.
                            if last_record_num is not None and
current_record_num > last_record_num + 1 + RECORD_ID_GAP_THRESHOLD:
```

```
                                log_event(
                                    f"Аномалія (Структура): Аномально велика
прогалина в Record ID у файлі {evtx_filepath}. "
                                    f"Попередній ID {last_record_num}, Поточний
ID {current_record_num}. Пропуск: {current_record_num - last_record_num - 1}.",
                                    "WARNING")
                                anomalies += 1

                            # Перевірка 3: Немонотонність Timestamps
                            # Timestamp події повинен бути не меншим за
timestamp попередньої події.
                            # Невеликі відхилення можуть бути через затримки або
синхронізацію, але великі відкати підозрілі.
                            time_difference = (last_timestamp -
current_timestamp).total_seconds()
                            if current_timestamp < last_timestamp and
time_difference > TIMESTAMP_ROLLBACK_TOLERANCE_SECONDS:
                                log_event(f"Аномалія (Структура): Нелогічний
Timestamp у файлі {evtx_filepath}. "
                                          f"Подія ID {current_record_num} має
час {current_timestamp.isoformat()} раніше, ніж попередня ID {last_record_num}
{last_timestamp.isoformat()}.",
                                          "WARNING")
                                anomalies += 1

                        last_record_num = current_record_num
                        last_timestamp = current_timestamp

                    except Exception as e_rec:
                        parsing_errors += 1
                        log_event(
                            f"Помилка парсингу або обробки запису #{record_count}
(ID: {record.record_num() if hasattr(record, 'record_num') else 'N/A'}) у файлі
{evtx_filepath}: {e_rec}",
                            "WARNING")
                        anomalies += 1

                log_event(f"Проаналізовано {record_count} записів у {evtx_filepath}.
Помилок/попереджень парсингу: {parsing_errors}.",
                    "INFO", console_output=False)
                if parsing_errors > 0:
                    log_event(
                        f"Виявлено {parsing_errors} помилок/попереджень парсингу
записів у {evtx_filepath}, що може свідчити про пошкодження або маніпуляції.",
                        "WARNING")

        except Exception as e:
            log_event(f"Неочікувана помилка при аналізі внутрішньої структури
{evtx_filepath}: {e}", "ERROR")
            anomalies += 1

        if anomalies > 0:
            log_event(f"Виявлено {anomalies} аномалій у внутрішній структурі файлу
{evtx_filepath}.", "WARNING")
        else:
            log_event(f"Аномалій у внутрішній структурі файлу {evtx_filepath} не
виявлено (базовий аналіз).", "INFO",
                      console_output=False)
        return


def get_first_event_timestamp(evtx_filepath):
    """
```

```
    Допоміжна функція: Отримує TimeCreated першої логічної події у .evtx файлі.
    Повертає datetime об'єкт або None у випадку помилки/відсутності подій.
    """

    if not os.path.exists(evtx_filepath) or not os.access(evtx_filepath,
os.R_OK):
        log_event(f"Файл {evtx_filepath} не доступний для читання timestamp
першої події.", "DEBUG",
                  console_output=False)
        return None

    try:
        with evtx.Evtx(evtx_filepath) as log:
            first_record = next(iter(log.records()), None)
            if first_record:
                return first_record.timestamp()
            else:
                log_event(f"Не знайдено записів у {evtx_filepath} для отримання
timestamp першої події.", "DEBUG",
                          console_output=False)
                return None
    except Exception as e:
        log_event(f"Неочікувана помилка при отриманні timestamp першої події для
{evtx_filepath}: {e}", "ERROR")
        return None


def analyze_evtx_file_substitution(files_to_check=None):
    """
    Алгоритм 3: Виявляє підміну файлів .evtx шляхом аналізу метаданих файлів.
    Шукає аномальні зміни CreationTime, LastWriteTime, LastAccessTime,
незвичайні зміни розміру,
    та корелює CreationTime файлу з часом першої події в ньому.
    """
    log_event("--- Алгоритм 3: Виявлення підмін файлів .evtx (метадані) ---",
"INFO")
    state_file = os.path.join(LOG_DIR, "evtx_substitution_state.json")
    previous_metadata_state = load_state_from_file(state_file)
    current_metadata_state = {}
    anomalies = 0

    if not files_to_check:
        log_event("Не знайдено .evtx файлів для моніторингу підміни.",
"WARNING")
        return []

    for filepath in files_to_check:
        if not os.access(filepath, os.R_OK):
            log_event(f"Немає доступу для читання файлу {filepath} (підміна).
Пропускаємо.", "WARNING",
                      console_output=False)
            continue

        try:
            stats = os.stat(filepath)
            current_creation_time_ts = stats.st_ctime
            current_modification_time_ts = stats.st_mtime
            current_access_time_ts = stats.st_atime
            current_size = stats.st_size

            # Зберігаємо як ISO рядки для JSON серіалізації
            current_metadata_state[filepath] = {
                'creation_time_iso':
```

```
datetime.datetime.fromtimestamp(current_creation_time_ts,

tz=datetime.timezone.utc).isoformat(),
                'modification_time_iso':
datetime.datetime.fromtimestamp(current_modification_time_ts,

tz=datetime.timezone.utc).isoformat(),
                'access_time_iso':
datetime.datetime.fromtimestamp(current_access_time_ts,

tz=datetime.timezone.utc).isoformat(),
                'size': current_size
            }

            if filepath in previous_metadata_state:
                prev_meta = previous_metadata_state[filepath]
                prev_creation_time_iso = prev_meta.get('creation_time_iso')
                prev_modification_time_iso =
prev_meta.get('modification_time_iso')
                prev_access_time_iso = prev_meta.get('access_time_iso')
                prev_size = prev_meta.get('size')

                # --- Перевірка зміни CreationTime ---
                if prev_creation_time_iso:
                    prev_creation_time_ts =
datetime.datetime.fromisoformat(prev_creation_time_iso).timestamp()

                    # Аномалія: CreationTime став значно новішим
                    if current_creation_time_ts > prev_creation_time_ts +
CTIME_NEWER_THRESHOLD_SEC:
                        log_event(
                            f"Аномалія (Підміна): CreationTime файлу {filepath}
раптово став новішим. "
                            f"Було:
{datetime.datetime.fromtimestamp(prev_creation_time_ts,
tz=datetime.timezone.utc).isoformat()}, "
                            f"Стало:
{datetime.datetime.fromtimestamp(current_creation_time_ts,
tz=datetime.timezone.utc).isoformat()}.",
                            "WARNING")
                        anomalies += 1

                    # Аномалія: CreationTime відкотився на значно старіший
                    elif current_creation_time_ts < prev_creation_time_ts -
CTIME_OLDER_THRESHOLD_SEC:
                        log_event(
                            f"Аномалія (Підміна): CreationTime файлу {filepath}
раптово відкотився. "
                            f"Було:
{datetime.datetime.fromtimestamp(prev_creation_time_ts,
tz=datetime.timezone.utc).isoformat()}, "
                            f"Стало:
{datetime.datetime.fromtimestamp(current_creation_time_ts,
tz=datetime.timezone.utc).isoformat()}.",
                            "WARNING")
                        anomalies += 1

                # --- Перевірка зміни LastWriteTime ---
                if prev_modification_time_iso:
                    prev_modification_time_ts =
datetime.datetime.fromisoformat(prev_modification_time_iso).timestamp()
                    # Аномалія: LastWriteTime відкотився на значно старіший
(ігноруємо збільшення, бо це нормальний ріст логу)
```

```
                    if current_modification_time_ts < prev_modification_time_ts
- MTIME_ROLLBACK_THRESHOLD_SEC:
                        log_event(
                            f"Аномалія (Підміна): LastWriteTime файлу {filepath}
раптово відкотився. "
                            f"Було:
{datetime.datetime.fromtimestamp(prev_modification_time_ts,
tz=datetime.timezone.utc).isoformat()}, "
                            f"Стало:
{datetime.datetime.fromtimestamp(current_modification_time_ts,
tz=datetime.timezone.utc).isoformat()}.",
                            "WARNING")
                        anomalies += 1

                # --- Перевірка зміни LastAccessTime ---
                if prev_access_time_iso:
                    prev_access_time_ts =
datetime.datetime.fromisoformat(prev_access_time_iso).timestamp()
                    # Аномалія: LastAccessTime відкотився на значно старіший
                    if current_access_time_ts < prev_access_time_ts -
ATIME_ROLLBACK_THRESHOLD_SEC:
                        log_event(
                            f"Аномалія (Підміна): LastAccessTime файлу
{filepath} раптово відкотився. "
                            f"Було:
{datetime.datetime.fromtimestamp(prev_access_time_ts,
tz=datetime.timezone.utc).isoformat()}, "
                            f"Стало:
{datetime.datetime.fromtimestamp(current_access_time_ts,
tz=datetime.timezone.utc).isoformat()}.",
                            "WARNING")
                        anomalies += 1

                # --- Перевірка зміни розміру ---
                if prev_size is not None:
                    size_diff = current_size - prev_size
                    percentage_change = abs(size_diff) / prev_size if prev_size
> 0 else float(
                        'inf') if size_diff != 0 else 0

                    if size_diff > 0 and (
                            size_diff >
SUBSTITUTION_SIZE_INCREASE_THRESHOLD_BYTES or percentage_change >
SUBSTITUTION_SIZE_INCREASE_THRESHOLD_PERCENT):
                        log_event(
                            f"Аномалія (Підміна): Раптовий значний приріст
розміру файлу {filepath}. Було {prev_size}, Стало {current_size}.",
                            "WARNING")
                        anomalies += 1
                    elif size_diff < 0 and percentage_change >
SUBSTITUTION_SIZE_DECREASE_THRESHOLD_PERCENT:
                        log_event(
                            f"Аномалія (Підміна): Раптове значне зменшення
розміру файлу {filepath}. Було {prev_size}, Стало {current_size}.",
                            "WARNING")
                        anomalies += 1

                # --- Кореляція CreationTime файлу з часом першої події у ньому
---
                first_event_dt = get_first_event_timestamp(filepath)

                if first_event_dt:
                    file_creation_dt =
```

```
datetime.datetime.fromtimestamp(current_creation_time_ts,

tz=datetime.timezone.utc)

                    # Аномалія: CreationTime файлу значно НОВІШИЙ, ніж час
першої події
                    if file_creation_dt > first_event_dt + datetime.timedelta(
                            days=FIRST_EVENT_CTIME_DIFFERENCE_NEWER_DAYS):
                        log_event(
                            f"Аномалія (Підміна): CreationTime файлу
({file_creation_dt.isoformat()}) значно новіший, "
                            f"ніж час першої події в ньому
({first_event_dt.isoformat()}) для {filepath}.",
                            "WARNING"
                        )
                        anomalies += 1

            else:
                log_event(f"Новий файл EVTX або вперше відстежується для
підміни: {filepath}", "INFO",
                          console_output=False)
        except FileNotFoundError:
            log_event(f"Файл {filepath} не знайдено під час перевірки підміни
(можливо, видалено).", "WARNING")
        except OSError as e:
            log_event(f"Помилка доступу до файлу {filepath} при перевірці
підміни: {e}", "ERROR")
        except Exception as e:
            log_event(f"Неочікувана помилка при аналізі підміни файлу
{filepath}: {e}", "ERROR")

    save_state_to_file(current_metadata_state, state_file)

    if anomalies:
        log_event(f"Виявлено {anomalies} аномалій, що вказують на можливу
підміну EVTX файлів (метадані).",
                  "WARNING")
    else:
        log_event(f"Аномалій підміни EVTX файлів (метадані) не виявлено.",
"INFO", console_output=False)
    return


def main_evtx_monitoring_loop():
    """
    Головний цикл моніторингу, який запускає всі алгоритми періодично.
    """
    log_event("--- Запуск циклу моніторингу модифікацій EVTX файлів ---",
"INFO")
    log_event(f"Логи зберігаються у: {LOG_FILE}", "INFO")
    log_event(f"Каталог стану: {LOG_DIR}", "INFO")

    # Визначення, які саме файли аналізувати
    files_for_analysis = [
        os.path.join(os.environ.get('SystemRoot', 'C:\\Windows'), 'System32',
'winevt', 'Logs', 'Security.evtx'),
        os.path.join(os.environ.get('SystemRoot', 'C:\\Windows'), 'System32',
'winevt', 'Logs', 'System.evtx'),
        os.path.join(os.environ.get('SystemRoot', 'C:\\Windows'), 'System32',
'winevt', 'Logs', 'Application.evtx'),
        os.path.join(os.environ.get('SystemRoot', 'C:\\Windows'), 'System32',
'winevt', 'Logs', 'Windows PowerShell.evtx'),
    ]
```

```
    try:
        while True:
            log_event("\n--- Початок нового циклу перевірки... ---", "INFO",
console_output=True)

            # Алгоритм 1: Моніторинг хеш-сум та розміру
            monitor_evtx_integrity(files_for_analysis)

            # Алгоритм 2: Аналіз внутрішньої структури
            if files_for_analysis:
                log_event(f"Запуск аналізу внутрішньої структури для
{len(files_for_analysis)} файлів...",
                          "INFO")
                for evtx_path in files_for_analysis:
                    log_event(f"Аналіз внутрішньої структури файлу:
{evtx_path}", "INFO", console_output=False)
                    internal_structure_anomalies =
analyze_evtx_internal_structure(evtx_path)
                    if internal_structure_anomalies:
                        log_event(
                            f"Виявлено {len(internal_structure_anomalies)}
аномалій у внутрішній структурі {evtx_path}.",
                            "WARNING")
            else:
                log_event("Немає файлів для глибокого аналізу внутрішньої
структури EVTX.", "INFO",
                          console_output=False)

            # Алгоритм 3: Виявлення підмін файлів за метаданими
            analyze_evtx_file_substitution(files_for_analysis)

            log_event(f"--- Цикл перевірки завершено. Очікування
{MONITORING_INTERVAL_SECONDS} секунд... ---", "INFO",
                      console_output=True)
            time.sleep(MONITORING_INTERVAL_SECONDS)

    except KeyboardInterrupt:
        log_event("Моніторинг EVTX файлів зупинено користувачем.", "INFO")
    except Exception as e:
        log_event(f"Критична помилка в головному циклі моніторингу: {e}",
"ERROR")


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

ДОДАТОК Б СКРИПТ EVTXSERVICEREGISTRYMONITOR.PY

```
import os
import datetime
import logging
import time
import winreg
import psutil
import win32evtlog
from EvtxMonitor import save_state_to_file, load_state_from_file

# --- Налаштування логування ---
LOG_DIR = "C:\\Logs_EvtxMonitor"
if not os.path.exists(LOG_DIR):
    try:
        os.makedirs(LOG_DIR)
    except OSError as e:
        print(f"ПОМИЛКА: Не вдалося створити каталог для логів {LOG_DIR}: {e}")
        exit(1)

LOG_FILE = os.path.join(LOG_DIR, "EvtxServiceRegistryMonitor.log")

logger = logging.getLogger("EvtxServiceRegistryMonitor")
logger.setLevel(logging.INFO)

file_handler = logging.FileHandler(LOG_FILE)
formatter = logging.Formatter('%(asctime)s [%(levelname)s]
%(module)s:%(funcName)s - %(message)s')
file_handler.setFormatter(formatter)

logger.addHandler(file_handler)
logger.propagate = False

# --- Константа для інтервалу моніторингу ---
MONITORING_INTERVAL_SECONDS = 30


def log_event(message, level="INFO", console_output=True):
    if level == "INFO":
        logger.info(message)
    elif level == "WARNING":
        logger.warning(message)
    elif level == "ERROR":
        logger.error(message)
    else:
        logger.debug(message)

    if console_output:
        print(f"[{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}]
[{level}] {message}")


# --- Основні Алгоритми ---
def get_service_status():
    try:
        service = psutil.win_service_get("eventlog")
        return service.as_dict()["status"].upper()  # RUNNING / STOPPED
    except Exception:
        return None


def monitor_eventlog_service_status():
```

```
    """
    Алгоритм 1: Моніторинг стану служби "Журнал подій Windows" (EventLog).
    Регулярно перевіряє стан служби та фіксує будь-які зміни як підозрілі події.
    Також аналізує події Windows Event Log, пов'язані з керуванням службою.
    """
    log_event("--- Алгоритм: Моніторинг стану служби 'Журнал подій Windows' ---
", "INFO")
    state_file = os.path.join(LOG_DIR, "eventlog_service_state.json")
    previous_state = load_state_from_file(state_file)
    current_status = get_service_status()
    anomalies = 0

    try:
        if current_status:
            log_event(f"Поточний стан служби 'Журнал подій Windows':
{current_status}", "INFO", console_output=False)
            current_data = {"status": current_status,
                            "timestamp":
datetime.datetime.now(datetime.timezone.utc).isoformat()}

            if "status" in previous_state and previous_state["status"] !=
current_status:
                log_event(
                    f"Аномалія (Служба EventLog): Зміна стану служби 'Журнал
подій Windows'. "
                    f"Було: {previous_state['status']} (з
{previous_state['timestamp']}), Стало: {current_status}.",
                    "WARNING"
                )
                anomalies += 1

            save_state_to_file(current_data, state_file)
        else:
            log_event(
                "Не вдалося визначити стан служби 'Журнал подій Windows'.
Можливо, служба не знайдена або недостатньо прав.",
                "ERROR")
            anomalies += 1

    except Exception as e:
        log_event(f"Неочікувана помилка при перевірці стану служби 'Журнал подій
Windows': {e}", "ERROR")
        anomalies += 1

    # Аналіз подій в System.evtx, пов'язаних з керуванням службою
    log_name = "System"
    system_log_state_file = os.path.join(LOG_DIR,
"system_log_processed_id.json")
    last_system_log_id =
load_state_from_file(system_log_state_file).get("last_record_id", 0)
    current_max_record_id = last_system_log_id
    try:
        hand = win32evtlog.OpenEventLog(None, log_name)  # None для локальної
машини
        flags = win32evtlog.EVENTLOG_BACKWARDS_READ |
win32evtlog.EVENTLOG_SEQUENTIAL_READ
        num_events_to_read_per_chunk = 100

        events_read = True
        while events_read:
            events = win32evtlog.ReadEventLog(hand, flags, 0,
num_events_to_read_per_chunk)
            if not events:
```

```
                    break

            for event in events:
                if event.RecordNumber > current_max_record_id:
                    current_max_record_id = event.RecordNumber

                if event.EventID in [7034, 7035, 7036, 7040, 7046] and
event.StringInserts and "eventlog" in event.StringInserts[0].lower():
                    log_event(
                        f"Аномалія (Служба EventLog - Подія Логу): Виявлено
подію зміни стану/конфігурації служби EventLog (ID {event.EventID}) у журналі
'{log_name}'. "
                        f"Record ID: {event.RecordNumber}, Час події:
{event.TimeGenerated.isoformat()}. Деталі: '{event.StringInserts[0]}'.",
                        "WARNING"
                    )
                    anomalies += 1

            if not events_read:
                break
        win32evtlog.CloseEventLog(hand)

        if current_max_record_id > last_system_log_id:
            save_state_to_file({"last_record_id": current_max_record_id},
system_log_state_file)

    except Exception as e:
        log_event(f"Неочікувана помилка при аналізі журналу '{log_name}' для
подій служби EventLog: {e}", "ERROR")
        anomalies += 1

    if anomalies > 0:
        log_event(f"Виявлено {anomalies} аномалій, пов'язаних зі службою 'Журнал
подій Windows'.",
                                        "WARNING")
    else:
        log_event(f"Аномалій, пов'язаних зі службою 'Журнал подій Windows', не
виявлено.", "INFO",
                                        console_output=False)


def monitor_registry_changes():
    """
    Алгоритм 2: Моніторинг змін у ключових гілках реєстру, пов'язаних з
конфігурацією журналів та Audit Policy.
    """
    log_event("--- Алгоритм: Моніторинг змін у реєстрі ---", "INFO")
    state_file = os.path.join(LOG_DIR, "registry_state.json")
    previous_state = load_state_from_file(state_file)
    current_state = {}
    anomalies = 0

    # Ключові гілки реєстру та параметри для моніторингу
    # Ці параметри прямо вказують на конфігурацію Event Logs
    registry_targets = {
        # Конфігурація самої служби EventLog
        "EventLogService": {
            "path": r"SYSTEM\CurrentControlSet\Services\EventLog",
            "values": ["Start"]  # 2=Auto, 3=Demand, 4=Disabled
        },
        # Конфігурація окремих журналів (MaxSize, Retention)
        "ApplicationLog": {
            "path": r"SYSTEM\CurrentControlSet\Services\EventLog\Application",
```

```
            "values": ["MaxSize", "Retention"]  # Retention: 0=Overwrite as
needed, 0xFFFFFFFF=Do not overwrite
        },
        "SecurityLog": {
            "path": r"SYSTEM\CurrentControlSet\Services\EventLog\Security",
            "values": ["MaxSize", "Retention"]
        },
        "SystemLog": {
            "path": r"SYSTEM\CurrentControlSet\Services\EventLog\System",
            "values": ["MaxSize", "Retention"]
        },
        "PowerShellLog": {
            "path": r"SYSTEM\CurrentControlSet\Services\EventLog\Windows
PowerShell",
            "values": ["MaxSize", "Retention"]
        },
    }

    for target_name, config in registry_targets.items():
        reg_path = config["path"]
        values_to_monitor = config["values"]

        try:
            with winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, reg_path, 0,
winreg.KEY_READ) as key:
                current_values_data = {}
                for val_name in values_to_monitor:
                    try:
                        value, type_id = winreg.QueryValueEx(key, val_name)
                        current_values_data[val_name] = value  # Зберігаємо
значення як є (int, str тощо)
                    except FileNotFoundError:
                        current_values_data[val_name] = None
                        log_event(f"DEBUG: Значення '{val_name}' не знайдено у
ключі {reg_path}.",
                                                            "DEBUG",
console_output=False)

                current_state[reg_path] = current_values_data

                if reg_path in previous_state:
                    prev_values_data = previous_state[reg_path]

                    for val_name, current_value in current_values_data.items():
                        prev_value = prev_values_data.get(val_name)

                        if current_value != prev_value:
                            log_message = (
                                f"Аномалія (Реєстр): Змінено значення
'{val_name}' для '{target_name}' у реєстрі ({reg_path}). "
                                f"Було: '{prev_value}', Стало:
'{current_value}'."
                            )
                            log_event(log_message, "WARNING")
                            anomalies += 1

                            # Додаткові деталі для критичних змін
                            if target_name == "EventLogService" and val_name ==
"Start":
                                if current_value == 4:  # 4 = Disabled
                                    log_event(
                                        f"Критична Аномалія: Службу 'Журнал
подій Windows' вимкнено у реєстрі (Start=4)!",
```

```
                                        "WARNING")
                                elif prev_value == 4 and current_value != 4:
                                    log_event(
                                        f"Інформація: Службу 'Журнал подій
Windows' увімкнено у реєстрі (Start з 4 на {current_value})!",
                                        "INFO")

                            elif val_name == "Retention":
                                if current_value == 0xFFFFFFFF:  # 0xFFFFFFFF =
Do not overwrite events
                                    log_event(
                                        f"Критична Аномалія: Журнал
'{target_name}' налаштовано НЕ перезаписувати події (зберігати вічно). Це може
призвести до зупинки логування при заповненні!",
                                        "WARNING")
                                elif prev_value == 0xFFFFFFFF and current_value
!= 0xFFFFFFFF:
                                    log_event(
                                        f"Інформація: Журнал '{target_name}'
змінено з 'не перезаписувати' на '{current_value}'.",
                                        "INFO")
                            elif val_name == "MaxSize":
                                if current_value is not None and prev_value is
not None and current_value < prev_value:
                                    log_event(
                                        f"Критична Аномалія: Зменшено
максимальний розмір журналу '{target_name}' з {prev_value} на {current_value}.
Це може призвести до швидшої ротації та втрати старих логів!",
                                        "WARNING")
                else:
                    log_event(
                        f"Вперше відстежуються параметри реєстру для
'{target_name}': {reg_path}", "INFO",
                        console_output=False)

        except FileNotFoundError:
            log_event(
                f"Ключ реєстру '{reg_path}' для '{target_name}' не знайдено.
Можливо, він не існує.", "WARNING")
        except PermissionError:
            log_event(
                f"Недостатньо прав для читання ключа реєстру {reg_path}.
Запустіть скрипт з правами адміністратора.",
                "ERROR")
            anomalies += 1
        except Exception as e:
            log_event(f"Неочікувана помилка при моніторингу реєстру для
'{target_name}': {e}",
                                                "ERROR")
            anomalies += 1

    save_state_to_file(current_state, state_file)

    if anomalies > 0:
        log_event(f"Виявлено {anomalies} аномалій, пов'язаних зі змінами у
реєстрі.", "WARNING")
    else:
        log_event(f"Аномалій, пов'язаних зі змінами у реєстрі, не виявлено.",
"INFO", console_output=False)


def analyze_audit_policy_events():
    """
```

```
    Алгоритм 3: Аналіз подій Windows Event Log, пов'язаних зі змінами Audit
Policy.
    Основна подія для моніторингу - Event ID 4719 (System audit policy was
changed).
    Також моніториться Event ID 1102 (The audit log was cleared).
    """
    log_event("--- Алгоритм: Аналіз подій зміни політики аудиту ---", "INFO")
    security_log_path = os.path.join(os.environ.get('SystemRoot',
'C:\\Windows'), 'System32', 'winevt', 'Logs',
                                     'Security.evtx')
    state_file = os.path.join(LOG_DIR, "audit_policy_events_state.json")
    last_event_id_processed =
load_state_from_file(state_file).get("last_event_id",

0)  # Зберігаємо останній оброблений Event ID
    anomalies = 0

    if not os.path.exists(security_log_path) or not os.access(security_log_path,
os.R_OK):
        log_event(
            f"Файл {security_log_path} не доступний для аналізу подій політики
аудиту. Пропускаємо.", "ERROR")
        return

    try:
        hand = win32evtlog.OpenEventLog(None, "Security")  # Відкриваємо журнал
Security
        flags = win32evtlog.EVENTLOG_BACKWARDS_READ |
win32evtlog.EVENTLOG_SEQUENTIAL_READ
        num_events_to_read_per_chunk = 100

        current_max_record_id = last_event_id_processed  # Для відстеження
найвищого обробленого ID в поточному запуску

        log_event(
            f"Аналіз журналу 'Security' для подій зміни політики аудиту (з
Record ID {last_event_id_processed}).",
            "DEBUG", console_output=False)

        events_read = True
        while events_read:
            events = win32evtlog.ReadEventLog(hand, flags, 0,
num_events_to_read_per_chunk)
            if not events:
                break

            for event in events:
                if event.RecordNumber <= last_event_id_processed:
                    events_read = False  # Зупиняємо обробку, дійшли до вже
оброблених
                    break

                if event.RecordNumber > current_max_record_id:
                    current_max_record_id = event.RecordNumber

                if event.EventID == 4719:  # Event ID для зміни системної
політики аудиту
                    log_event(
                        f"Аномалія (Політика Аудиту): Виявлено зміну системної
політики аудиту (Event ID 4719) у журналі 'Security'. "
                        f"Record ID: {event.RecordNumber}, Час події:
{event.TimeGenerated.isoformat()}.",
                        "WARNING"
```

```
                    )
                    anomalies += 1
                elif event.EventID == 1102:  # Event ID для очищення журналу
                    log_cleared_by = "N/A"
                    if event.StringInserts and len(event.StringInserts) > 1:
                        log_cleared_by = event.StringInserts[1]  # Account Name

                    log_event(
                        f"Критична Аномалія (Очищення Журналу): Виявлено
очищення журналу '{event.StringInserts[0] if event.StringInserts else
'Невідомий'}' (Event ID 1102) у журналі 'Security'. "
                        f"Record ID: {event.RecordNumber}, Час події:
{event.TimeGenerated.isoformat()}. "
                        f"Ким очищено: {log_cleared_by}.",
                        "WARNING"
                    )
                    anomalies += 1

            if not events_read:  # Якщо ми вийшли з внутрішнього циклу через вже
оброблені події
                break

        win32evtlog.CloseEventLog(hand)

        if current_max_record_id > last_event_id_processed:
            save_state_to_file({"last_event_id": current_max_record_id},
state_file)

    except Exception as e:
        log_event(
            f"Неочікувана помилка при аналізі журналу 'Security' для подій
політики аудиту: {e}", "ERROR")
        anomalies += 1

    if anomalies > 0:
        log_event(f"Виявлено {anomalies} аномалій, пов'язаних зі змінами
політики аудиту.",
                                          "WARNING")
    else:
        log_event(f"Аномалій, пов'язаних зі змінами політики аудиту, не
виявлено.", "INFO",
                                          console_output=False)


def main_EvtxServiceRegistryMonitor_loop():
    """
    Головний цикл моніторингу, що запускаються періодично.
    """
    log_event("--- Запуск циклу моніторингу перевірок EventLog ---", "INFO")
    log_event(f"Логи зберігаються у: {LOG_FILE}", "INFO")
    log_event(f"Каталог стану: {LOG_DIR}", "INFO")

    try:
        while True:
            log_event("\n--- Початок нового циклу перевірок... ---", "INFO",
console_output=True)

            monitor_eventlog_service_status()
            monitor_registry_changes()
            analyze_audit_policy_events()

            log_event(f"--- Цикл перевірок завершено. Очікування
{MONITORING_INTERVAL_SECONDS} секунд... ---",
```

```
                    "INFO", console_output=True)
            time.sleep(MONITORING_INTERVAL_SECONDS)

    except KeyboardInterrupt:
        log_event("Моніторинг перевірок EventLog зупинено користувачем.",
"INFO")
    except Exception as e:
        log_event(f"Критична помилка в головному циклі перевірок: {e}", "ERROR")


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

ДОДАТОК В СКРИПТ EVTXPROCESSMONITOR.PY

```
import os
import datetime
import logging
import time
import win32evtlog
import re
from EvtxMonitor import save_state_to_file, load_state_from_file

# --- Налаштування логування ---
LOG_DIR = "C:\\Logs_EvtxMonitor"
if not os.path.exists(LOG_DIR):
    try:
        os.makedirs(LOG_DIR)
    except OSError as e:
        print(f"ПОМИЛКА: Не вдалося створити каталог для логів {LOG_DIR}: {e}")
        exit(1)

LOG_FILE = os.path.join(LOG_DIR, "EvtxProcessMonitor.log")

logger = logging.getLogger("EvtxProcessMonitor")
logger.setLevel(logging.INFO)

file_handler = logging.FileHandler(LOG_FILE)
formatter = logging.Formatter('%(asctime)s [%(levelname)s]
%(module)s:%(funcName)s - %(message)s')
file_handler.setFormatter(formatter)

logger.addHandler(file_handler)
logger.propagate = False

# --- Константа для інтервалу моніторингу ---
MONITORING_INTERVAL_SECONDS = 10


def log_event(message, level="INFO", console_output=True):
    if level == "INFO":
        logger.info(message)
    elif level == "WARNING":
        logger.warning(message)
    elif level == "ERROR":
        logger.error(message)
    else:
        logger.debug(message)

    if console_output:
        print(f"[{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}]
[{level}] {message}")


def analyze_malicious_tool_usage():
    """
    Алгоритм: Виявлення маніпуляцій з журналами подій через командний рядок.
    Моніторить Event ID 4688 (Process Creation) у журналі Security на наявність
    команд, що відповідають техніці MITRE ATT&CK T1562.002.
    """
    log_event("--- Алгоритм: Аналіз команд для маніпуляції журналами (T1562.002)
---", "INFO")
    state_file = os.path.join(LOG_DIR, "log_tampering_commands_state.json")
    last_event_id_processed =
load_state_from_file(state_file).get("last_event_id", 0)
```

```
    anomalies = 0

    tampering_patterns = {
        "LOG_CLEARING": [
            r"wevtutil(?:\.exe)?\s+cl(?:ear-
log)?\s+(?:security|system|application|setup)",
            r"clear-eventlog\s+-
\s*logname\s+(?:security|system|application|setup)"
        ],
        "LOG_SERVICE_DISABLE": [

r"wevtutil(?:\.exe)?\s+sl\s+(?:security|system|application|setup)\s+(?:/e:false|
/ms:\d+)",
            r"limit-eventlog",
            r"sc(?:\.exe)?\s+config\s+eventlog\s+start=\s*disabled",
            r"set-service\s+-\s*name\s+eventlog\s+-\s*startuptype\s+disabled"
        ],
        "LOG_SERVICE_STOP": [
            r"sc(?:\.exe)?\s+stop\s+eventlog",
            r"net\s+stop\s+eventlog",
            r"taskkill\s+/f\s+/fi\s+\"services\s+eq\s+eventlog\"",
            r"stop-service\s+-\s*name\s+eventlog"
        ],
        "LOG_REGISTRY_TAMPERING": [
            r"reg\s+add\s+hklm\\system\\currentcontrolset\\services\\eventlog",

r"reg\s+delete\s+hklm\\system\\currentcontrolset\\services\\eventlog"
        ],
        "LOG_FILES_ACL_TAMPERING": [
            r"icacls\s+.*",
            r"takeown\s+.*"
        ]
    }

    try:
        hand = win32evtlog.OpenEventLog(None, "Security")
        flags = win32evtlog.EVENTLOG_BACKWARDS_READ |
win32evtlog.EVENTLOG_SEQUENTIAL_READ
        current_max_record_id = last_event_id_processed

        events = True
        while events:
            events = win32evtlog.ReadEventLog(hand, flags, 0)
            if not events:
                break

            for event in events:
                if event.RecordNumber <= last_event_id_processed:
                    events = False
                    break

                if event.RecordNumber > current_max_record_id:
                    current_max_record_id = event.RecordNumber

                if event.EventID == 4688:  # Process Creation
                    # CommandLine - це індекс 8
                    if len(event.StringInserts) > 8:
                        command_line = event.StringInserts[8] if
event.StringInserts[8] else ""
                        if not command_line:
                            continue

                        # Перетворюємо на нижній регістр і замінюємо множинні
```

```
пробіли на один, обрізаємо пробіли на кінцях
                        normalized_command_line = re.sub(r'\s+', ' ',
command_line.lower()).strip()

                        for category, patterns in tampering_patterns.items():
                            for pattern_regex in patterns:
                                if re.search(pattern_regex,
normalized_command_line):
                                    # Окремий, більш точний контроль для ACL
маніпуляцій
                                    # Ця перевірка тепер спрацює після того, як
загальний regex для icacls/takeown знайшов збіг
                                    if category == "LOG_FILES_ACL_TAMPERING":
                                        # Тригер спрацьовує, тільки якщо команда
стосується папки з логами
                                        if "winevt\\logs" not in
normalized_command_line:
                                            continue  # Це легітимне
використання icacls/takeown, ігноруємо

                                    process_name =
os.path.basename(event.StringInserts[5])
                                    log_event(
                                        f"АНОМАЛІЯ ({category}): Виявлено
підозрілу команду маніпуляції журналами! "
                                        f"Процес: '{process_name}', Патерн
(regex): '{pattern_regex}', "
                                        f"Record ID: {event.RecordNumber}, Час:
{event.TimeGenerated.isoformat()}. "
                                        f"Повний рядок:
'{event.StringInserts[8]}'",
                                        "WARNING"
                                    )
                                    anomalies += 1
                                    break
                            else:
                                continue
                            break

        win32evtlog.CloseEventLog(hand)
        if current_max_record_id > last_event_id_processed:
            save_state_to_file({"last_event_id": current_max_record_id},
state_file)

    except Exception as e:
        log_event(f"Неочікувана помилка при аналізі команд: {e}", "ERROR")

    if anomalies > 0:
        log_event(
            f"Виявлено {anomalies} аномалій, пов'язаних з використанням
легітимних системних інструментів.", "WARNING")
    else:
        log_event(f"Аномалій використання легітимних системних інструментів не
виявлено.", "INFO",
                                    console_output=False)


def main_process_monitor_loop():
    """
    Головний цикл моніторингу для аналізу використання легітимних системних
інструментів.
    """
    log_event("--- Запуск циклу моніторингу використання системних інструментів
```

```
---", "INFO")
    log_event(f"Логи зберігаються у: {LOG_FILE}", "INFO")
    log_event(f"Каталог стану: {LOG_DIR}", "INFO")

    try:
        while True:
            log_event("\n--- Початок нового циклу аналізу процесів... ---",
"INFO",
                                              console_output=True)

            analyze_malicious_tool_usage()

            log_event(
                f"--- Цикл аналізу процесів завершено. Очікування
{MONITORING_INTERVAL_SECONDS} секунд... ---", "INFO",
                console_output=True)
            time.sleep(MONITORING_INTERVAL_SECONDS)

    except KeyboardInterrupt:
        log_event("Моніторинг використання системних інструментів зупинено
користувачем.", "INFO")
    except Exception as e:
        log_event(f"Критична помилка в головному циклі аналізу процесів: {e}",
"ERROR")


if __name__ == "__main__":
    # Для Event ID 4688, необхідно увімкнути "Audit Process Creation"
    # та "Include command line in process creation events" в Group Policy.
    main_process_monitor_loop()
```