Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)
на тему: «Програмний модуль виявлення шкідливого додатку в ОС Windows на основі поведінкового аналізу»

Виконавець: студент IV курсу, групи КБ-42

_____ **Іван НЕЧИПОРЕНКО**
(підпис) (ім'я прізвище)

	Прізвище, ініціали	**Підпис**
Керівник	Лариса МИРУТЕНКО	

Нормоконтроль	Андрій ФЕСЕНКО	

Київ 2023

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньо-професійної програми)

Студента **КБ-42** **НЕЧИПОРЕНКА Івана Петровича**
(група) (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Програмний модуль виявлення шкідливого додатку в ОС Windows на основі поведінкового аналізу

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2023 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Архітектура операційних систем, події безпеки, поведінковий аналіз.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно дослідити види шкідливого програмного забезпечення та методи його дослідження. Дослідити особливості ОС Windows в контексті атак та вбудовані джерела логів, спроектувати модуль виявлення шкідливого ПЗ та розробити його.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Програмний модуль виявлення шкідливих додатків та дослідження поведінки будь-якого процесу в ОС Windows

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:24 жовтня 2022 року

Завдання видала _____ (підпис) Лариса МИРУТЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Іван НЕЧИПОРЕНКО (ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.10.2022 – 12.11.2022	*виконано*
2	Аналіз відкритих джерел	12.01.2023 – 10.02.2023	*виконано*
3	Дослідження використання поведінкового аналізу для виявлення шкідливих додатків	11.02.2023-13.03.2023	*виконано*
4	Огляд вбудованих засобів логування подій безпеки в ОС Windows	14.03.2023 – 15.04.2023	*виконано*
5	Дослідження додатку SYSMON	16.04.2023 – 03.05.2023	*виконано*
6	Розробка архітектури програмного модуля для виявлення шкідливого додатку на основі поведінкового аналізу	4.05.2023 – 18.05.2023	*виконано*
7	Програмна реалізація модуля	19.05.2023 – 30.05.2023	*виконано*
8	Проведення тестування	31.05.2023 – 02.06.2023	*виконано*
9	Оформлення пояснювальної записки	03.06.2023 – 07.06.2023	*виконано*
10	Підготовка до захисту кваліфікаційної роботи	08.06.2023 – 11.06.2023	*виконано*

Завдання видав _____ (підпис) Лариса МИРУТЕНКО (ім'я, прізвище)

Завдання прийняв до виконання _____ (підпис) Іван НЕЧИПОРЕНКО (ім'я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається з вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 82 сторінки, включає в себе зміст, вступ, три розділи кваліфікаційної роботи, висновки та список джерел. Робота містить 4 додатки із загальною кількістю сторінок 17. У пояснювальній записці кваліфікаційної роботи міститься 37 рисунків і 7 таблиць.

Метою роботи є розробка програмного модуля для виявлення шкідливих додатків в операційній системі Windows на основі поведінкового аналізу системних подій, які моніторяться за допомогою Sysmon.

Об'єктом дослідження є процеси моніторингу та аналізу системних подій в операційній системі Windows за допомогою Sysmon.

Предметом дослідження є програмний модуль, що використовує методи поведінкового аналізу для виявлення шкідливих додатків, використовуючи дані системного моніторингу.

Методи дослідження кваліфікаційної роботи:

- аналіз відкритих джерел інформації;
- аналіз технічної документації;
- спостереження шляхом логування подій системи;
- порівняння.

Для досягнення зазначеної мети поставлено наступні завдання:

- дослідити структуру операційної системи Windows та методики моніторингу системних подій;
- проаналізувати техніки поведінкового аналізу для виявлення шкідливих додатків;
- розробити програмний модуль, який здійснює моніторинг та аналіз системних подій для виявлення шкідливих додатків в операційній системі Windows.

Практичною цінністю отриманих результатів є програмний модуль, який може бути використаний для виявлення шкідливих додатків в операційній системі Windows.

Ключові слова: операційна система Windows, системний моніторинг, поведінковий аналіз, шкідливий додаток, програмний модуль.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ	–	програмне забезпечення
ОС	–	операційна система
Sysmon	–	System Monitor
C#	–	мова програмування C#
API	–	Application Programming Interface
DLL	–	Динамічні бібліотеки
EDR	–	Endpoint Detection and Response
TTPs	–	Tactics, Techniques and Procedures
SIEM	–	Security Information and Event Management
IOCs	–	Indicators of Compromise
DLL	–	Dynamic-Link Library
XML	–	eXtensible Markup Language
GUI	–	Graphical User Interface

ВСТУП

Інформаційні технології є невід’ємною частиною нашого життя та стали важливою частиною багатьох його аспектів, починаючи з базової обробки та поширення інформації і закінчуючи складними бізнес-операціями, управлінням критичною інфраструктурою та обробкою надважливих даних. Джерелом цих змін є швидкий розвиток інформаційних технологій, які відкривають безліч нових можливостей, але одночасно створюють нові виклики і загрози.

Одним з ключових елементів інформаційних систем є програмне забезпечення. Воно є основою більшості сучасних девайсів - від смартфонів і персональних комп'ютерів до серверів, автомобілів і критичної інфраструктури. Однак, разом з цим, виникає і низка проблем, пов'язаних з безпекою програмного забезпечення.

Всі різновиди шкідливого ПЗ були є і будуть одними з найбільших загроз для інформаційних технологій. Між появою першого доступного широким масам комп’ютера, що виконував прості арифметичні операції та появою першого вірусу в історії пройшло 7 років, і це за умові відсутності інтернету. Цей факт демонструє динаміку розвитку шкідливого програмного забезпечення, його модифікацій та розвитку незалежно від типу системи. Оскільки завжди будуть ті, хто хоче порушити правила та втрутитися в чужу систему.

Тому, виявлення шкідливого програмного забезпечення є одним з ключових завдань кібербезпеки. Одним з ефективних підходів до цієї проблеми є поведінковий аналіз, який спрямований на виявлення аномальної поведінки в системі, яка може вказувати на наявність шкідливого ПЗ.

Метою цієї кваліфікаційної роботи є розробка програмного модуля виявлення шкідливого додатку в ОС Windows на основі поведінкового аналізу. Для досягнення цієї мети, буде проведено дослідження теоретичних основ виявлення шкідливого програмного забезпечення, компонентів операційної системи Windows в процесі атаки.

РОЗДІЛ 1
МЕТОДИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ПОВЕДІНКОВОГО АНАЛІЗУ

1.1 Шкідливе програмне забезпечення

Шкідливе ПЗ, відоме як malware, включає в себе різні типи програмного забезпечення, призначеного для здійснення впливу на інформацію, систему обробки цієї інформації та мережу по якій здійснюється передача інформації з метою порушення її властивостей, а саме цілісності, доступності та конфіденційності. Особливість malware полягає в можливості автономного функціонування.

Його можна класифікувати за наступними типами : Viruses, Worms, Trojans, Rootkits, Spyware, Adware, Ransomware, Botnets.

- Viruses (Віруси) – це шкідливі програми, яким для розповсюдження та розмноження необхідно модифікувати код інших програм.
- Worms (Черв’яки) – це шкідливі програми, які самостійно розповсюджуються, без необхідності втручання в інші програми.
- Trojans (Троянські коні) – це програми, які вдають корисні або нешкідливі, але при цьому містять шкідливий код.
- Rootkits (Руткіти) – це шкідливі програми, які приховують від користувача і системи присутність інших шкідливих програм.
- Spyware (Шпигунське ПЗ) – це шкідливе ПЗ, яке використовується для слідкування за діями користувача без його відома.
- Adware (Рекламне ПЗ) – це шкідливе ПЗ, яке використовується для відображення небажаної реклами на комп'ютері користувача.
- Ransomware (Вимагачі) – це шкідливе ПЗ, яке блокує доступ до файлів користувача або до всієї системи до того, як буде сплачено викуп.
- Cryptominers (криптомайнери) – це шкідливе ПЗ, що використовує обчислювальні ресурси зараженої системи для майнінгу криптовалют.

Окремою категорією слід розглянути хакерські інструменти. Самі по собі вони не несуть зловмисного функціоналу та можуть використовуватися як допоміжні утиліти адміністрування та моніторингу систем. Проте у руках зловмисника вони стають інструментами атаки, тобто з таким типом ПЗ повинен взаємодіяти безпосередньо зловмисник, надсилаючи команди управління. Через це їх можна відносити до шкідливого ПЗ.

До інструментів відносяться наступні типи ПЗ : Scanners, Keyloggers, Backdoors, Exploits, Automation Scripts.

- Scanners (сканери) – це інструменти, що використовуються для дослідження мережі. Зловмисники використовують модифіковані сканери з базами вразливостей, завдяки чому виявляють слабкі місця цільової системи.
- Keyloggers (кейлогери) – це програми, що відслідковують натискання клавіш з метою отримання парольних даних або іншої конфіденційної інформації з подальшою передачею зібраної інформації зловмиснику.
- Backdoors (запасний вихід) – це програми, завдяки яким зловмисник здійснює доступ до вже захопленої системи.
- Exploit kits (набори експлойтів) – це інструменти з наборами вразливостей ПЗ за допомогою якого зловмисник здійснює атаки на цільові системи, тобто це засіб автоматизації.
- Automation Scripts (скрипти автоматизації) – це готові сценарії, що запускаються зловмисником на зараженій системі в процесі атаки для автоматизації.

Найпоширенішими способами розповсюдження шкідливого ПЗ є поштові спам-кампанії, фішингові атаки, експлуатація вразливостей в системах, пристроях і програмному забезпеченні, втручання в ланцюги постачання оновлень ПЗ, веб-сайти, що містять контент для завантаження з інфікованим вмістом, використання неліцензійного ПЗ та через змінні носії інформації такі як USB-флешки та інші периферійні пристрої.

Цілями шкідливого ПЗ є комп'ютери, сервери, мережеві пристрої, хмарна інфраструктура, пристрої інтернет речей, смартфони, IT/OT – інфраструктура та інформація, що обробляється в цих системах.

Шкідливе ПЗ може поширюватися як самостійно (malware), так і в ролі допоміжного засобу (хакерські інструменти) при реалізації цілеспрямованої атаки на інформаційну систему, надаючи йому можливості виконувати команди через мережу та закріпити отриманий доступ в цільовій системі.

Необхідно сприймати середовище, в якому здійснюється захист інформації як абсолютно вороже. Це означає, що всі елементи системи, включаючи користувачі системи та наявний функціонал несуть потенційну загрозу. Розробники шкідливого ПЗ використовують вбудовані можливості та функції операційних систем та програм, приклад – використання VBA Macross в документах з пакету програм office).

Частою причиною компрометації системи шкідливим ПЗ стає неправильна конфігурація, приклад є групова політика, що дозволяє використання SMB протоколу з діалектом першої версії дає можливість здійснити Bind Shell.

1.2 Поведінковий аналіз для дослідження шкідливого ПЗ

Дослідження шкідливого ПЗ поділяється на дві основні методології - статичний та динамічний аналіз.

Статичний аналіз полягає у дослідженні коду програм, таблиць імпорту та експорту, PE-заголовків та не вимагає запуску програм. Результатами таких досліджень є шаблони (сигнатури) за якими можливо виявити шкідливе ПЗ ще до його запуску в системі та вжити відповідних дій.

Динамічний аналіз вимагає запустити досліджуваний зразок ПЗ (на відміну від статичного) у контрольованому середовищі та відслідкувати з якими компонентами ОС він взаємодіє. Для цього можна використовувати наступні інструменти динамічного аналізу такі як Process Explorer, Process Hacker, пісочниці, sysmon. Результатами таких досліджень є створення поведінкових сигнатур, завдяки яким можна відрізнити активність легітимного ПЗ від шкідливого та вжити відповідних заходів, не порушуючи нормальну експлуатацію системи.

Методи статичного та динамічного аналізу представлені в таблиці 1.1

Таблиця 1.1

Методи аналізу шкідливого програмного забезпечення

Метод	Статичний аналіз	Динамічний аналіз
Signature Analysis (Сигнатурний аналіз)	Здійснює пошук відомих шкідливих сигнатур в коді програм.	Може використовуватися в комбінації зі статичним
Hash Analysis (Аналіз хешу)	Порівнює хеш-суми файлів з базою шкідливих файлів (VirusTotal).	Може використовуватися в комбінації зі статичним
Heuristic Analysis (Евристичний аналіз)	Використовує набір правил для визначення можливих шкідливих властивостей в коді програми.	Визначення можливих шкідливих дій під час виконання.
Аналіз викликів API (API Call Analysis)	Може використовуватися в комбінації з динамічним	Слідкує за викликами API в процесі виконання
Behavioral Analysis (Поведінковий аналіз)	Може використовуватися в комбінації з динамічним	Відслідковує поведінку програми під час її виконання, аналізуючи взаємодію з системою.
Техніка ізоляції (Sandboxing)	Може використовуватися в комбінації з динамічним	Запуск ПЗ у віртуальному середовищі , дослідження взаємодії з системою.
Memory Dump Analysis (Аналіз дампу пам'яті (Memory)	Може використовуватися в комбінації з динамічним	Аналізує вмісту пам'яті системи після виконання програми

Поведінковий аналіз - це один з методів динамічного аналізу, який полягає в дослідженні дій шкідливого ПЗ в операційній системі. У порівнянні з іншими методами динамічного та поведінкового аналізу він має наступні головні переваги:

1) Виявляє невідомі варіанти шкідливого ПЗ, оскільки методи ухилення від систем захисту постійно вдосконалюються, але складові ОС, до яких воно намагається отримати доступ залишається незмінним.

2) Особливо ефективний метод у боротьбі з поліморфним шкідливим ПЗ, яке постійно змінює свій код зі збереженням закладеного функціоналу для ухилення від методів сигнатурного аналізу.

Саме ці переваги і стали вирішальними при виборі саме цього методу для використання у програмному модулі.

1.3 Маркери компрометації

Розглянемо поняття маркерів компрометації які також відомі як індикатори компрометації - це ознаки, що вказують на наявність шкідливого ПЗ в системі. Вони є результатом досліджень шкідливого ПЗ з використанням методів статичного та динамічного аналізу. До них відносяться наступні ознаки:

- Зміна хеш-сум файлів. Шкідливе ПЗ модифікує системні файли або файли застосунків. Порівняння хеш-сум оригіналу та модифікованого файлу і є маркером компрометації.
- Аномальна мережева активність. Це включає незвичайні мережеві з'єднання, підозрілу активність на мережевому рівні, невідомі або незвичайні порти, використання непідтримуваних протоколів.
- Зміни в реєстрі. Багато шкідливих програм змінюють налаштування в реєстрі операційної системи. Моніторинг змін гілок реєстру може допомогти виявити компрометацію.
- Нестандартні активності процесів. Це включає незвичайну активність процесів, споживання надмірних ресурсів, запуск непідтримуваних процесів або незвичайні зв'язки між процесами.
- Фрагменти коду. Маркери компрометації можуть включати підозрілі фрагменти коду в яких наявні ознаки використання технік обфускації або команд які спрямований на нетипову взаємодію з компонентами системи.
- Нестандартна активність облікових записів в системі. Це включає появу нових облікових записів, зміни в налаштуваннях груп, взаємодія під такими обліковими записами з компонентами системи, що раніше не відбувалися.

- Нестандартні зміни в системних налаштуваннях. Це можуть бути зміни в налаштуваннях безпеки, політиках груп, зміни правил доступу до файлів або інших системних параметрах.
- Використання невідомих або заборонених програм. Шкідливе ПЗ потрапивши в систему намагається ухилитися від систем захисту та уваги користувача шляхом маскування під відомі програм. Для виявлення виконується порівняння наявних програм зі списком тих що дозволені. У випадку виникнення розбіжностей знайдений файл поміщається в зону карантину та проводиться його додаткове дослідження
- Запуск системних служб. Шкідливе ПЗ для закріплення/просування в системі може скористатися доступними службами які не використовуються при звичайній експлуатації системи, наприклад планувальник завдань. Дослідження активних служб може підсвітити шкідливе ПЗ.
- Зміни в файловій системі: це можуть бути зміни в директоріях, файлових шляхах, доступі до файлів або інших аспектах файлової системи, що свідчать про несанкціоновану діяльність.
- Очищення системних журналів. Шкідливе ПЗ намагається приховати сліди перебування в системі. Маючи достатній рівень прав у системі воно очищає, перезаписує або вибірково видаляє записи системних журналів.

В процесі розслідування або перевірки маркери компрометації допомагають виявити наявність шкідливого ПЗ в системі та локалізувати заражену частину як окремої системи так і цілої інфраструктури, помістивши її в зону карантину для запобігання поширення зараження.

1.4 Особливості ОС Windows у контексті виявлення шкідливого ПЗ

Для того, щоб ефективно виявляти шкідливе ПЗ на основі поведінки, необхідно розглянути компоненти ОС Windows які піддаються атакам, або їх функціонал використовується для проведення атаки на інші компоненти.

Доцільно поглянути на компоненти ОС Windows в рамках концепції Kill Chain (Ланцюг знищення). Ця концепція була розроблена для відображення етапів атак та дій зловмисника на цих етапах (рисунок 1.1).

Delivery	Exploitation	Installation	Command & Control	Actions on Objectives	Execution
Email client,	Windows API	PowerShell	PowerShell	PowerShell	PowerShell
PowerShell,	VBScript	MSIExec	BITSAdmin	WMI	WMI
BITSAdmin,	PowerShell	Regsvr32	WMI	At.exe	at.exe
HTA files,	JScript	Regasm	Certutil	Schtasks	schtasks.exe
JavaScript	HTA	Regsvcs	Windows API	Net	services.exe
	.cpl	XSL Script Processing	Commonly Used Port	Windows API	cmd
	.lnk	InstallUtil	Standard Application	COM	sc.exe
	.sct	New Service	Layer Protocol	DCOM	net.exe
	Java	Shortcut Modification	Data Encoding	NTFS file attributes	Scheduled Tasks
	COM	Office Application	Network Stack	BITS Jobs	System Services
	DCOM	Startup	Windows Registry	Scheduled Transfer	
	Browser	at.exe		Pass the Hash	
	Email Client	schtasks.exe		Valid Accounts	
	Document	services.exe		File System	
	Reader	sc.exe		Network Stack	
	Windows	net.exe		Windows Registry	
	Kernel	WMI		User Account Control	
	DLL libraries	File System		Windows Kernel	
	EXE files	Windows Registry		Windows Services	
	WMI	Windows Services		Valid User Accounts	
		Scheduled Tasks			
		System Services			

Рисунок 1.1 Компоненти ОС Windows в рамках Kill Chain

Reconnaissance (розвідка) – зловмисник досліджує цільову систему використовуючи активні засоби, а саме мережеві сканери такі як nmap. В даному випадку ці засоби виступають в ролі інструментів атаки. За допомогою них отримується інформація про версію операційної системи, встановлене на ній ПЗ яке має доступ в мережу, версію цього ПЗ, що важливо в контексті пошуку потенційних вразливостей які можна експлуатувати для отримання початкового доступу, наявність та тип рішення безпеки, що захищає цільову систему та конфігураційні налаштування. Також зловмисник може використовувати пасивні інструменти

аналізу мережевого трафіку – сніфери мережевих пакетів такі як WireShark для дослідження вмісту цих пакетів та отримання додаткової інформації про складові цільової системи, що навіть не мають прямого доступу в мережу.

Weaponization (підготовка) – після отримання початкових даних та їх аналізу зловмисник здійснює підготовку до атаки. Знаючи версію та конфігурацію операційної системи він може наперед визначити які інструменти йому потрібно використовувати. Маючи інформацію про тип безпекового рішення системи жертви, зловмисник може підняти лабораторне середовище для створення модифікацій шкідливого ПЗ яке не буде виявлене цим рішенням.

Delivery (доставка) – зловмисник повинен отримати початковий доступ до системи жертви. Для цього є два основних підходи.

- Виконати Bind Shell , тобто експлуатувати виявлену на першому етапі вразливість ПЗ, що має доступ до мережі.
- Виконати Revers Shell, тобто відправити шкідливе ПЗ на систему жертви електронною поштою, месенджером, спонукати користувача системи самостійно завантажити шкідливе ПЗ з інтернет ресурсу або використовуючи змінні носії такі як USB – флешки, щоб після запуску шкідливого ПЗ користувачем отримати зворотною оболонку.

Exploitation (зараження) – доставлене на попередньому етапі шкідливе ПЗ необхідно запустити, щоб воно виконало закладені в нього функції.

Надалі розглянемо компонент операційної системи, його призначення та використання на різних етапах атаки.

- PowerShell – розширений кросплатформний інтерпретатор командної оболонки та середовище виконання сценаріїв, що вбудований в ОС Windows.

На етапі Exploitation використовується для завантаження та запуску шкідливого ПЗ, тобто не обов’язкового одразу надсилати шкідливе ПЗ на кінцеву точку. Можна відправити скрипт PowerShell, який після виконання завантажить та запустить основне шкідливе ПЗ. На етапі Installation (Встановлення) використовується для закріплення шкідливого ПЗ в системі та ухилення від систем захисту. На етапі Command and Control має широкий спектр використання, а саме дослідження системи

з середини, керування, підняття привілеїв. На етапі Actions on Objectives (Керування та контроль) використовується для викрадення облікових даних у вигляді хешів паролів, завантаження додаткового інструментарію (хакерських інструментів), а також для переміщення по мережі. Останній етап – Execution (Виконання). Отримавши права адміністратора або системи зловмисник може використати повний функціонал PowerShell для виконання будь-яких дій в системі.

- Component Object Model (COM) та Distributed Component Object Model (DCOM) – це механізм міжпроцесорної взаємодії IPC (inter-process communication), що в свою чергу входять до Native Windows API та використовуються для взаємодії з іншими об’єктами операційної системи, а саме DLL бібліотеками та EXE файлами.

На етапі Exploitation можуть бути використані для виконання коду, а саме запуску шкідливих бінарних файлів та скриптів. На етапі Installation через COM та DCOM можуть бути використані для обходу обмежень безпеки та здійснити взаємодію з іншими компонентами ОС. На етапі Command and Control зловмисник може використати DCOM через який здійснюється взаємодія COM об’єктів на різних системах через мережу для виконання шкідливого ПЗ віддалено. На останніх етапах - Actions on Objectives та Execution COM та DCOM об’єкти можуть бути використані для підвищення привілеїв.

- Windows Management Instrumentation (WMI) – це функція для забезпечення єдиного середовища доступу до системних компонентів ОС Windows. Її можна використовувати як локально так і через мережу 135 порт.

На етапах Delivery та Exploitation WMI може бути використано для віддаленого виконання шкідливого коду. На етапі Installation зловмисник може здійснити закріплення та ухилення від виявлення використовуючи постійні підписки на події WMI, що будуть виконуватися тільки при виконанні певних умов. На етапі Command and Control може використовуватися для віддаленого адміністрування. На етапах Actions om Objectives та Execution можна підняти привілеї в процесів експлуатації вразливостей або помилкової конфігурації WMI.

- User Account Control (UAC) – досить важливий компонент Windows через який здійснюється контроль за змінами системи або надання функціоналу, що здатен

змінювати систему без повідомлення користувача та отримання підтвердження. Цей механізм одночасно є механізмом як захистом так і вразливістю, оскільки у випадку отримання облікових даних та паролю адміністратора зловмисник зможе змінювати систему та не піддаватися контролю та моніторингу системами захисту. Також у випадку, якщо програма має автоматичне ухвалення UAC без потреби підтвердження зловмисник зможе запускати шкідливе ПЗ з правами адміністратора.

На етапах Command and Control та Actions and Objects компрометація UAC дозволить виконати будь-які дії в системі.

- Microsoft Office це пакет програм для роботи з текстами, презентаціями, електронними таблицями для виконання офісних завдань. На етапах Delivery, Installation та Execution для зловмисника є цікавим функціонал таких програм як підтримка VBA macros (мікрокоманди). Вони використовуються для автоматизації багатьох задач при роботі з документами в системі. Проте зловмисник може ними скористатися, щоб доставити шкідливе ПЗ в документі Office, та виконати його.

На етапі Command and Control макроси можуть бути використані ухилення від систем захисту. Також на цьому етапі може бути використана техніка DDE (Dynamic Data Exchange) для виконання шкідливого ПЗ та скриптів управління.

- Dynamic Link Library (DLL) – це дуже важлива складова ОС Windows яка надає можливість використовувати вбудовані функції (набори інструкцій) системи для декількох застосунків одночасно в процесі їх виконання. Це досить великий вектор атаки.

На етапах Installation, Execution та Command and Control - DLL бібліотеки можуть використовуватися для виконання шкідливого коду. Це відбувається шляхом підміни або втручання в систему DLL бібліотеку. Після чого в процесі запуску легітимного процесу буде виконана шкідлива інструкція, що здійснить зміни в системі, або запустить шкідливе ПЗ. Між етапами дана техніка відрізняється тільки з правами запуску і відповідним функціоналом, що зможе отримати зловмисник реалізувавши цю техніку.

- Net.exe – утиліта командного рядка, що використовується для управління мережею, виконання операцій з обліковими записами, службами та іншими компонентами ОС Windows.

На етапі Delivery через утиліту net.exe може бути використана для отримання початкового доступу до системи. На етапах Installation та Execution через net.exe можуть бути здійснені операції над службами. Наприклад вимкнення служби антивірусного захисту або запуск шкідливого ПЗ у вигляді служби. На етапі Command and Control через net.exe можна змінити мережеві налаштування та отримати можливість зв'язку через мережу з інфраструктурою атакуючого. На етапі Actions on Objectives зловмисник може використати всі можливості роботи net.exe з обліковими записами для збору інформації. Створенню нових облікових записів, видалення існуючих, маніпуляцій групами користувачів.

- NTFS (New Technology File System) – файлова система Microsoft, що прийшла на заміну сімейства FAT. Вона є посередником між диском та програмами та виконує ряд функцій таких як підтримка метаданих, файли та папки.

На етапах Delivery та Installation зловмисники можуть використовувати функції NTFS для приховування шкідливого ПЗ від систем захисту. Одним з таких методів є ADS (Alternative Data Streams), що першочергово були створені для сумісності з іншими файловими системами, що вимагають декілька потоків для зберігання даних. Записане шкідливе ПЗ в ці потоки не відображається, може підтримуватися будь-який розмір та тип даних. Завдяки цій функції досягається стійкість та ухилення від систем захисту. На етапі Actions on Objectives зловмисник може маніпулювати правами доступу NTFS для отримання доступу до захищених файлів та папок в яких зберігаються облікові дані та код складових ОС на яких зав'язано функціонування всієї системи.

- Реєстр – централізована база даних в якій зберігається конфігурація операційної системи, програм та обладнання.

На етапах Delivery та Exploitation зловмисники можуть використовувати вразливості програм, налаштування яких залежить вір реєстру для виконання шкідливого ПЗ та ухилення від систем захисту. На етапі Installation зловмисники

можуть змінювати реєстр, додаючи нові ключі та змінюючи існуючі для забезпечення стійкості в захопленій системі. На етапі Command and Control при можливості модифікувати реєстр можливо змінити налаштування систем безпеки, отримати нові можливості, підняти привілеї та змінити конфігурації мережі, для отримання стабільного зв'язку зараженої машини з сервером управління зловмисника. Також на цьому етапі зловмисник може провести пошук по гілкам реєстру важливої інформації про конфігурацію системи та навіть знайти облікові дані. Яскравий приклад це використання системи віртуалізації Proxmox, на якій ОС Windows розгорнута як віртуальна машина. В процесі інсталяції перший введений пароль зберігається у відкритому вигляді в реєстрі. Якщо він не був змінений. то зловмисник, скориставшись інструментами автоматизації для пошуку по ключовим словам може отримати пароль адміністратора і як наслідок повний контроль над системою. На етапі Actions on Objectives маючи дозволи на зміни всіх гілок реєстру зловмисник може здійснювати будь-які зміни конфігурації системи.

- Драйвери – це програми, завдяки яким операційна система може взаємодіяти з апаратним забезпеченням. Вони працюють на рівні ядра операційної системи та мають високій рівень доступу до ресурсів системи.

На етапах Delivery та Exploitation зловмисники можуть використовувати вразливості драйверів для запуску шкідливого ПЗ, встановити шкідливі драйвери або здійснити модифікації існуючих. Оскільки драйвери працюють на рівні ядра операційної системи, на етапі Installation закріплення на рівні ядра через драйвери є більш стійкою та менш помітною для систем захисту. На етапах Command and Control та Actions on Objectives зловмисник може надсилати команди через вразливий або фальшивий драйвер, для ухилення від систем захисту, та отримання повного контролю над системою.

- Scheduled Tasks (Заплановані завдання) – це технологія автоматизації запуск сценаріїв та програм в конкретний час або за дотримання визначених умов. В ОС Windows дана технологія реалізовується двома утилітами at.exe та schtasks.exe.

На етапах Delivery та Exploitation зловмисники можуть маніпулювати запланованими завданнями для запуску шкідливого ПЗ.

Розглянемо сценарій, в якому зловмисне ПЗ, що складається з двох функціональних частин. Після доставки такого ПЗ на машину та запуску користувачем виконується перша частина такого ПЗ, що взаємодіє з компонентами at або schtasks та створює заплановану задачу на запуск другої частини в якій закладено головний функціонал шкідливого ПЗ після перезавантаження системи.

На етапі Installation зловмисники можуть скористатися механізмом запланованих задач для досягнення стійкості, та збереження доступу до зараженої системи після її перезавантажень. На етапі Command and Control заплановані задачі можуть бути використані для віддаленого виконання команд, або завантаження додаткового інструментарію, вимкнення систем захисту та підвищення привілеїв. На етапі Actions on Objectives з достатніми правами використання запланованих задач може надати можливість змінювати налаштування системи та виконання шкідливого коду, наприклад запуск шифрувальника.

- Win32API - це набір функцій для розробки додатків в ОС Windows. Завдяки цим функціям додатки можуть взаємодіяти з компонентами операційної системи, взаємодіяти з процесами та мережею.

На етапі Delivery завдяки Win32API можна завантажити шкідливе ПЗ, що маскується під додатки Windows. На етапах Exploitation та Installation завдяки Win32API можна запустити шкідливе ПЗ в обхід систем захисту. На етапі Command and Control використовується можливість взаємодії з мережею для встановлення зв’язку с сервером управління зловмисника. На етапі Actions on Objectives з правами адміністратора зловмисник може здійснити модифікацію, викрадення або шифрування даних.

Це не всі складові ОС Windows які варто розглядати в контексті атаки. І не можна розглядати їх окремо, оскільки функціонал кожного з них розкривається на різних етапах з різними можливостями в рамках прав з якими з ними відбувається взаємодія.

Окремо слід розглянути розміщення цих компонентів відносно рівнів операційної системи, а саме рівня ядра та рівня користувача.

Рівень користувача – тут виконуються всі програми, що запущені користувачем. Цей рівень також містить деякі частини операційної системи, які взаємодіють безпосередньо з програмами користувача, наприклад, Windows API, драйвери рівня користувача та інтерфейси користувача. На цьому рівні програми мають обмежений доступ до системних ресурсів та не можуть безпосередньо взаємодіяти з обладнанням. Вони взаємодіють з ядром операційної системи через встановлений набір системних викликів (API). Рівень користувача, що містить більшість застосунків та послуг, є на першій лінії взаємодії з кінцевим користувачем. Це робить його більш вразливим до атак шкідливого ПЗ

Рівень ядра (Kernel Mode) – найнижчий і найбільш привілейований рівень операційної системи. Компоненти, що працюють на цьому рівні мають повний контроль над всіма ресурсами системи, мають безпосередній рівень до апаратного забезпечення та можуть виконувати будь-які інструкції машинного коду. До нього входять такі складові як драйвери, системні служби, файлова система яка є посередником між програмою та фізичним пристроєм довгострокової пам’яті, та саме ядро операційної системи. Шкідливе ПЗ, що може отримати доступ до рівня ядра, може отримати повний контроль над системою.

Компоненти ОС у контексті рівнів користувача та ядра операційної системи зображені на рисунку 1.2.

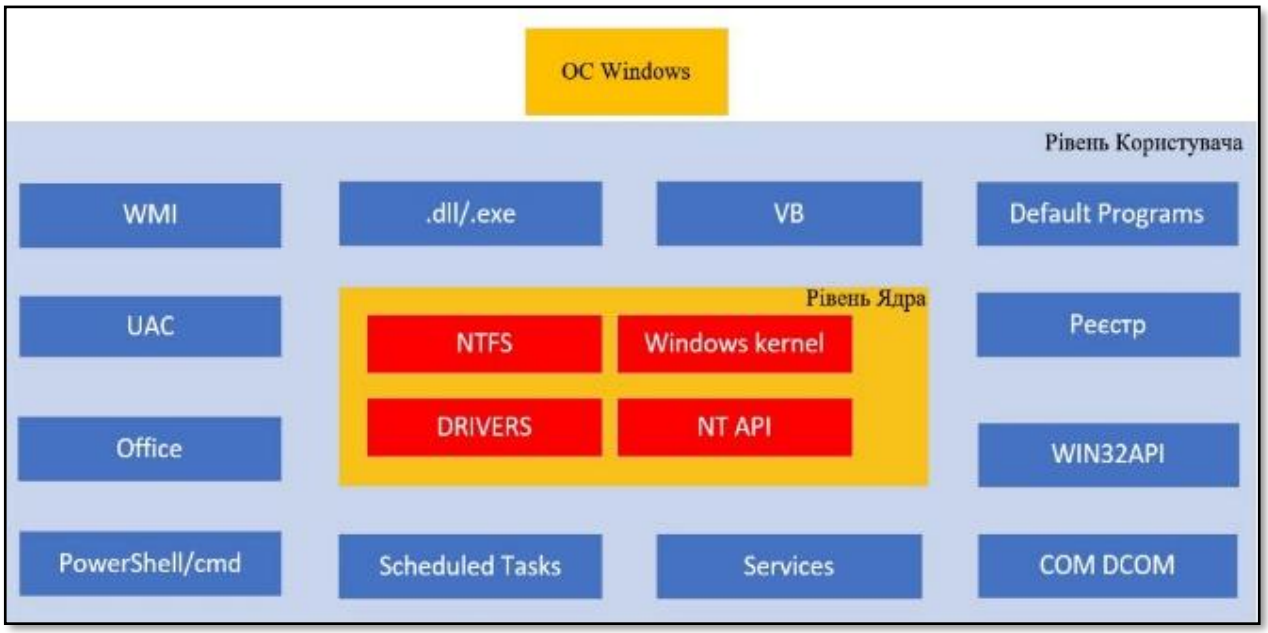


Рисунок 1.2 - Схема компонентів ОС Windows по рівнях

1.5 Джерела маркерів компрометації та поведінкових сигнатур в ОС Windows

В ОС Windows передбачені вбудовані джерела подій маркерів компрометації та поведінкових сигнатур завдяки яким можна здійснювати моніторинг за складовими на різних рівнях системи. Основними є наступні джерела подій:

Windows Events (Події Windows): Це джерело подій включає різні категорії, що пов'язані з безпекою системи, станом системи та діями застосунків. Ці події реєструються в системних журналах подій які називаються System, Security, Application – три основних журнали.

C:\Windows\System32\winevt\Logs. В цій частині файлової системи зберігаються всі файли журналів з розширенням .evtx, кожен з яких відповідає за моніторинг груп компонентів операційної системи.

Взаємодія з ними відбувається через утиліту EventViewer, яка є складовою mms.exe, що зображено на рисунку 1.3.

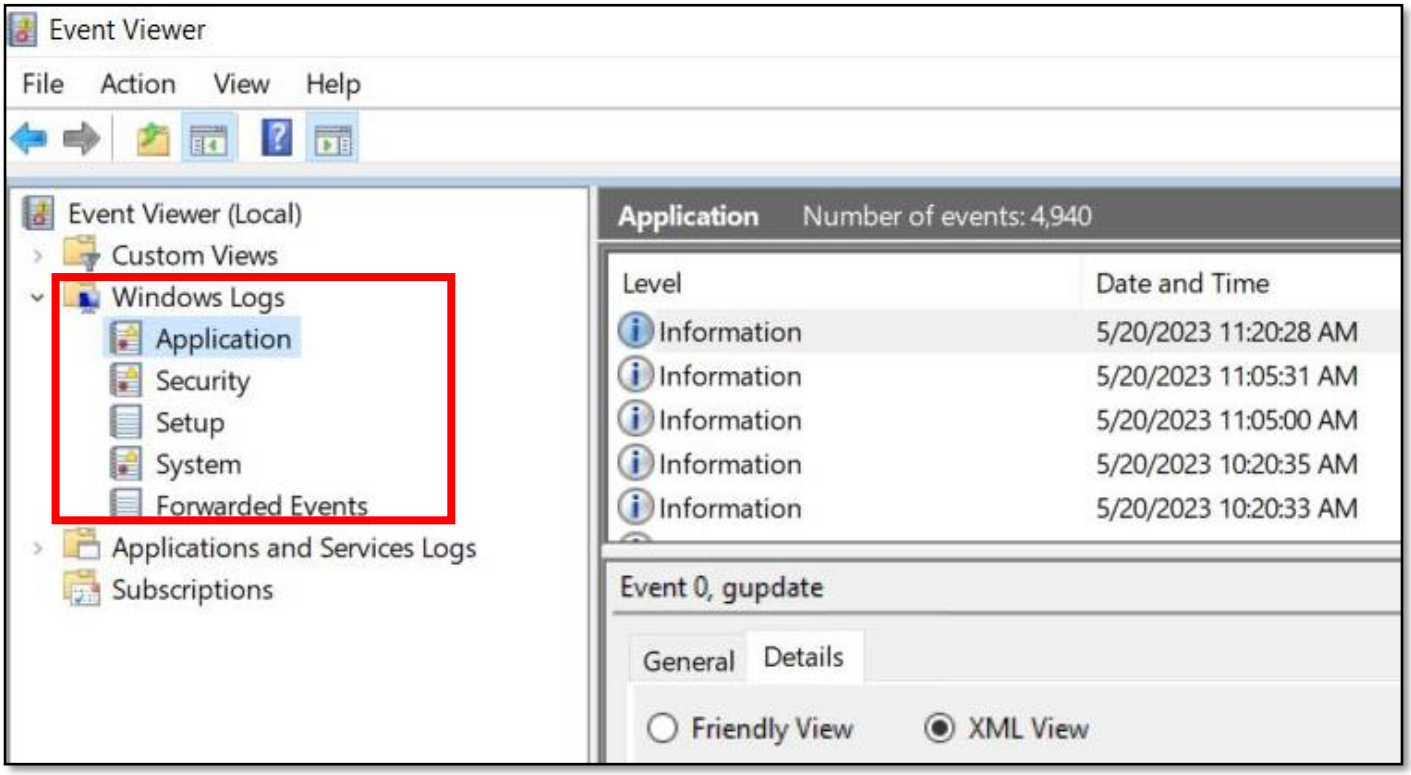


Рисунок 1.3 Event Viewer

• Windows Defender (Події антивірусного захисту Windows Defender). Тут зберігаються події Windows Defender. Вони реєструються в системному журналі подій Microsoft-Windows-Windows Defender/Operational.evtx.

• Windows Firewall with Advanced Security Events (Події додаткової захисту Windows Firewall). Це джерело подій відстежує активність фаєрвола Windows, включаючи блокування з'єднань, атаки, винятки, правила безпеки та інші події, пов'язані з мережевим захистом. Вони реєструються журналі подій Microsoft-Windows-Windows Firewall With Advanced Security/Firewall.

• Microsoft-Windows-Sysmon (Події системного моніторингу Microsoft). Це джерело подій пов'язане зі встановленою утилітою Sysmon, яка надає розширену функціональність моніторингу системи. Вони реєструються журналі подій Microsoft-Windows-Sysmon.

1.5.1 Журнал подій системи ОС Windows

Журнал подій System в ОС Windows містить важливу інформацію про стан та функціонування операційної системи. В цей журнал збираються події, що відображають наступні взаємодії з компонентами операційної системи та самою системою такі як завантаження та вимкнення системи, встановлення та видалення програм, потенційні проблеми системи такі як попередження про заповнення дискового простору, повідомлення про стан апаратних складових системи, а це відповідно моніторинг драйверів, моніторинг системних служб та оновлення системи. Тобто це моніторинг рівня ядра операційної системи та його складових.

Журнал подій системи System Event зберігається в C:\Windows\System32\winevt\Logs\System.evtx

Його інтерфейс та синтаксис xml-документу події мають вигляд зображений на рисунку 1.4.

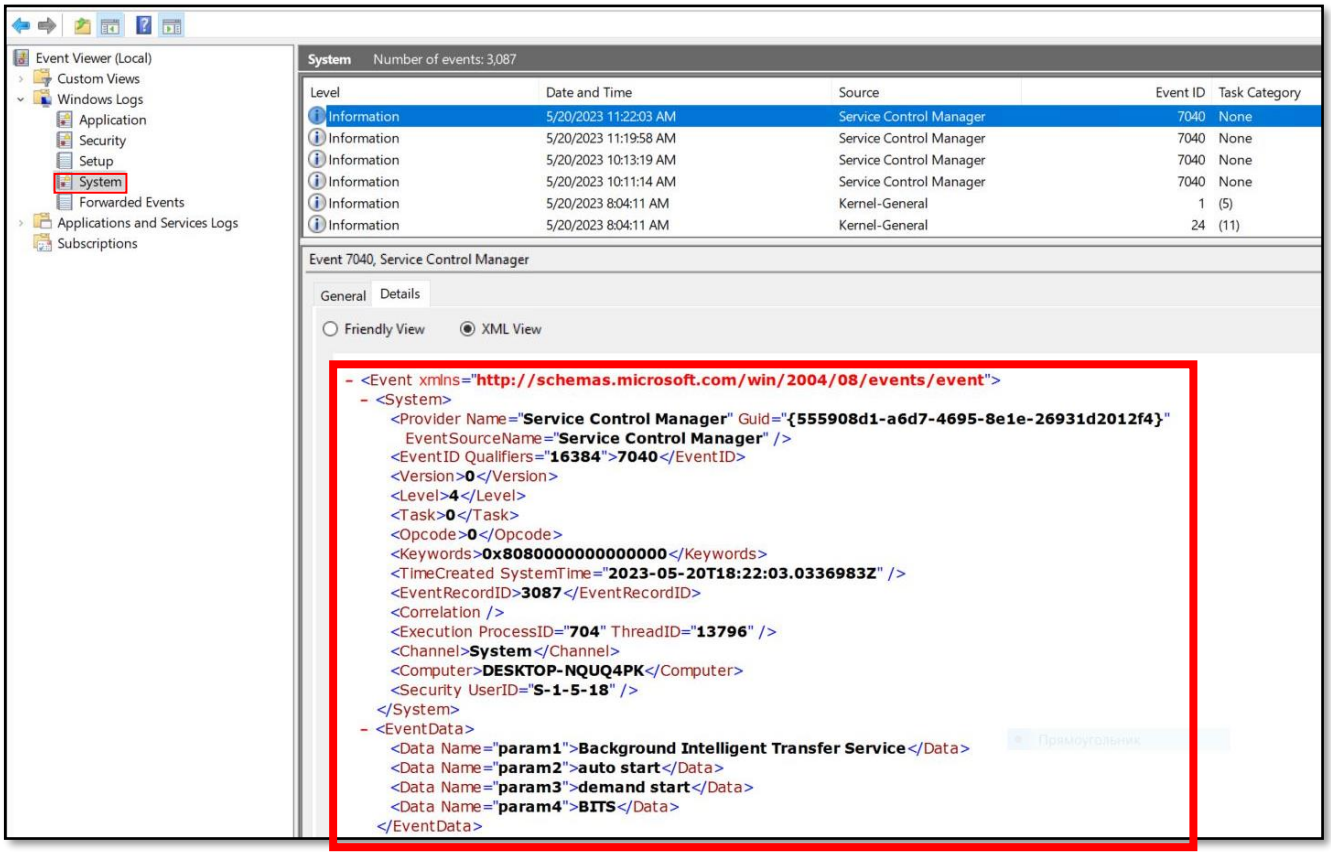


Рисунок 1.4 – xml відображення логу System.evtx

Деякі з найпоширеніших кодів подій та відповідні типи подій, які можна знайти в журналі системи ОС Windows, наведені в таблиці 1.2.

Таблиця 1.2

Коди подій журналу System.evtx

Код події	Опис
Event ID 6005	Журнал подій стартований: Цей код події вказує на те, що журнал подій був успішно запущений після запуску або перезавантаження системи.
Event ID 6006	Журнал подій зупинений: Цей код події вказує на те, що журнал подій був зупинений, зазвичай це відбувається при вимкненні системи.
Event ID 6008	Неочікуване вимкнення: Цей код події вказує на те, що система була несподівано вимкнена, що може вказувати на проблеми з живленням або апаратними збоями.
Event ID 7026	Невдала ініціалізація драйвера: Цей код події вказує на те, що один або кілька драйверів не змогли ініціалізуватися при запуску системи.

Практична користь журналу подій System ОС Windows для моніторингу компонентів ОС, що працюють на рівні ядра , а також використовуючи інформацію з цього журналу можна встановити причини помилок з апаратним забезпеченням та відновити нормальну роботу системи.

1.5.2 Журнал подій безпеки ОС Windows

Журнал подій Security в ОС Windows реєструє всі дії над обліковими записами. А саме вдалі та невдалі входи в систему, створення та видалення облікових записів, зміни груп користувачів, та всі виклики UAC). Журнал подій Security зберігається в C:\Windows\System32\winevt\Logs\Security.evtx

Його інтерфейс та синтаксис xml-документу зображені на рисунку1.5.

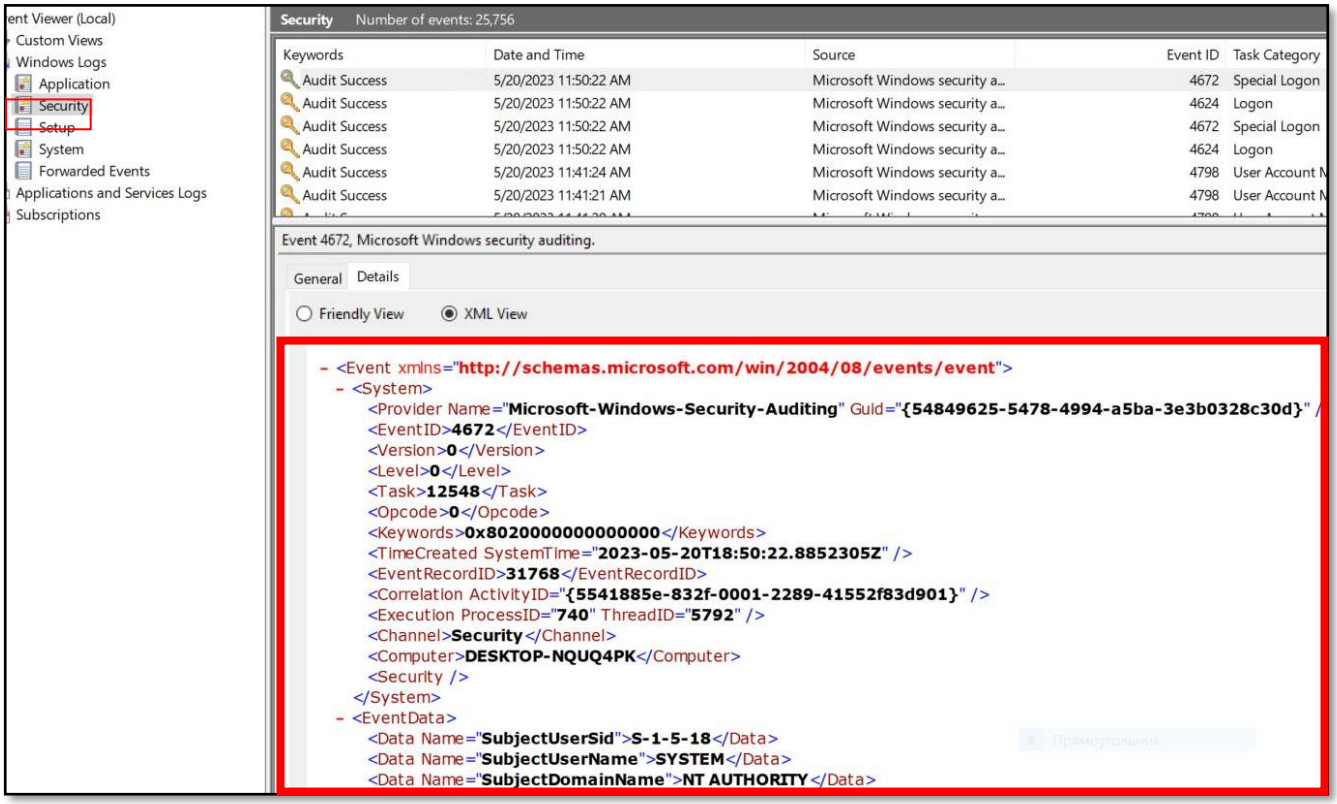


Рисунок 1.5 – xml відображення логу Security.evtx

Додатково потрібно вказати, що журнал Security також може збирати інформацію про доступ до об’єктів операційної системи таких як реєстр, файли,

пристрої, створення та припинення процесів. Цей функціонал за замовчуванням вимкнутий.

Деякі з найпоширеніших кодів подій та відповідні їм описи, що зберігаються в журналі безпеки ОС Windows, представлені в таблиці 1.3.

Таблиця 1.3

Коди подій журналу Security.evtx

Код події	Опис
Event ID 4624	Успішна аутентифікація. Цей код події вказує на успішну аутентифікацію користувача в системі.
Event ID 4625	Невдала аутентифікація. Цей код події вказує на невдалу спробу аутентифікації користувача в системі.
Event ID 4634/4647	Відключення користувача. Цей код події вказує на відключення користувача від системи.
Event ID 4648	Запит на вищий рівень аутентифікації. Цей код події вказує на запит користувача на вищий рівень аутентифікації, наприклад, запит на виконання певних привілеїв.
Event ID 4688	Створення нового процесу. Цей код події вказує на створення нового процесу в системі.
Event ID 4697	Зміна привілеїв. Цей код події вказує на зміну привілеїв користувача або групи.
Event ID 4719	Зміна політики обліку довіреної комп'ютерної мережі. Цей код події вказує на зміну політики обліку довіреної комп'ютерної мережі.
Event ID 4798	Зміна політики об'єкту безпеки. Цей код події вказує на зміну політики об'єкту безпеки, такого як об'єкт безпеки файлу або папки.
Event ID 5156	Зміна мережевої політики. Цей код події вказує на зміну мережевої політики, яка визначає правила доступу до мережевих ресурсів.
Event ID 6281	Невідповідність хешів. Цей код події вказує на помилку підпису файлу, що свідчить про внесення несанкціонованих змін.

Ці коди подій та відповідні типи подій у журналі безпеки ОС Windows допомагають ідентифікувати безпекові інциденти, відстежувати активність користувачів, виявляти незвичайну поведінку та забезпечувати безпеку системи.

1.5.3 Журнал подій застосунків ОС Windows

Журнал подій Application в операційній системі Windows реєструє події пов'язані з виконанням програмного забезпечення, встановленого на системі. Цей журнал містить інформацію про події, що відбуваються в застосунках, службах Windows або компонентах DLL, які використовуються цими застосунками.

Журнал подій застосунків Application Event зберігається в C:\Windows\System32\winevt\Logs\Application.evtx

Його інтерфейс та синтаксис xml-документу мають наступний вигляд (рисунок 1.6).

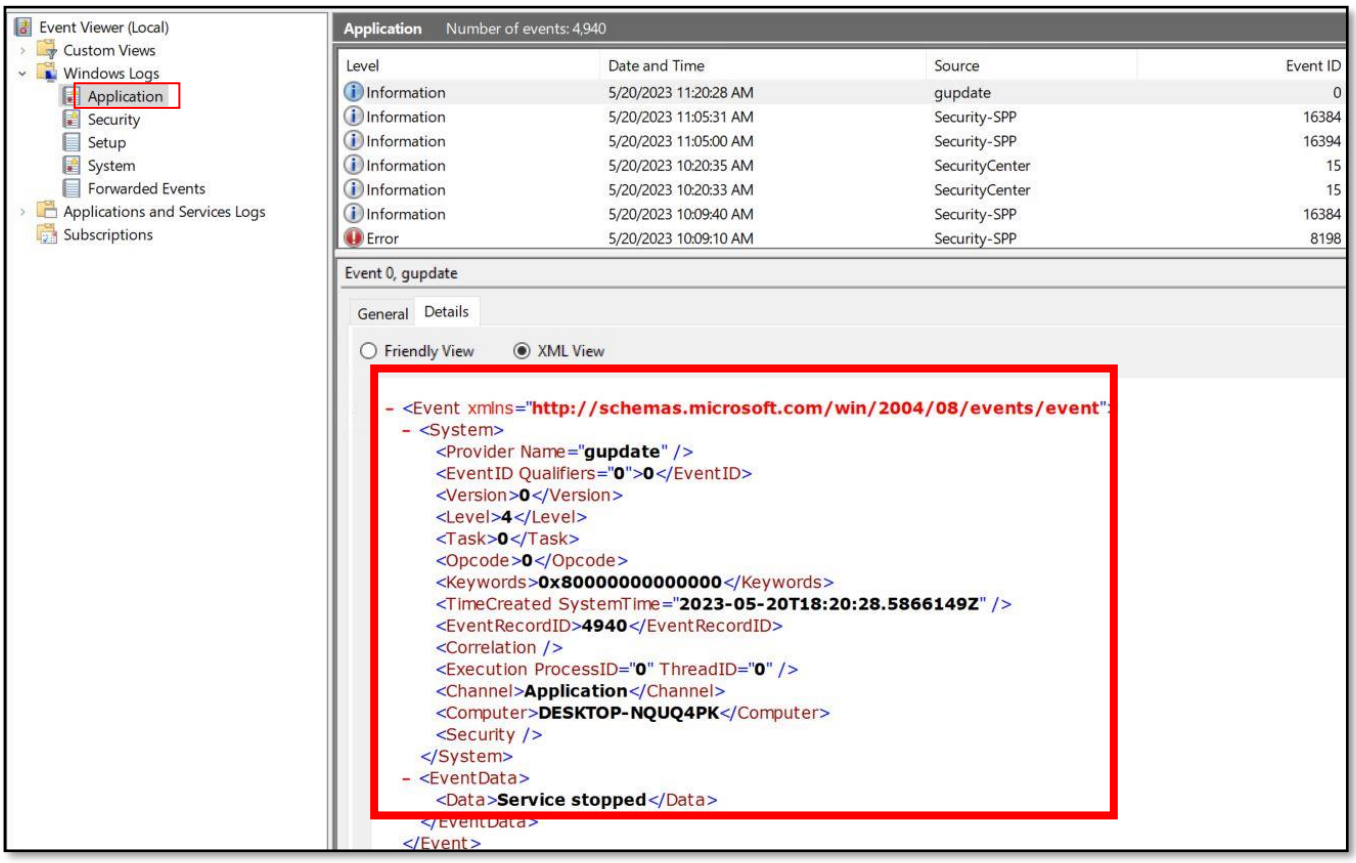


Рисунок 1.6 – xml відображення логу Application.evtx

Деякі з найпоширеніших кодів подій та відповідні їм описи, що зберігаються в журналі застосунків ОС Windows наведені в таблиці 1.4

Таблиця 1.4

Коди подій журналу Application.evtx

Код події	Опис
Event ID 1000	Збій застосунку: Цей код події вказує на те, що застосунок зазнав збою.
Event ID 1001	Звіт про збій: Цей код події вказує на те, що звіт про збій був відправлений в Microsoft.
Event ID 1002	Застосунок не відповідає: Цей код події вказує на те, що застосунок не відповідає на запити системи.
Event ID 1026	.NET Framework вийняток: Цей код події вказує на те, що сталася помилка в коді, написаному за допомогою .NET Framework.
Event ID 1309	Подія ASP.NET: Цей код події вказує на те, що відбулася помилка в застосунку, який використовує ASP.NET.

Журнал подій застосунків використовується для відстеження виконання програмного забезпечення, виявлення проблем, які можуть виникнути під час роботи застосунків, а також вирішення проблем зі стабільністю застосунків. В контексті виявлення шкідливого ПЗ завдяки цьому журналу можна виявити аномалії, ознаки зловживання DLL та нетипову поведінку застосунків.

1.5.4 Журнал подій Sysmon

Sysmon (System Monitor) це утиліта з пакету sysinternals - (в пакет входить 76 утиліт серед яких такі легенди як TCPView, ProcessMonitor PSExec, Process Explorer тощо). Розробник є Марк Русімович - відомий фахівець в галузі технологій і співзасновник компанії Winternals Software.

Sysmon на момент випуску мав відкритий код та позиціонувався як безкоштовний інструмент для моніторингу операційної системи Windows, завдяки якому можна здійснювати моніторинг на рівні ядра операційної системи.

У 2006 році компанія Microsoft придбала компанію Winternals Software, включаючи права на всі програми з пакету sysinternals. Також після придбання

компанією Microsoft sysmon припинив бути додатком з відкритим кодом. Тепер його розробкою та підтримкою займається компанія Microsoft.

Sysmon має дві складові – сервіс sysmon та драйвер Sysmon (саме завдяки драйверу здійснюється логування подій на рівні ядра операційної системи). Sysmon надає розширену функціональність моніторингу та аудиту подій безпеки. Він фіксує такі події в системі як запуск процесів, взаємодія з файловою системою, мережеві підключення, зміни в реєстрі, завантаження драйверів.

Sysmon веде окремий журнал подій, який зберігається в файловій системі в C:\Windows\System32\winevt\Logs\Microsoft-Windows-Sysmon\Operational.evtx

Синтаксис xml-документу мають наступний вигляд (рисунок 1.7).

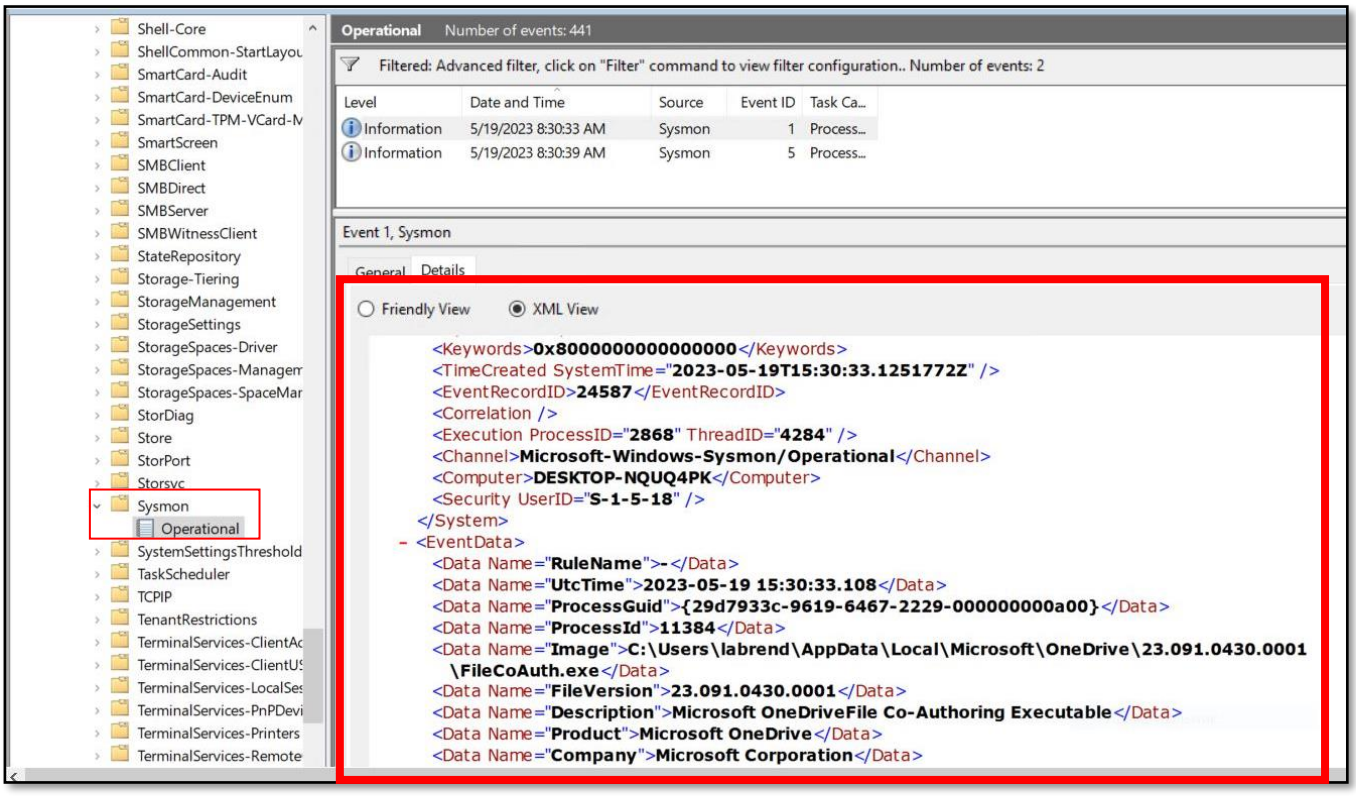


Рисунок 1.7 xml відображення логу в Sysmon\Operetional.evtx

Sysmon використовує власну класифікацію подій. Вони позначаються номерами від 1 до 23. Коди подій Sysmon та їх опис представлені у таблиці 1.5.

Таблиця 1.5

Класифікація подій sysmon

Код події	Опис
1	Process Create - створення процесу
2	File creation time - зміна часу створення файлу
3	Network connection - мережеве підключення
4	Sysmon service state changed - зміна стану Sysmon сервісу
5	Process terminated - завершення процесу
6	Driver loaded - завантаження драйвера
7	Image loaded - завантаження бінарного файлу в пам'ять
8	CreateRemoteThread - створення віддаленого потоку
9	RawAccessRead - читання диска на низькому рівні
10	ProcessAccess - доступ до процесу
11	FileCreate - створення файлу
12, 13, 14	RegistryEvent (Object create and delete, Value Set, Key and Value Rename) - зміни в реєстрі
15	FileCreateStreamHash - створення альтернативних потоків даних файлу
16	Sysmon configuration change - зміни в конфігурації Sysmon
17	Pipe Created - створення іменованого каналу
18	Pipe Connected - підключення до іменованого каналу
19	WmiEvent (WmiEventFilter activity detected) - активність WMI фільтрів
20	WmiEvent (WmiEventConsumer activity detected) - активність WMI споживачів
21	WmiEvent (WmiEventConsumerToFilter activity detected) - зв'язок між WMI фільтром та споживачем
22	DNS query - DNS-запити
23	File Delete - видалення файлів

Для кращого розуміння логіки логів sysmon доцільно провести категоризацію подій Sysmon. Перш за все треба звернути увагу на процеси, в sysmon є тільки дві дії з процесами (створення та завершення) це EventID 1 та 5, наступними є операції з об’єктами, їх два типи – драйвери та образи. Відповідні EventID 6 та 7. Наступні досить важливі логи це дії, що виконуються в рамках певного процесу, тобто завдяки Sysmon ми не тільки маємо можливість фіксувати процеси, а ще й отримувати окремі

логи опису операцій які відбуваються в рамках процесу. Тобто процес виступає в ролі контейнера, а вже в його рамках відбуваються дії - зміна часу створення файлу, створення файлів, операції з реєстром, створення потоків даних файлу, створення каналів та видалення файлів - EventID 2,11,12,13,14,15,17,23. Також окремо варто віднести дії між процесами. Це створення віддалених потоків та доступ до інших процесів - EventID 8 та 10. Мережева активність також відбувається в рамках процесів. Але варто віднести їх в окрему категорію, до неї відносяться EventID 3 та 22 відповідно. І залишилися тільки події безпеки та аудиту, до яких відносяться всі операції з Sysmon. А саме зміна конфігураційного файлу, зупинка драйвера Sysmon. А також в цю категорію відноситься використання WMI. Вся категоризація наведена в таблиці 1.6.

Таблиця 1.6

Категоризація подій sysmon

Категорія	Події(EventID)	Опис
Процеси	1, 5	Створення та завершення процесів
Завантаження об'єктів	6, 7	Завантаження драйверів та образів
Дії, що виконуються процесами	2, 11, 12, 13, 14, 15, 17, 23	Зміна часу створення файлу, створення файлів, зміни в реєстрі, створення потоків даних файлу, створення іменованих каналів, видалення файлів
Взаємодія між процесами	8, 10	Створення віддалених потоків, доступ до інших процесів
Мережева активність	3, 22	Мережеві підключення, DNS-запити
Події безпеки та аудиту	4, 16, 9, 18, 19, 20, 21	Зміни стану Sysmon сервісу, зміни в конфігурації Sysmon, читання диска на низькому рівні, підключення до іменованих каналів, активність WMI фільтрів та споживачів

1.5.5 Порівняння журналів

Моніторинг системних журналів - критичний аспект управління інформаційною безпекою. Журнали застосунків, безпеки та системи ведуть запис про ключові події в операційній системі, дозволяючи інженерам з інформаційної безпеки відстежувати системну активність, виявляти потенційні проблеми та розслідувати інциденти безпеки. Однак, існують обмеження класичних систем логування, які можуть перешкоджати ефективному виконанню цих завдань.

Одним з основних викликів є розсіяність даних. Журнали безпеки, застосунків та системи зберігаються в окремих файлах або на окремих серверах, що може ускладнити зіставлення подій та розуміння загальної картини. Це стає особливо складним, коли потрібно розуміти послідовність подій або виявляти шкідливе ПЗ за допомогою кореляції подій між різними журналами.

Крім того, класичні журнали часто надають обмежену інформацію про події. Вони можуть повідомити, що певна подія відбулась, але не надають достатньої деталізації про контекст цієї події. Наприклад, журнал може зафіксувати, що файл було видалено, але не вказати, яким процесом або користувачем ця дія була виконана. Або в журналі може бути зафіксовано створення нового користувача, але не вказано який саме процес і відповідно яка саме утиліта була використана для цього.

З цих причин, інструменти, такі як Sysmon, що надають більше деталей і зберігають інформацію в одному місці, стають все більш цінними в сучасному кіберпросторі. Sysmon зберігає деталізовану інформацію про події в системі, включаючи інформацію про процеси, мережеву активність, зміни в файлах та навіть окрему категорія про вплив на сам sysmon. Ця деталізована інформація може допомогти в розслідуванні інцидентів безпеки, виявленні шкідливих дій і розробці стратегій захисту.

Таким чином, порівняно з традиційними журналами, Sysmon надає більше деталізації та контексту для кожної системної події. З цією додатковою інформацією, аналітики безпеки можуть більш ефективно відстежувати активність в системі та розслідувати потенційні інциденти безпеки.

1.6 Огляд існуючих рішень для виявлення шкідливого ПЗ

На даному етапі розвитку інформаційних технологій вже є досить багато розробок призначення яких - захист від шкідливого ПЗ. Вони є як у відкритому доступі так і комерційними. Відповідно до функцій такі рішення поділяються на наступні види:

- Antivirus (Антивіруси) - це програми, що встановлюються на кінцеву точку та використовуються для виявлення , видалення та запобігання виконання шкідливого ПЗ. До такого типу рішень відносяться наступні марки: Symantec, McAfee, Avast, Bitdefender, Webroot.
- Sandbox (Пісочниця) – це віртуальна машина яка симулює кінцеву точку. При завантаженні будь-якого файлу з інтернету він спочатку потрапляє в пісочницю. Після чого вміст виконується в ізольованому середовищі щоб перевірити чи є контент шкідливим шляхом дослідження його поведінки. Сучасні пісочниці , наприклад від компанії CheckPoint здатні симулювати до 25 типів операційних систем, швидко проганяючи досліджуваний файл через кожну з них, тим самим не залишаючи шансів шкідливому ПЗ потрапити на кінцеву точку. Приклади таких рішень включають CheckPoint sandbox, FireEye Network Security, Cisco AMP for Endpoints.
- SIEM (Керування інформацією про безпеку) – це рішення для централізованого збору, обробки та візуалізації подій безпеки шляхом вичитки логів з відповідних журналів на контрольованих кінцевих точках. Прикладами такого рішення є ELK, WAZUH, IBM QRadar, Splunk, SolarWinds (SIEM), FortiSIEM,.
- EDR (виявлення загроз кінцевої точки) - це сукупність технології, яка виявляє підозрілу активність на контрольованій кінцевій точці після чого збирає додаткову інформацію про цю активність та відправляє зібрані дані до центрального сервера обробки для проведення додаткового дослідження. Після перевірки система або в ручному або в автоматичному режимі виконає відповідь на інцидент. До такого типу рішень відносяться наступні марки. Crowdstrike Falcon, Microsoft Defender, SentinelOne, FireEye Endpoint Security, Cybereason.

Розглянемо сильні та слабкі сторони таких рішень.

Слабкі сторони

- Недостатній аналіз поведінки. Рішення, що постачаються від розробників операційних систем, часто не блокують дії, що виглядають легітимно. Вони можуть не враховувати функціонал, передбачений самою системою, що може призвести до проходження шкідливого ПЗ через захисні бар'єри.
- Часті хибні спрацювання. Оскільки шкідливе ПЗ зазвичай використовує вбудований функціонал систем постає складне завдання розробки алгоритмів, що зможуть досить точно відрізнити шкідливі дії від легітимних. Через це відбуваються часті хибні спрацювання систем захисту які в свою чергу вимагають залучення відділу інформаційної безпеки.
- Блокування легітимних процесів. Іншою стороною недосконалості алгоритмів розрізнення є блокування легітимних процесів. Через це страждає користувач системи.
- Неврахування індивідуальних особливостей використання системи кінцевим користувачем. Зазвичай більшість рішень для виявлення шкідливого ПЗ не адаптують свої алгоритми під конкретну інфраструктуру або унікальну конфігурацію кінцевої точки.
- Велике споживання апаратних ресурсів. Деякі рішення для виявлення шкідливого ПЗ вимагають великої кількості ресурсів для своєї роботи, яскравим прикладом є пісочниця, оскільки на повну перевірку файлу потрібен час, через що сповільнюються бізнес-процеси.
- Недостатній рівень захисту від загроз нульового дня. Більшість антивірусних рішень і рішень для виявлення шкідливого ПЗ базуються на вже відомих сигнатурах, через що їх ефективність проти загроз нульового дня зводиться до нуля.
- Відомість алгоритмів та методів виявлення. Хакери не втрачають можливості адаптувати свої розробки під актуальні системи захисту. Вони створюють цілі лабораторії, та досліджують яким чином відбувається детектування шкідливого ПЗ конкретним безпековим рішенням. Далі вони аналізують отримані результати да вдосконалюють свої розробки

Сильні сторони

- Розширене детектування. Існуючі рішення вже давно мають напрацьовані методи та підходи для детектування найрізноманітніших типів шкідливого ПЗ , а саме черв'яків, троянських коней, вірусів шифрувальників та руткітів.
- Багаторівневий захист. Більшість сучасних рішень боротьби з шкідливим ПЗ використовують багаторівневий захист, який по суті об'єднує в рамках одного рішення поведінковий аналіз, евристичний аналіз, сигнатурний аналіз, а також аналіз мережевого трафіку для виявлення шкідливого ПЗ.
- Адаптація до нових загроз. Безпекові рішення не стоять на місці і постійно вдосконалюються. Розробники таких рішень створюють цілі лабораторії та досліджують шкідливе ПЗ з усього світу, виявляючи нові маркери детектування, які потім будуть адаптовані та додані до баз сигнатур при оновлені таких систем.

Висновки за розділом 1

В першому розділі було проведено аналіз типів шкідливого програмного забезпечення. А саме malware, що може діяти в автономному режимі за попередньо визначеним сценарієм та інструментів, що вимагають безпосереднього отримання команд від зловмисника.

Розглянуто методи дослідження шкідливого ПЗ, основними з яких є поведінковий та сигнатурний аналіз. Розглянуто поняття маркерів компрометації. Також було висвітлено, які компоненти ОС Windows стають цілями атаки та на яких етапах в рамках методології Kill Chain. Зловмисники використають будь-який функціонал системи для її компрометації навіть такий, що раніше був невідомий і відповідно це не буде виявлено рішеннями безпеки. Не має значення як способом це реалізовано, важливо до якого компонента системи зловмисник хоче отримати доступ та які функції від цього компонента йому потрібні для досягнення мети атаки, тобто методи змінюються, а цілі залишаються ті самі.

Після детального огляду складових операційної системи було розглянуто вбудовані журнали подій ОС Windows, їх призначення, переваги та недоліки.

Додаткову увагу варто приділити утиліті Sysmon завдяки якій можна збирати логи про взаємодію та використання всіх компонентів ОС Windows в уніфікованому вигляді та з високим рівнем деталізації.

Також окрему увагу потрібно приділити рішенням безпеки, оскільки з одного боку це серйозні комерційні рішення з великим досвідом, але з іншого вони зазвичай дають тільки спрацювання в автоматичному режимі з обмеженим оглядом інциденту. Ніхто не захистить свою систему краще ніж її власник. Досить часто бізнес та користувачі просто закривають питання захисту без розуміння власної системи. Саме така політика часто і робить їх вразливими до атак, оскільки безпекове рішення повинно виконувати виключно допоміжну функцію в неперервному процесі забезпечення безпеки від атак, що здійснюються в тому числі з використанням шкідливого ПЗ.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНУГО МОДУЛЯ

2.1 Вимоги до програмного модуля

Перед розробкою програмного модуля виявлення шкідливого ПЗ на основі поведінкового аналізу першим кроком необхідно сформувати ряд обов’язкових функціональних вимог.

- Здійснення моніторингу в режимі реального часу. Тобто необхідно підписатися на джерела подій і отримувати їх напряму, а не вичитувати файли журналів.
- Робота на рівні ядра ОС. Нам не потрібно здійснювати моніторинг кожного компонента операційної системи окремо і потім зіставляти дані у різних форматах, достатньо встановити логування на рівні ядра операційної системи. А саме перед тим як інтерпретована операція потрапить в ядро на виконання. З цим також впорається sysmon, який працює на рівні ядра ОС.
- Незалежність від налаштувань системи. Модуль повинен забезпечувати незалежний від налаштувань системи моніторинг. Це означає, що він повинен здатен працювати точно та ефективно незалежно від використовуваних конфігурацій системи.
- Єдиний формат даних. Модуль повинен подавати дані в єдиному форматі, що в свою чергу спрощує їх аналіз та обробку. Саме для цього і буде використовуватися sysmon оскільки він надає інформацію про події в операційної системи в єдиному форматі.
- Логіка об'єднання логів в інцидент. Взаємодіяти з логами системи доступна і без програмного модуля, використовуючи Event Viewer. Проте в журналі подій всі логи розпорошені, а зіставлення їх у групи вручну перетвориться в непосильну задачу. Саме тому у модулі обов’язково повинен бути передбачений алгоритм, що автоматично згрупує логи у відповідні групи з урахуванням належності до дій, що відбуваються в рамках конкретної активності та час протягом якого це відбувається.

• База сигнатур. Обов’язково необхідно передбачити наявність бази сигнатур, відповідно до яких буде здійснюватися детектування шкідливого ПЗ виходячи з його поведінки яка відображається в логах у вигляді полів, які в свою чергу будуть порівнюватися з тими, що знаходяться в цій базі. Також така база дозволить користувачу самостійно вказати, які події необхідно відслідковувати, а які події можна ігнорувати, завдяки чому модуль буде споживати менше системних ресурсів та враховувати специфічні побажання користувача системи. Завдяки цьому можна здійснювати обмін базами поведінкових сигнатур між користувачами такого модуля на різних системах.

• Зручний інтерфейс. Програмний інтерфейс повинен буди зрозумілим та простим у використанні. Це можна досягти завдяки графічним елементам з якими можна взаємодіяти та отримувати додаткову інформацію.

•Використання рідної для ОС Windows мови програмування. Модуль повинен бути написаний на мові програмування, яка є рідною для ОС Windows. Це дозволяє забезпечити найкращу сумісність та оптимальну продуктивність.

2.2 Розробка архітектури програмного модуля.

Архітектура програмного модуля включає такі ключові компоненти:

• Sysmon виступає як первинне джерело системних подій, надаючи деталізовану інформацію про події на рівні ядра ОС.

• Конфігураційний файл XML Sysmon використовується в ролі кастомізованого фільтра та як база сигнатур для відстеження поведінки шкідливого ПЗ.

• Event Logs, а саме sysmon/Operetional який виконує функції бази даних логів, що були згенеровані програмою Sysmon при фільтрації з використанням конфігураційного файлу.

• Програмний модуль. Він створений на мові програмування C#, “підписується” на події Sysmon, отримує їх в режимі реального часу, та відсортовує

їх по групам відповідно до дій, що відбуваються в рамках процесів взаємодії з компонентами системи.

- Інтерфейс для візуалізації. Логи можна переглядати і через вбудовані інструменти ОС Windows – Event Viewer, але для зручності аналітика та відповідно підвищення розуміння контексту подій в рамках системи необхідно забезпечити візуалізацію подій

Кожен з цих компонентів виконує специфічну роль, в сукупності складаючи збалансований та надійний системний моніторинг. Розроблений модуль працює з логами подій Sysmon, використовуючи вибіркові дані з конфігураційного файлу XML для фільтрації значущих подій та їх підпису. Ці події зберігаються в Event Logs, що служать як динамічна база даних для системних логів.

Схема архітектури додатку представлена на рисунку 2.1.

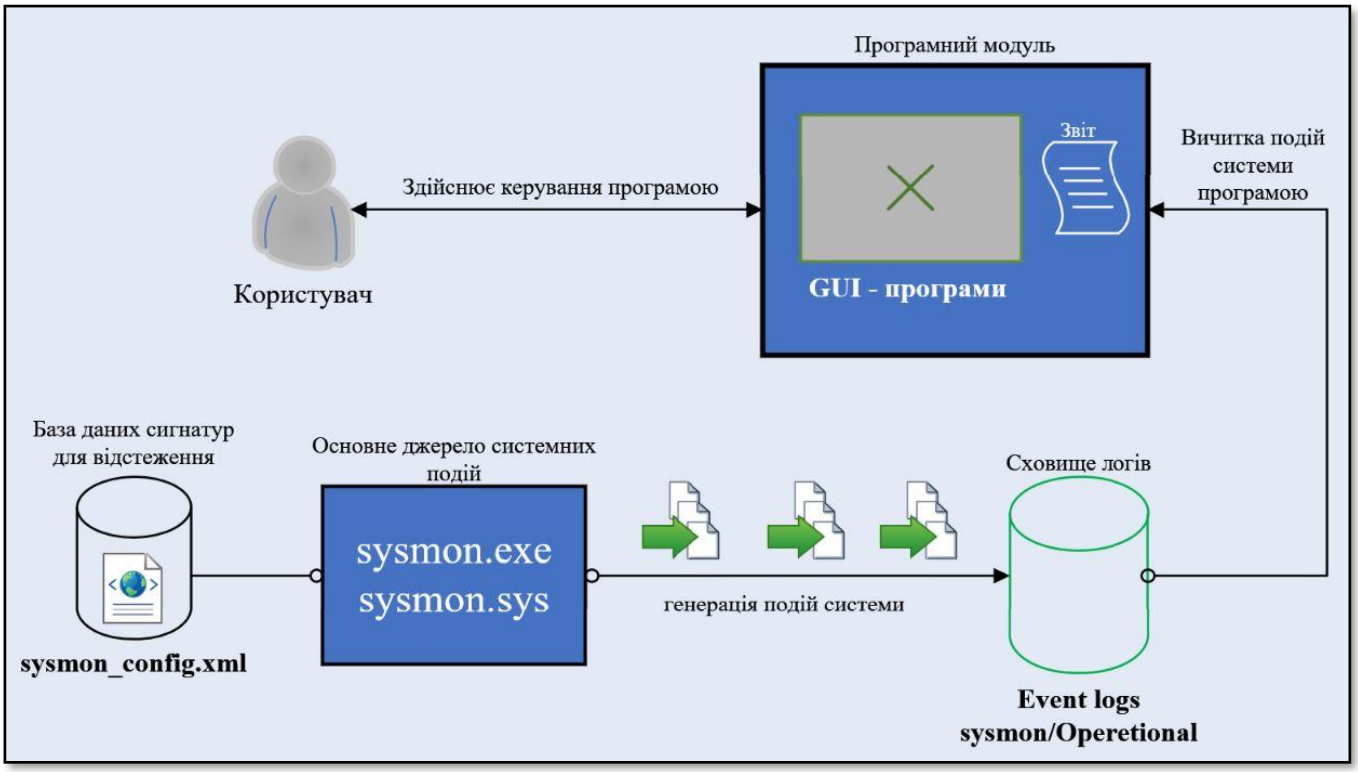


Рисунок 2.1 - Схема архітектури додатку

Програмний модуль вичитує події з журналу Sysmon\Operetional в режимі реального часу та формує актуальний звіт.

2.3 Обґрунтування вибору C# як мови для реалізації модуля

Для створення програмного модуля була обрана мова програмування C#

Підтримка платформи .NET: C# є основною мовою платформи .NET, що робить його ідеальним для розробки застосунків у середовищі Windows. Платформа .NET надає багатий набір бібліотек та інструментів, що дозволяють працювати з системними подіями та журналами подій Windows.

Зручність розробки та підтримки: C# - це мова з сильною статичною типізацією, що сприяє створенню чистого, зрозумілого та надійного коду. Вона також має ряд сучасних особливостей, що спрощують процес розробки та підтримки.

Порівняння мов програмування для реалізації програмного модуля виявлення шкідливого ПЗ в середовищі ОС Windows наведено у таблиці 2.1.

Таблиця 2.1

Порівняльна таблиця мов програмування

Параметри/Мова програмування	C#	Python	Java	JavaScript	Go	Rust
Підтримка платформи .NET	+	-	-	-	-	-
Інтеграція з Windows	+	-	+/-	+/-	+/-	-
Зручність розробки та підтримки	+	+	+	+	+	-
Сильна статична типізація	+	-	+	-	+	+
Потужне IDE	+	+	+	+	+	+
Швидкодія	+	-	+	-	+	+

Середовище розробки: C# має одне з найпотужніших інтегрованих середовищ розробки - Visual Studio яке надає широкий спектр інструментів для розробки, відлагодження та тестування програмного коду. Глибока інтеграція з платформою .NET забезпечує зручність та ефективність розробки на C#.

На основі таблиці видно, що мова C# відповідає всім критеріям для успішної реалізації цього проекту. В порівнянні з іншими мовами, такими як Python або Java, C# має ряд вирішальних переваг. На відміну від Python, C# має глибоку інтеграцію з Windows та платформою .NET, що робить його більш зручним для роботи з системними подіями. Java, будучи мовою орієнтованою на портативність, не має такої глибокої інтеграції з конкретними ОС, як C#. Додатково, для виконання Java-програм потрібне середовище виконання Java (JRE), що може стати обмеженням. Таким чином, враховуючи потреби цього проекту, специфіку задачі та середовище використання, мова програмування C# виявилась найбільш оптимальним вибором для реалізації цього модуля.

2.4 Використання sysmon в архітектурі програмного модуля

Sysmon є важливим компонентом в архітектурі програмного модуля, розробленого в рамках цієї кваліфікаційної роботи. Sysmon - це служба та драйвер фільтра подій, що моніторить різні події на системному рівні в операційній системі Windows. Sysmon може збирати різні типи даних про події, включаючи мережеві з'єднання, зміни в реєстрі системи та інші події.

Наступні основні причини обґрунтовують використання Sysmon в архітектурі програмного модуля:

- Висока деталізація системних подій. Sysmon надає детальну інформацію про велику кількість системних подій, що відбуваються в операційній системі. Це включає в себе реєстрацію подій створення процесів та їх взаємодію з іншими процесами як локально так і через мережу, завантаження драйверів, модифікації гілок реєстру та навіть тих, що спрямовані на sysmon та його складові.
- Сумісність з іншими інструментами безпеки. Sysmon веде журнал подій в Event Logs та записує логи в .xml форматі. Використовуючи цю особливість в перспективі можна створити окремий сервер на якому буде здійснюватися обробка, зберігання та візуалізація логів. А якщо поєднувати логи з систем, що знаходяться в одній мережі то вдасться виявляти складні атаки на інфраструктуру.

- Фільтрація. В конфігураційному файлі є можливість налаштувати фільтри для вибору подій, які повинні збиратися. Тобто є можливість відсіяти логи системи та зосередити увагу саме на подіях безпеки, тим самим економити обчислювальні ресурси, уникнути нагромадження логів. Додатково здійснення операцій з файлами конфігурації та налаштувань Sysmon можна за допомогою групових політик, використання PowerShell та WMI.
- Доступність та відкритість: Sysmon є безкоштовним і відкритим інструментом, що постійно оновлюється і підтримується Microsoft. Тобто фактично ми маємо генератор системних подій, що підписаний компанією Microsoft і якому можна довіряти.

2.4.1 Огляд архітектури модуля SYSMON

System Monitor складається з сервісу та драйвера.

Сервіс Sysmon.exe запускається ще до того як користувач здійснить авторизацію в системі та є відповідальним за запуск і виконання основних функцій Sysmon, а саме встановлення та видалення драйвера фільтрації подій, обробка конфігураційних файлів, запис логів в журнал Sysmon/Operetional.

Драйвер фільтрації подій Sysmon.sys відповідає за моніторинг системи. Він встановлюється на рівні ядра ОС і використовує фільтри , що прописані в конфігураційному файлі для перехоплення подій, що відбуваються в системі ще до того як вони підуть на виконання у ядро.

Журнал подій Windows Sysmon\Operetional.evtx записує дані про події в журнал подій. Дані зберігаються в XML форматі, що спрощує їх аналіз та обробку.

Конфігураційний файл використовується для налаштування фільтрів Sysmon, включаючи визначення типів подій за якими необхідно здійснювати моніторинг , та умови, за яких ці події повинні бути зареєстровані.

Ці компоненти взаємодіють, щоб забезпечити моніторинг за компонентами ОС Windows на всіх рівнях операційної системи.

Також окрему увагу sysmon приділяє відстеженню інструментів WMI (Windows Management Instrumentation) для відстеження подій, що стосуються системного управління та конфігурації.

WmiPrvSE.exe - це процес, що виконує WMI запити від різних додатків і служб у Windows. Sysmon відстежує WMI запити та відповіді, а також логує події, що були створені з використанням WMI.

Взаємодію всіх компонентів Sysmon та рівнів операційної системи Windows зображено на рисунку 2.2.

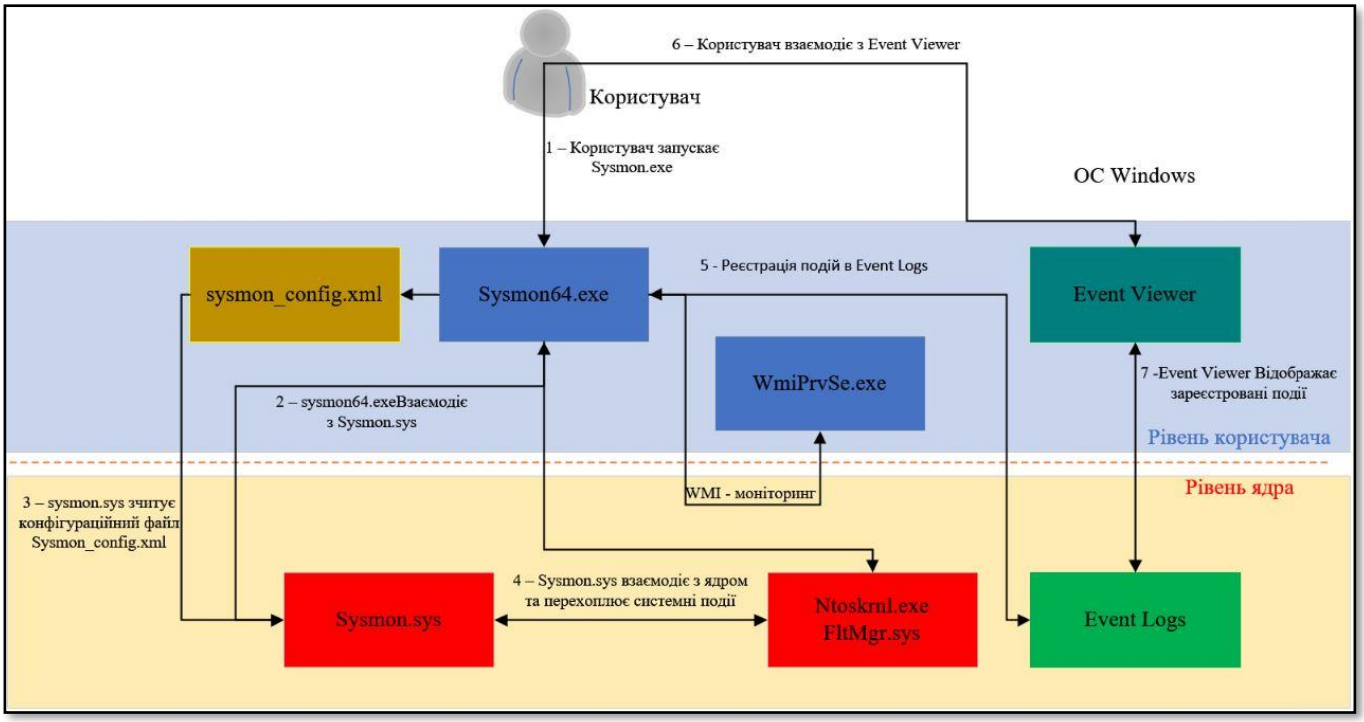


Рисунок 2.2 – Схема роботи Sysmon в ОС Windows

Розглянемо детальніше схему роботи Sysmon в ОС Windows:

На рівні ядра:

- Ntoskrnl.exe: це ядро ОС Windows, яке взаємодіє з драйвером фільтрації файлової системи (fltMgr.sys).
- fltMgr.sys: це драйвер фільтрації файлової системи, який взаємодіє як з ядром ОС (Ntoskrnl.exe), так і з драйвером Sysmon (sysmon.sys).

- sysmon.sys: це драйвер Sysmon, який взаємодіє з драйвером фільтрації файлової системи (fltMgr.sys) та записує дані про події в журнал подій (Event Log).
- Event Log: це журнал подій Windows, в який записуються дані про події від Sysmon.

На рівні користувача:

- Sysmon Service: це служба Sysmon, яка взаємодіє з WmiPrvSE.exe та використовує конфігураційний файл. Вона також взаємодіє з драйвером Sysmon на рівні ядра.
- WmiPrvSE.exe: це процес, який виконує запити WMI від різних додатків та служб Windows. Sysmon Service відстежує цей процес для відстеження запитів та відповідей WMI.
- Configuration File: це конфігураційний файл, який використовує Sysmon Service для налаштування поведінки Sysmon.
- Event Viewer: це інструмент, який читає дані з журналу подій на рівні ядра.

Перевага SYSMON полягає в тому, що він працює на рівні ядра операційної системи, це означає, що він логує дії перед тим, як вони пройдуть через ядро та будуть виконані. Це дозволяє отримати детальну інформацію про події системи, включаючи ті, що могли бути змінені або приховані на вищих рівнях. Водночас, це гарантує, що логи SYSMON важко змінити або видалити, що забезпечує їх надійність.

2.4.2 Дослідження структури полів логів sysmon

В попередньому підрозділі було проведено дослідження механізму роботи Sysmon. Тепер необхідно провести детальне дослідження структури логів Sysmon. Для цього було використано наступний конфігураційний файл, що включає логування усіх типів подій (рисунок 2.3).

```
<Sysmon schemaversion="4.82"> <!-- Версія схеми конфігураційного файлу -->
  <!-- Визначає алгоритми хешування, які будуть застосовуватися -->
  <HashAlgorithms>MD5</HashAlgorithms> <!-- Використовується MD5 алгоритм хешування -->
  <EventFiltering> <!-- Початок блоку фільтрації подій -->
        <!-- Виключає логування подій створення процесу -->
        <ProcessCreate onmatch="exclude">

        </ProcessCreate>
        <!-- Виключає логування подій зміни часу створення файлу -->
        <FileCreateTime onmatch="exclude">

        </FileCreateTime>
        <!-- Виключає логування подій мережевого з'єднання -->
        <NetworkConnect onmatch="exclude">

        </NetworkConnect>
        <!-- Виключає логування подій завершення процесу -->
        <ProcessTerminate onmatch="exclude">

        </ProcessTerminate>
        <!-- Виключає логування подій завантаження драйвера -->
        <DriverLoad onmatch="exclude">

        </DriverLoad>
        <!-- Виключає логування подій завантаження зображення -->
        <ImageLoad onmatch="exclude">

        </ImageLoad>
        <!-- Виключає логування подій створення віддаленого потоку -->
        <CreateRemoteThread onmatch="exclude">

        </CreateRemoteThread>
        <!-- Виключає логування подій зчитування сирих даних -->
        <RawAccessRead onmatch="exclude">

        </RawAccessRead>
        <!-- Виключає логування подій доступу до процесу -->
        <ProcessAccess onmatch="exclude">

        </ProcessAccess>
        <!-- Виключає логування подій створення файлу -->
        <FileCreate onmatch="exclude">

        </FileCreate>
        <!-- Виключає логування подій реєстру -->
        <RegistryEvent onmatch="exclude">

        </RegistryEvent>
        <!-- Виключає логування подій створення потоку хешу файлу -->
        <FileCreateStreamHash onmatch="exclude">

        </FileCreateStreamHash>
        <!-- Виключає логування подій каналу -->
        <PipeEvent onmatch="exclude">

        </PipeEvent>
        <!-- Виключає логування подій WMI -->
        <WmiEvent onmatch="exclude">

        </WmiEvent>
        <!-- Виключає логування подій DNS-запитів -->
        <DnsQuery onmatch="exclude">

        </DnsQuery>
        <!-- Виключає логування подій видалення файлів -->
        <FileDelete onmatch="exclude">

        </FileDelete>

    </EventFiltering> <!-- Кінець блоку фільтрації подій -->
</Sysmon>
```

Рисунок 2.3 – Конфігураційний файл Sysmon

Цей конфігураційний файл є трохи оманливим, оскільки він має атрибут onmatch='exclude' для всіх типів подій, але всі ці фільтри залишаються порожнім. У мові XML, пусті елементи можуть бути використані для вказівки на відсутність конкретних значень. Тому, якщо блоки фільтрації залишаються порожніми, це означає відсутність критеріїв фільтрації для виключення і Sysmon буде збирати всі події цього типу без виключень.

Наступним кроком є застосування нового файлу конфігурації. Це відбувається запуском команди sysmon64.exe -c конфігураційний_файл.xml.

Sysmon64.exe це служба завдяки чому ним можна керувати з будь-якої частини файлової системи, але зміни налаштувань будуть прийняті тільки при виконанні команд з правами системного адміністратора. Також потрібно прописати повний шлях до файлу конфігурації, або звернутися до сервісу з тієї частини файлової системи де знаходиться конфігураційний файл (рисунок 2.4).

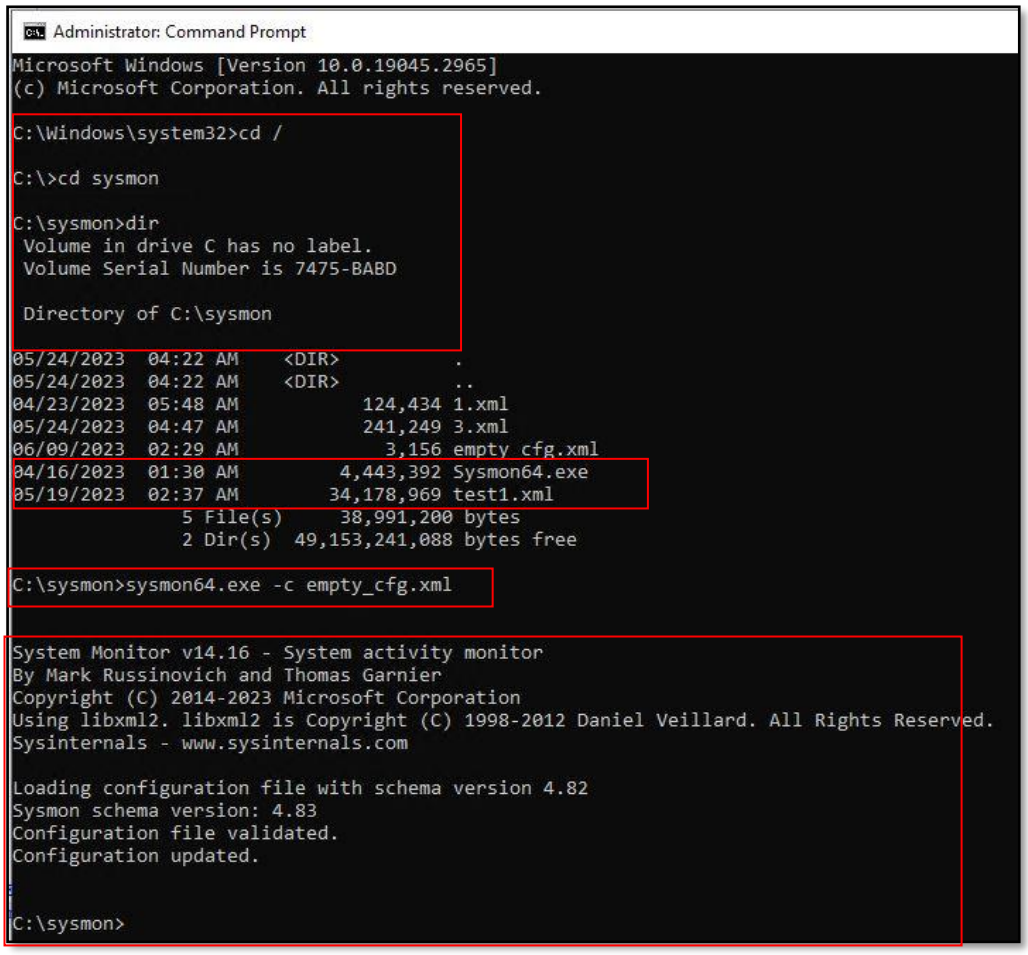


Рисунок 2.4 – Налаштування sysmon

Конфігураційний файл було застосовано успішно. Переглянути діючий файл конфігурації можна командою sysmon64.exe -c (рисунок 2.5).

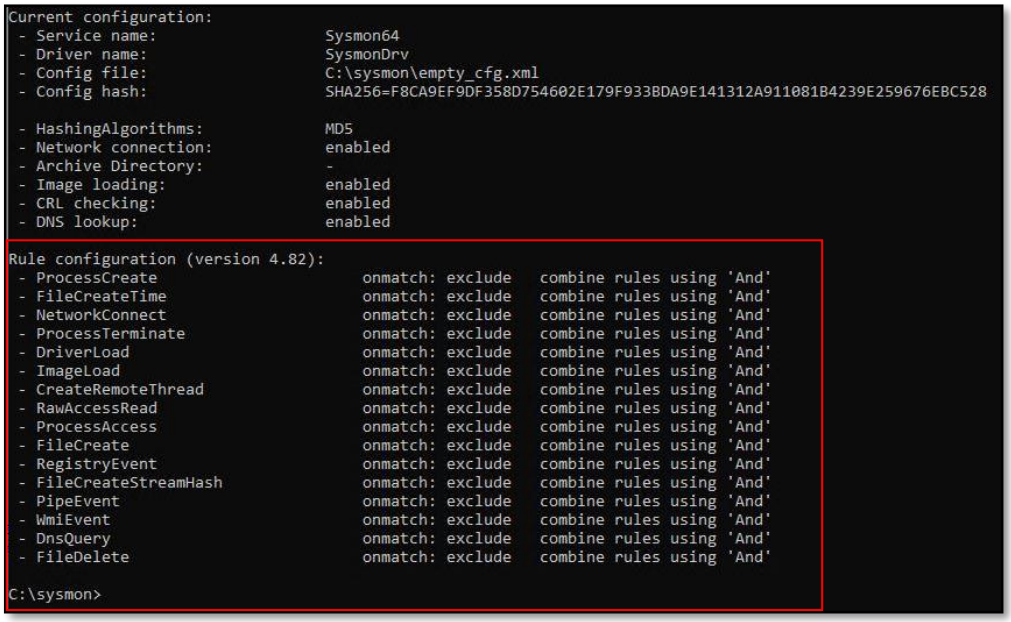


Рисунок 2.5 – Перевірка конфігураційного файлу

Sysmon працює на збирання всіх типів подій. Для проведення дослідження запустимо cmd.exe через Explorer.exe, у запущеній cmd.exe командою start cmd.exe запустимо новий командний рядок, і в ньому запустимо зовнішню команду ping.exe з параметром DNS – сервера Google 8.8.8.8 (рисунок 2.6)

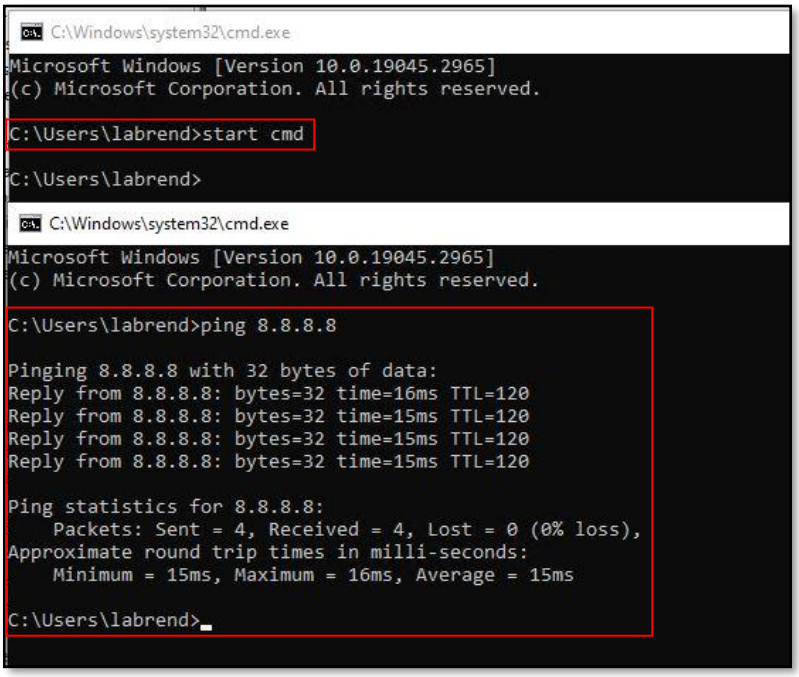


Рисунок 2.6 – Виконання дій через cmd.exe

Через EventViewer можна отримати логи, що зберігаються в Event logs які були згенеровані цими діями. Оскільки конфігураційний файл збирає всі події, що проходять через ядро операційної системи без попередньої фільтрації з урахуванням виключень відбувається досить велике нагромадження згенерованих подій, що продемонстровано на рисунку 2.7.

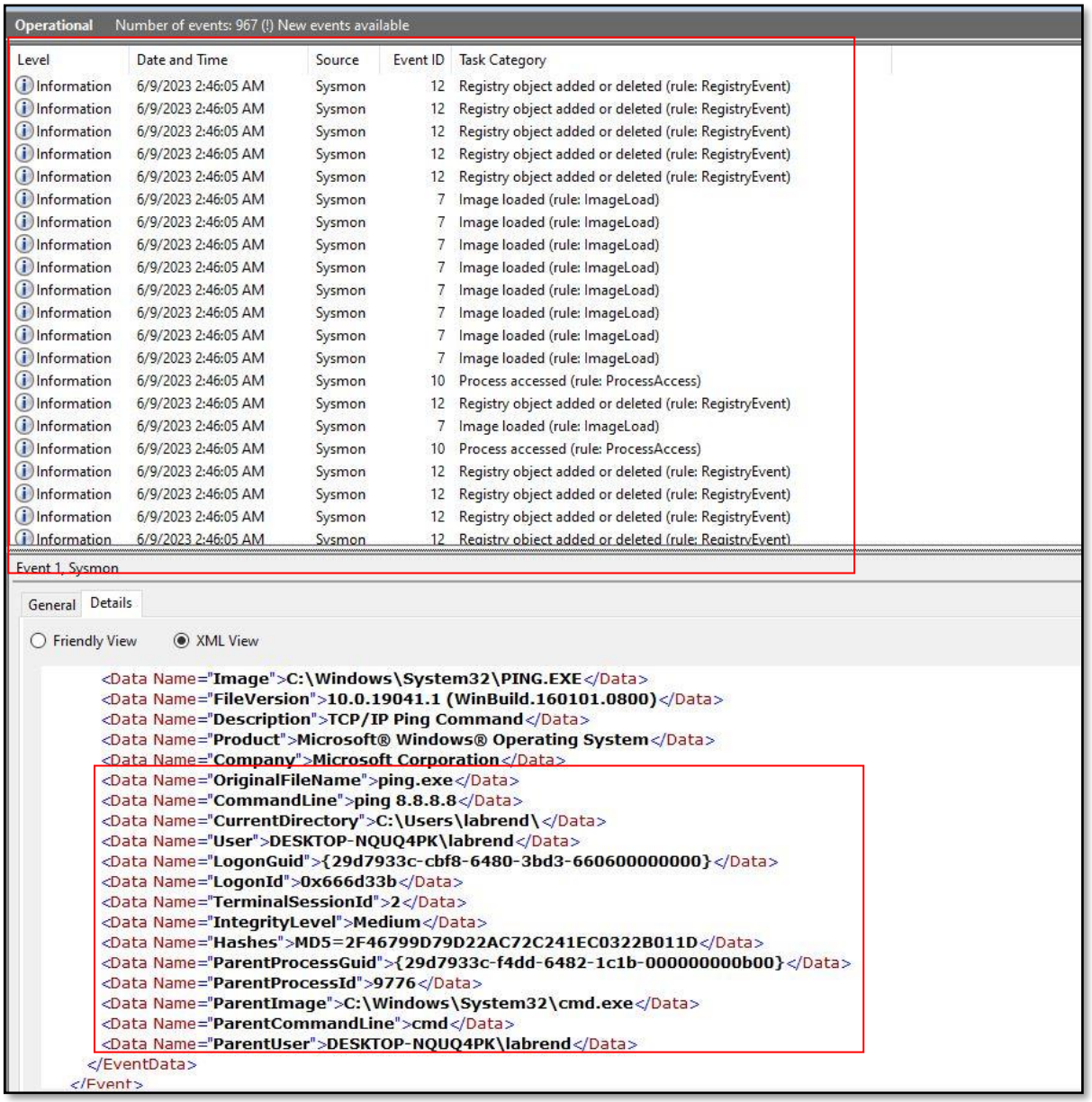


Рисунок 2.7 – Події згенеровані діями в cmd.exe

Для дослідження полів логів Sysmon було обрано ProcessCreate який відбувся при запуску команди ping.exe, що представлений на рисунку 2.8.

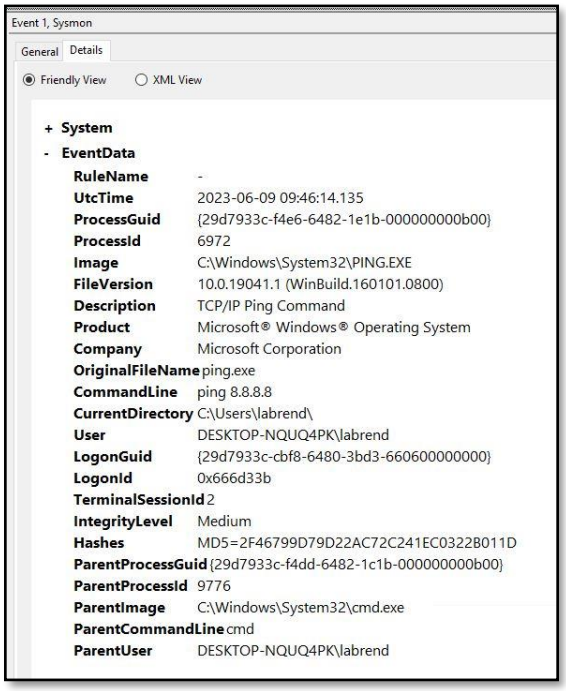


Рисунок 2.8 – Подія Sysmon

Це формат для читання, проте для глибшого дослідження доцільно буде розглянути цей лог у форматі xml. Що представлений на рисунку 2.9.

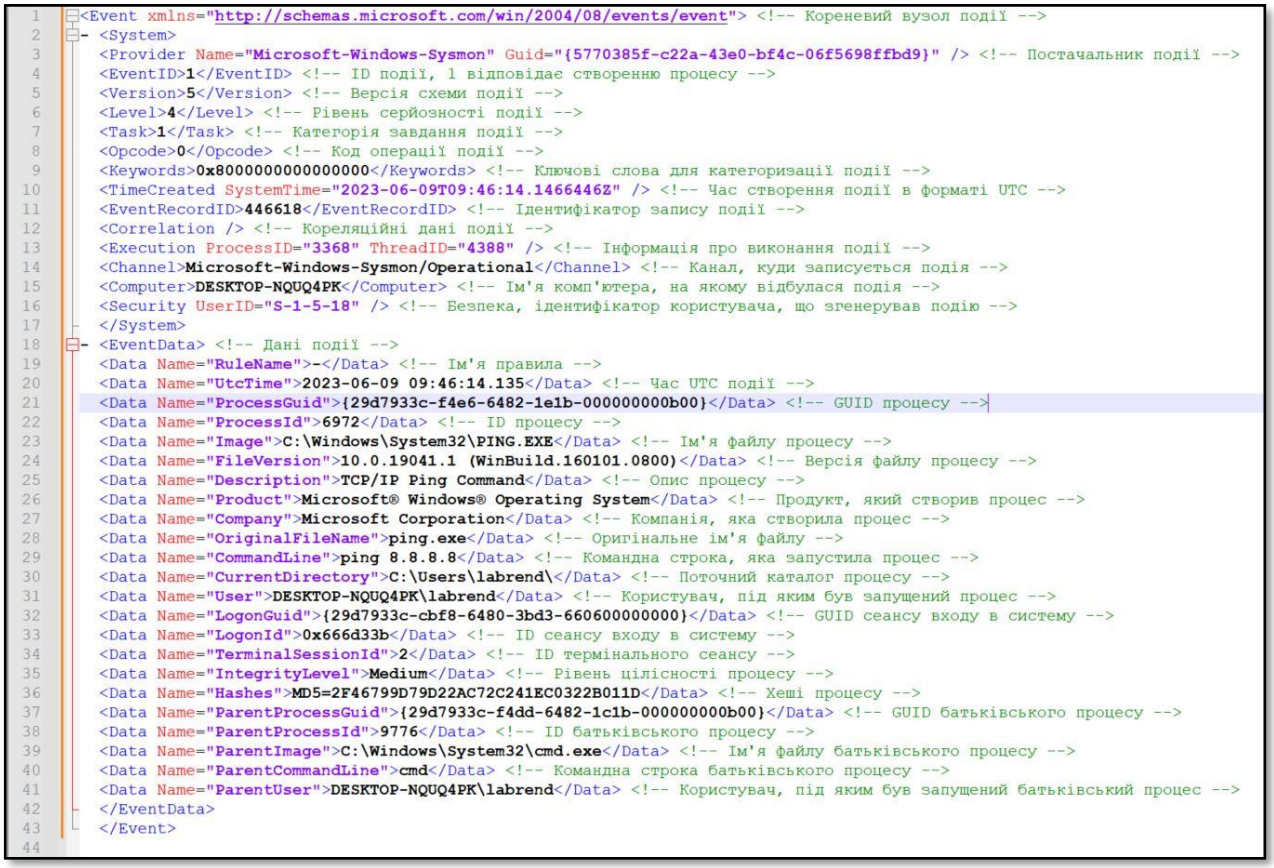


Рисунок 2.9 – Опис полів події Sysmon

Сама подія складається з двох блоків - <System> та <EventData>

Блок <System>: Цей блок містить системну інформацію про подію. Він включає в себе інформацію про постачальника події, ID події, версію схеми події, рівень події, категорію завдання події, код операції події, ключові слова для категоризації події, час створення події, ідентифікатор запису події, кореляційні дані події, інформацію про виконання події, канал, в який записується подія, ім'я комп'ютера, на якому відбулася подія, та інформацію про користувача, що згенерував подію.

Блок <EventData>: Цей блок містить детальну інформацію про саму подію. В залежності від типу події, що відстежується, вміст цього блоку може змінюватися. Досліджувана подія включає в себе інформацію про створення нового процесу. До цієї інформації входять такі дані, як назва правила, час UTC події, GUID процесу, ID процесу, ім'я файлу, версія файлу, опис, продукт, оригінальне ім'я файлу, командний рядок, що запустила процес, поточний каталог процесу, користувач, під яким був запущений процес, GUID сеансу входу в систему, ID сеансу входу в систему, ID термінального сеансу, рівень цілісності процесу, хеші процесу, GUID батьківського процесу, ID батьківського процесу, ім'я файлу батьківського процесу, командний рядок батьківського процесу, користувач, під яким був запущений батьківський процес.

Розглянемо детально кожне поле та за що воно відповідає:

Детальний опис полів <System>

- Provider: ім'я постачальника подій, тут вказано "Microsoft-Windows-Sysmon".
- EventID: ID події, це унікальний номер події у Sysmon.
- Version: версія формату події.
- Level: рівень журналу подій (Error, Warning, Information тощо).
- Task: тип завдання, яке згенерувало подію.
- Opcode: код операції, яка була здійснена при генерації події.
- Keywords: ключові слова, пов'язані з подією.
- TimeCreated: час створення події.
- EventRecordID: унікальний номер запису події.

- ProcessID та ThreadID: ідентифікатори процесу та потоку, які згенерували подію.
- Channel: канал, в який було записано подію.
- Computer: ім'я комп'ютера, на якому була згенерована подія.
- Security UserID: ідентифікатор користувача, під яким була згенерована подія.

Детальний опис полів <EventData>

- RuleName: ім'я правила, яке спрацювало для генерування події.
- UtcTime: час генерування події у форматі UTC.
- ProcessGUID та ProcessId: унікальний ідентифікатор процесу.
- Image: повний шлях до виконуваного файлу.
- FileVersion: версія файлу.
- Description: опис виконуваного файлу.
- Product: продукт, до якого належить виконуваний файл.
- Company: компанія, що створила виконуваний файл.
- OriginalFileName: оригінальне ім'я файлу.
- CommandLine: командний рядок, що був використаний для запуску процесу.
- CurrentDirectory: поточна директорія процесу.
- User: ім'я користувача, під яким був запущений процес.
- LogonGUID та LogonId: унікальний ідентифікатор сеансу входу користувача.
- TerminalSessionId: ідентифікатор термінального сеансу.
- IntegrityLevel: рівень цілісності процесу.
- Hashes: хеші виконуваного файлу.
- ParentProcessGUID та ParentProcessId: унікальний ідентифікатор батьківського процесу.
- ParentImage: повний шлях до виконуваного файлу батьківського процесу.
- ParentCommandLine: командний рядок, що був використаний для запуску батьківського процесу.
- ParentUser: ім'я користувача, під яким був запущений батьківський процес.

- Для здійснення обєднання подій в інциденти безпеки та побудови поведінкових сигнатур було обрано наступні:
- ProcessGUID та ProcessId: їх можна використовувати для відслідковування дій конкретного процесу впродовж часу.
- ParentProcessGUID та ParentProcessId: вони допомагають з'ясувати ієрархію процесів і визначити, який процес спочатку запустив інший процес.
- TimeCreated: це поле допомагає визначити послідовність подій.
- CommandLine та ParentCommandLine: ці поля можуть допомогти зрозуміти, які дії були виконані в рамках певного процесу.
- Image та ParentImage: ці поля дають змогу визначити, які програми були запущені в рамках певного процесу.

Логи Sysmon містять велику кількість унікальних ідентифікаторів для процесів та користувачів, завдяки чому можна з високою точність відтворити послідовність подій системи та вибудувати ієрархію процесів в якій буде відображено зв’язки процесів батько-син, взаємодію незалежних процесів, а також вплив віддалених процесів через мережу.

Після побудови деталізованої ієрархії процесів та їх взаємозв’язків доцільно провести дослідження таких полів як CommandLine/ParentCommandLine , Image/ParentImage, OriginalFileName, CurrentDirectory значення яких використати для створення патернів поведінки.

Таким чином, Sysmon є потужним інструментом для моніторингу системи на предмет підозрілої активності, а конфігураційний файл Sysmon стає базою даних поведінкових сигнатур.

2.5 Механізм поєднання логів

Розуміння інцидентів безпеки у контексті послідовності подій є важливою частиною аналізу безпеки. Для створення цілісного зображення дій, які відбулися перед, під час і після інциденту, необхідно зібрати та проаналізувати логи, та

розглянути їх в контексті інших процесів, що їх створили і які були створені тим процесом який потрапив під фільтр.

Система має відображати не просто факт створення процесу(EventID1), а що взагалі відбувається в системі, а для цього потрібно відсортувати логи за спільним ідентифікатором у групи та вибудувати зв’язки батько-син і врахувати логи дій, що виконувалися в рамках даного процесу і у випадку якщо процес не завершився(EventID5) продовжувати доповнювати такі групи додатковою інформацією.

Для об'єднання логів ключовими є поля ProcessGUID та ParentProcessGUID, які дозволяють відстежити ієрархію процесів та визначити, які дії були виконані в межах кожного процесу. Також важливою є інформація про час створення процесу UtcTime, яка дозволяє відтворити послідовність подій.

GUID (Globally Unique Identefier) – ціле число довжиною 128 біт, що ідентифікує ресурси, екземпляри та дії та має наступний вигляд - {00000000-0000-0000-0000-000000000000}.

Важливо пояснити різницю між ProcessGUID та ProcessId

ProcessGUID - це унікальний ідентифікатор, який Sysmon присвоює кожному процесу в системі. Його головною особливістю є унікальність на протязі всього життя системи. Навіть після перезавантаження системи використані ProcessGUID не будуть назначатися новим логам.

ProcessID – це стандартний ідентифікатор процесу в більшості операційних систем, включаючи Windows. Він може бути сумісним з великою кількістю відомих інструментів моніторингу. Крім того, ProcessID є простим числовим ідентифікатором, який швидше обробляється системою.

Проте тут є один великий недолік. Після завершення процесу система може переприсвоїти уже використаний ProcessID іншому процесу, що створить порушення всієї логіки побудови ієрархії батько-син. Цей недолік критичний і саме тому буде використовуватися ProcessGUID.

Після групування подій та побудови дерев процесів та дій, що відбуваються в рамках цих процесів необхідно визначити чи несуть ці дії/поведінка загрозу. Тобто

необхідно провести їх аналіз. Для цього можна використовувати файл конфігурації Sysmon_config.xml, який у випадку спрацювання фільтра додає опис в поле логу, завдяки чому при аналізі наче підсвічується не окрема подія, а ціла група подій.

Використовуючи такий підхід, можна визначити широкий спектр атак, від простих до складних, і розробити ефективні стратегії захисту.

Розглянемо це на прикладі ping 8.8.8.8 . Створимо фільтр в EventViewer на відсортування всіх подій з ProcessGUID {29d7933c-1809-6483-b41b-000000000b00} який продемонстровано на рисунку 2.10.

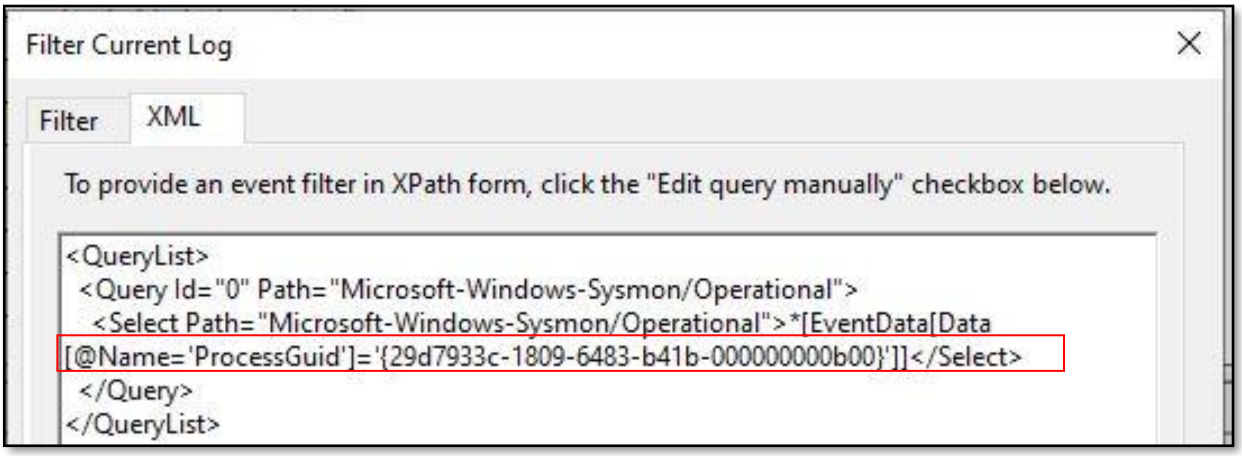
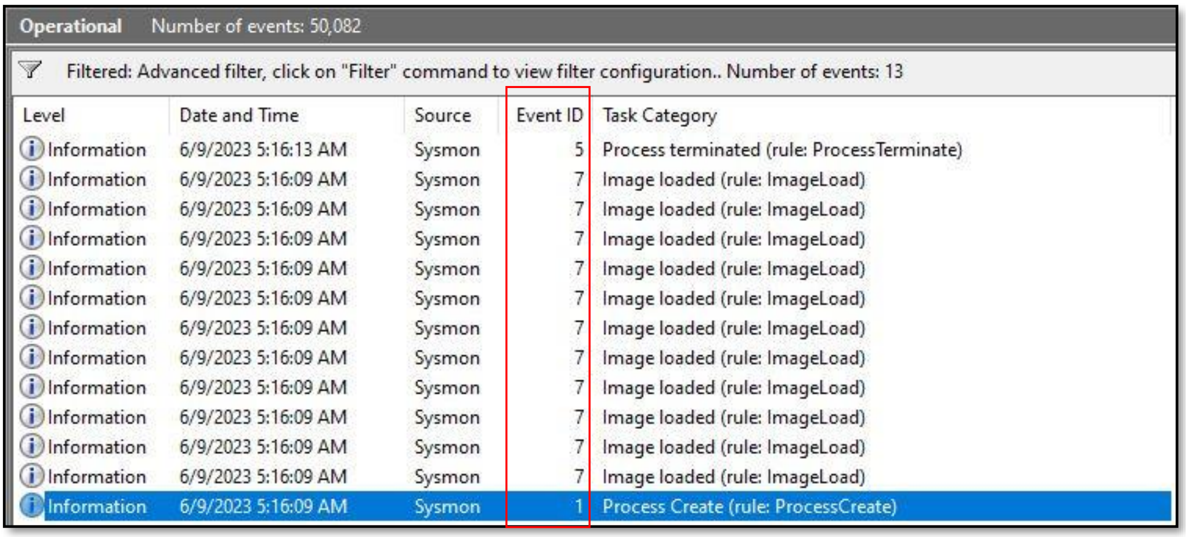


Рисунок 2.10 - Фільтр в EventViewer

В результаті застосування даного фільтру було відсортовано логи з однаковим GUID. Що продемонстровано на рисунку 2.11.



Operational Number of events: 50,082

Filtered: Advanced filter, click on "Filter" command to view filter configuration.. Number of events: 13

Level	Date and Time	Source	Event ID	Task Category
Information	6/9/2023 5:16:13 AM	Sysmon	5	Process terminated (rule: ProcessTerminate)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:09 AM	Sysmon	1	Process Create (rule: ProcessCreate)

Рисунок 2.11 – Відсортовані логи

Це все, що відбулося в рамках запуску зовнішньої команди ping. Розглянемо детально кожен лог, а саме типи подій 1,7,5 та виділимо важливі поля (рисунки 2.12, 2.13, 2.14).

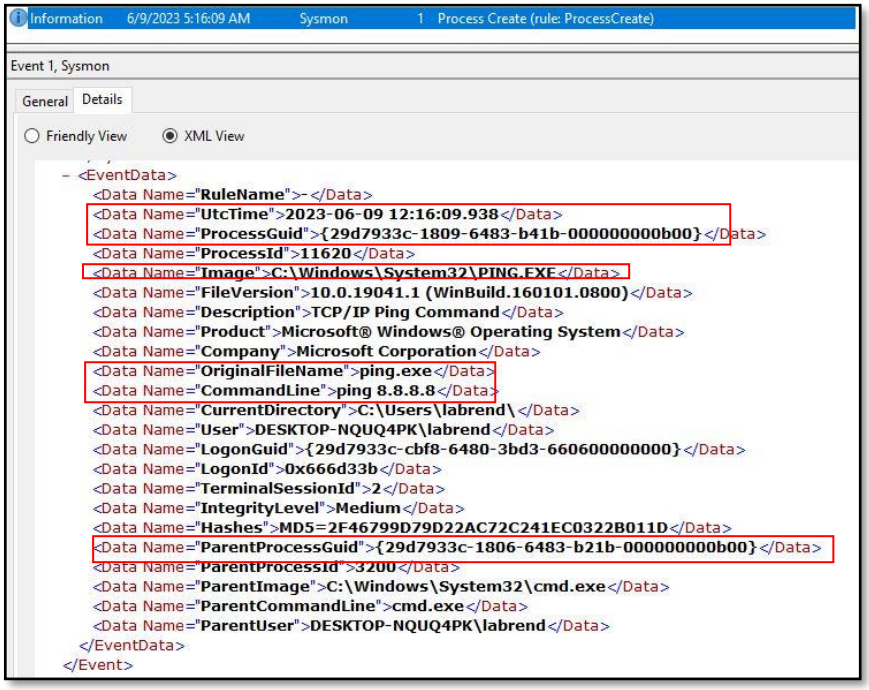


Рисунок 2.12 – Лог події в форматі xml

Особливу увагу варто приділити полям UtcTime, ProcessID, Image, ProcessGUID, OriginalFIleName, CommandLine та PrarentProcessGUID.

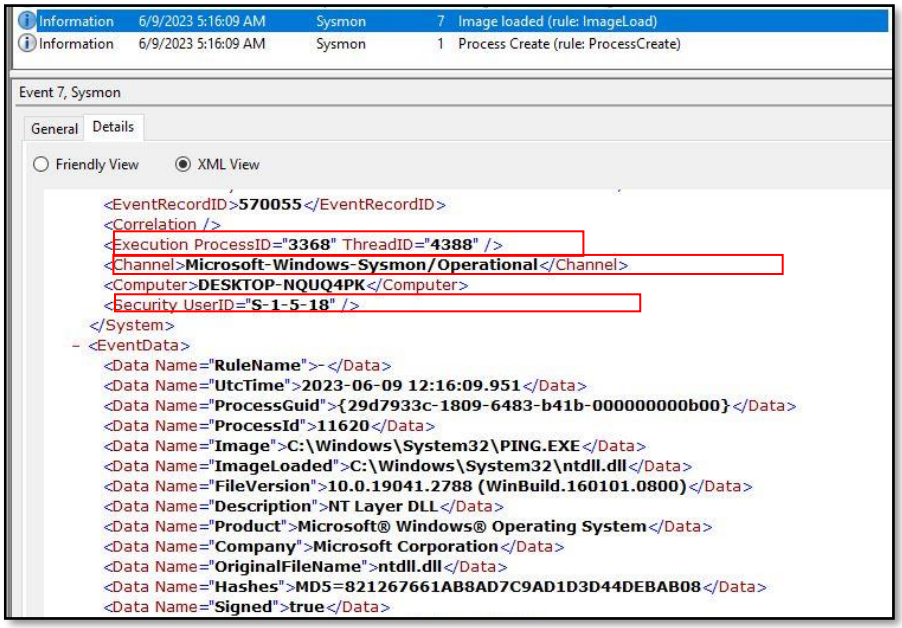


Рисунок 2.13 - Лог події в форматі xml

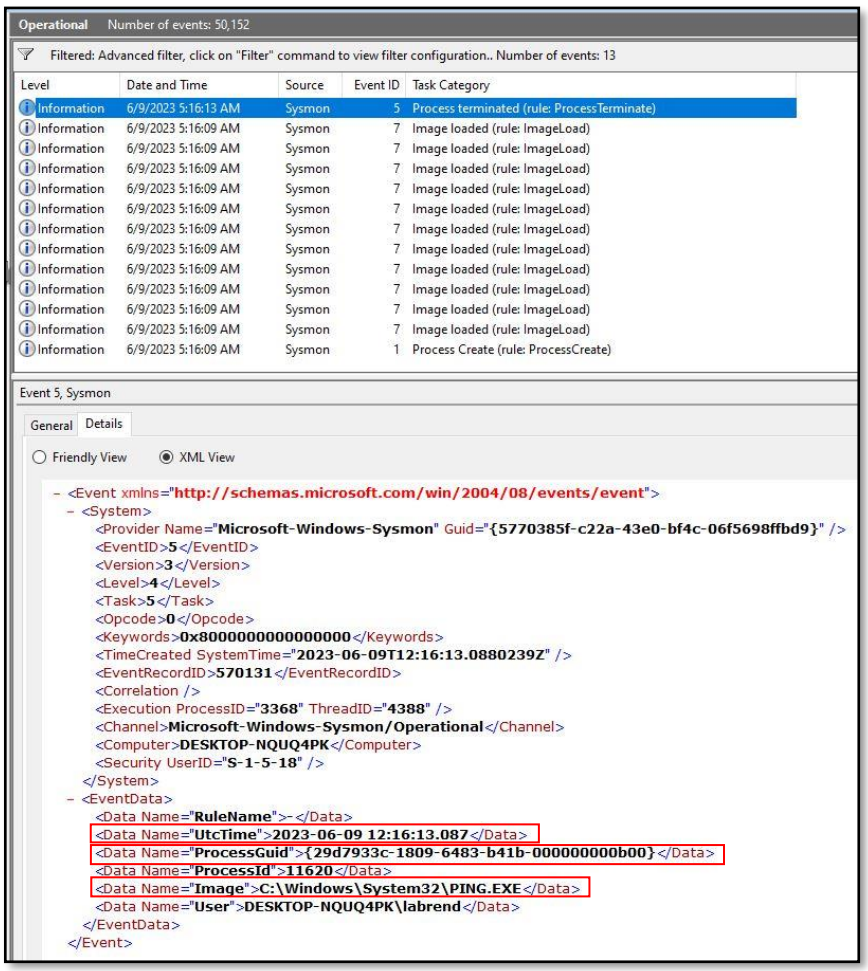


Рисунок 2.14 - Лог події в форматі xml

Тепер створимо фільтр за ProcessGUID {29d7933c-1806-6483-b21b-000000000b00}, оскільки саме це значення було в ParentProcessGUID попереднього процесу, який продемонстровано на рисунку 2.15

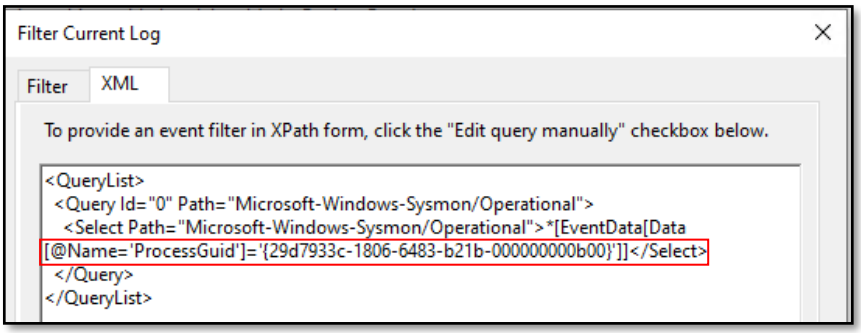
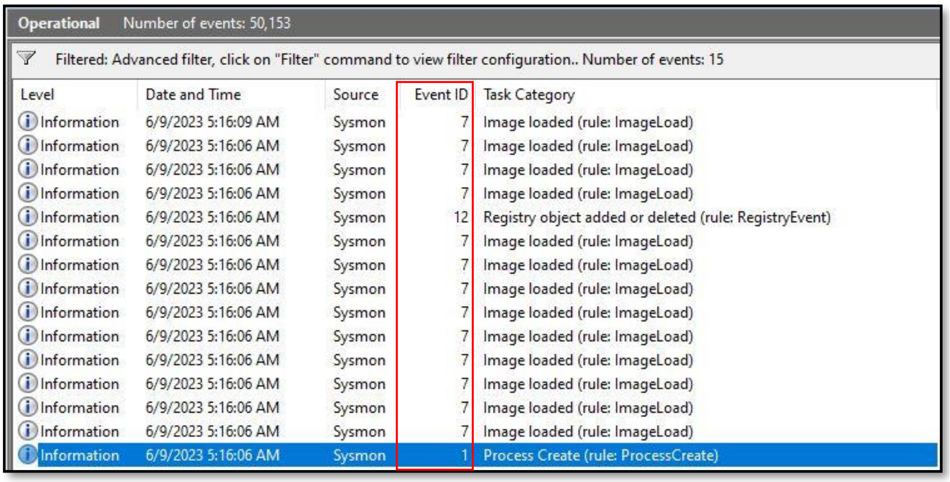


Рисунок 2.15 - Фільтр в EventViewer

В результаті застосування даного фільтру було відсортовано логи з однаковим GUID – рисунок 2.16



Operational Number of events: 50,153

Filtered: Advanced filter, click on "Filter" command to view filter configuration.. Number of events: 15

Level	Date and Time	Source	Event ID	Task Category
Information	6/9/2023 5:16:09 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	12	Registry object added or deleted (rule: RegistryEvent)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	7	Image loaded (rule: ImageLoad)
Information	6/9/2023 5:16:06 AM	Sysmon	1	Process Create (rule: ProcessCreate)

Рисунок 2.16 Відсортовані логи

Це все, що відбулося в рамках cmd.exe. Розглянемо детально кожен лог, а саме типи 1,7,12 та дослідимо дії, що відбувалися в рамках цього процесу. Тобто умовно в контексті логів Sysmon, cmd.exe можна вважати певним контейнером, де Process Create виступає в ролі заголовка та є ланкою зв'язку з іншими процесам(такими самими контейнерами).

Першим йде EventID-1 Process Create , значення його ProcessGUID аналогічне до ParentProcessGUID попередньої групи логів (рисунки 2.17 , 2.18, 2.19).

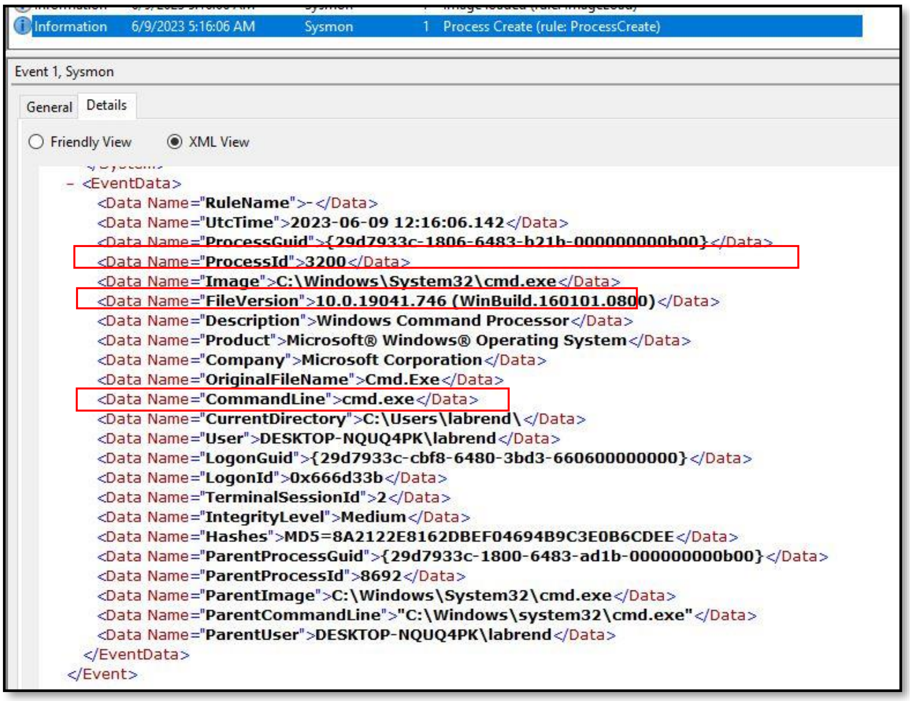


Рисунок 2.17 - Лог події в форматі xml

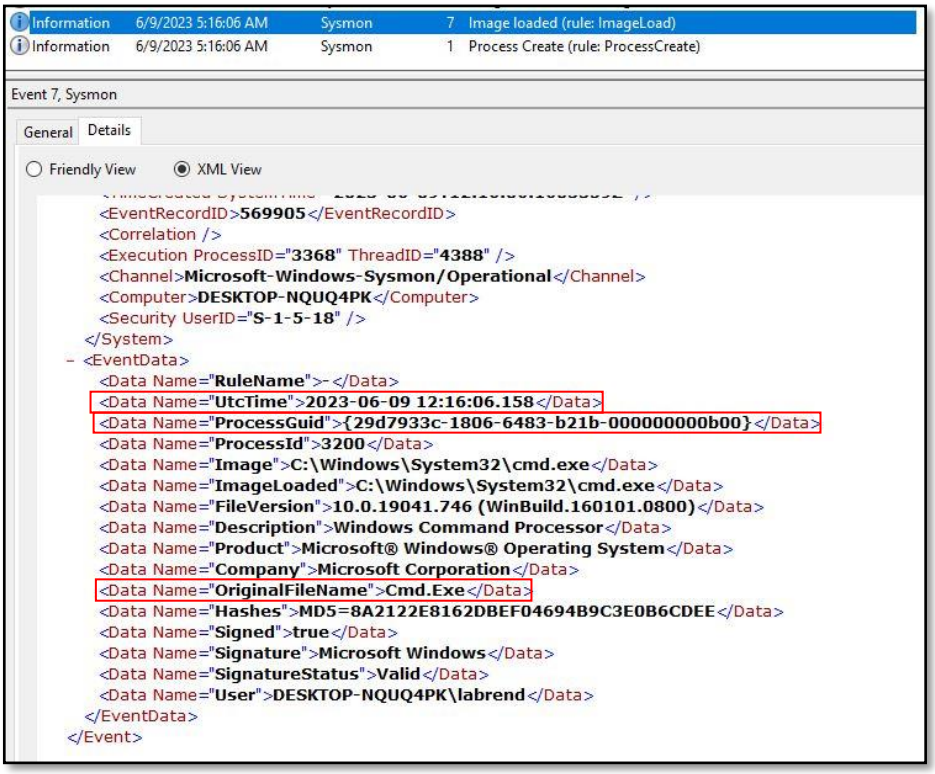


Рисунок 2.18 - Лог події в форматі xml

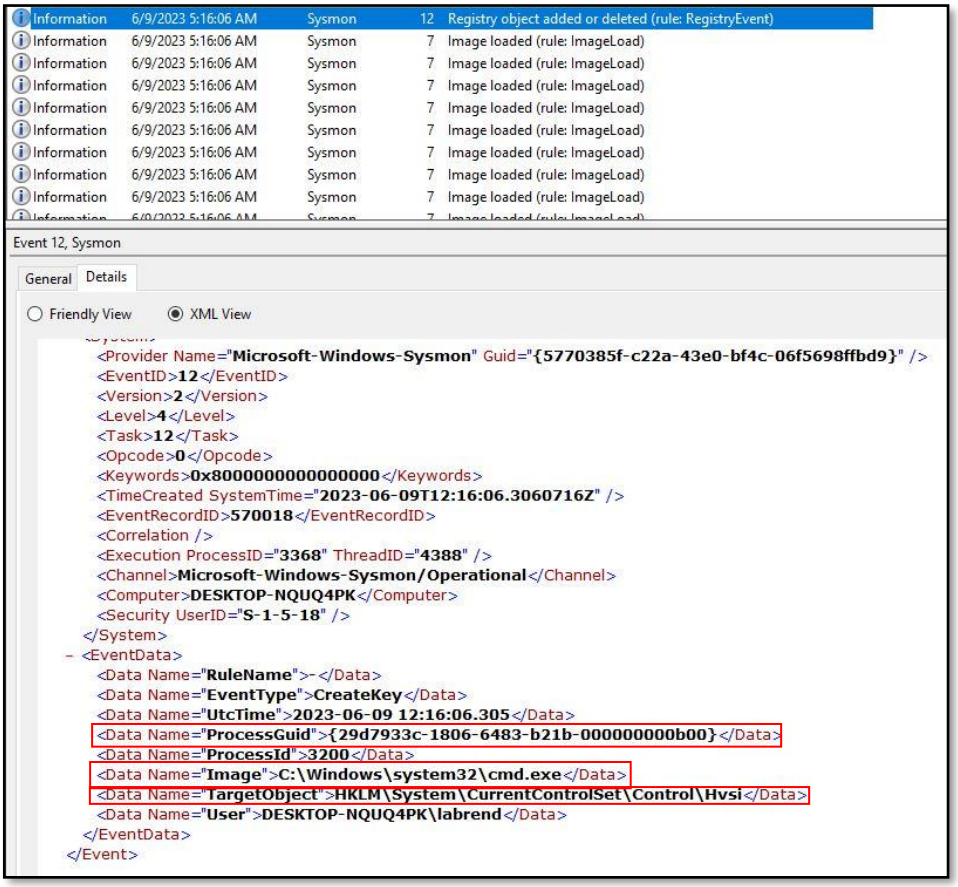


Рисунок 2.19 - Лог події в форматі xml

Таким чином об’єднання логів у групи та побудова дерев процесів з відображення наслідування батько-син мають схему, що зображена на рисунку 2.20.

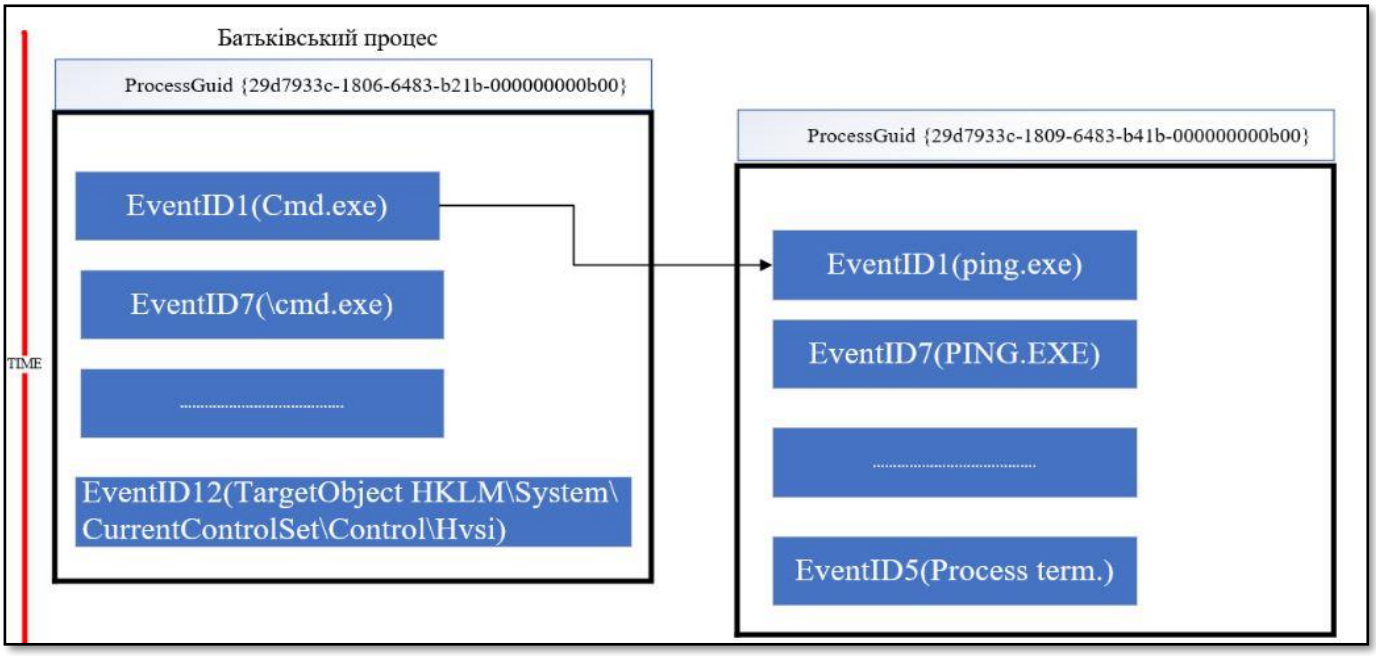


Рисунок 2.20 - Схема дерева процесів

У даному розділі було проведено дослідження інструменту Sysmon як потужного засобу збору та аналізу системних логів в контексті обробки інцидентів безпеки. Специфіка структури логів Sysmon була детально розглянута, включаючи роз'яснення значень ключових полів та їх ролі в процесі відстеження та розуміння системних подій.

Висновки за розділом 2

Цей розділ присвячений створенню архітектури програмного модуля виявлення шкідливого ПЗ на основі поведінкового аналізу. Для його написання доцільно використовувати мову C# та середовище програмування Visual Studio.

Розглянуто особливості інструменту Sysmon як потужного генератора системних логів. В процесі дослідження було детально розглянуто архітектуру додатку Sysmon, його складові та принцип роботи, а також використання конфігураційного файлу.

Було проведено детальний аналіз структури логів Sysmon, що включав роз'яснення значень ключових полів, таких як "UtcTime", "ProcessGuid", "ParentProcessGuid", "Image", "CommandLine", "User" та інших. Саме ці поля є основою для побудови взаємозв’язків між логами і як наслідок створення точних шаблонів поведінки ПЗ. Використання Sysmon у даному контексті може забезпечити детальне розуміння поведінки на всіх рівнях операційної системи.

Особливу увагу було приділено полю "ProcessGUID", що виявився ключовим для відстеження ієрархії процесів та дій, що виконуються в межах кожного з них. Його унікальність на протязі всього часу життя системи робить його важливим інструментом при відстеженні векторів атак, що включають складні послідовності дій. Завдяки цьому вдалося створити механізм поєднання логів у групи.

Отримані в даному розділі результати стануть основою для подальшої практичної реалізації додатку виявлення шкідливого ПЗ.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ДЛЯ ВИЯВЛЕННЯ ШКІДЛИВОГО ДОДАТКУ

3.1 Взаємодія з програмою

Для забезпечення ефективної взаємодії з програмою було створено головне вікно програми, за допомогою класу FormMain – головний елемент графічного інтерфейсу. Це вікно включає в себе важливі елементи керування, такі як listBox_events та updateTreeView, які використовуються для представлення інформації про події в системі користувачу.

Також реалізовано автоматичне оновлення інтерфейсу користувача. Ця функція, реалізована через метод autoRefresh - здійснює постійне оновлення форми.

Таким чином, використання різноманітних інструментів і методів для створення графічного інтерфейсу користувача забезпечує простоту і зручність використання програми "Windows WatchDog".

3.1.1 Використання FormMain для створення головного вікна програми

FormMain - головне вікно програми. Виокремлення цього класу дозволило зосередитися на розробці користувацького інтерфейсу, відокремленого від основного коду програми. FormMain надає користувацькому відображення подій системи та управління їх відстеженням. Декілька основних компонентів, які були використані в цьому класі, включають:

- ListBox для відображення подій системи.
- Button для запуску відстеження подій та
- TreeView для подання інформації про події в системі у вигляді дерева процесів для користувача форматі.

Ось фрагмент коду, що демонструє використання FormMain:

```
public partial class FormMain : Form {
    private EventLogWatcher watcher = null;

    public FormMain() {
        InitializeComponent();
        ...
    }

    private void button_start_Click(object sender, EventArgs e)
{
        if (watcher != null) {
            watcher.Dispose();
            watcher = null;
        }
        start();
    }
}
```

У цьому фрагменті ми можемо побачити, що клас FormMain включає приватну змінну watcher типу EventLogWatcher, яка використовується для відстеження подій системи. Конструктор FormMain() ініціалізує компоненти Windows Forms, включаючи кнопки, списки та інші елементи інтерфейсу. Кнопка "Start" починає процес відстеження подій, запускаючи метод start(), коли користувач натискає на неї.

Вигляд інтерфейсу наведено на рисунок 3.1

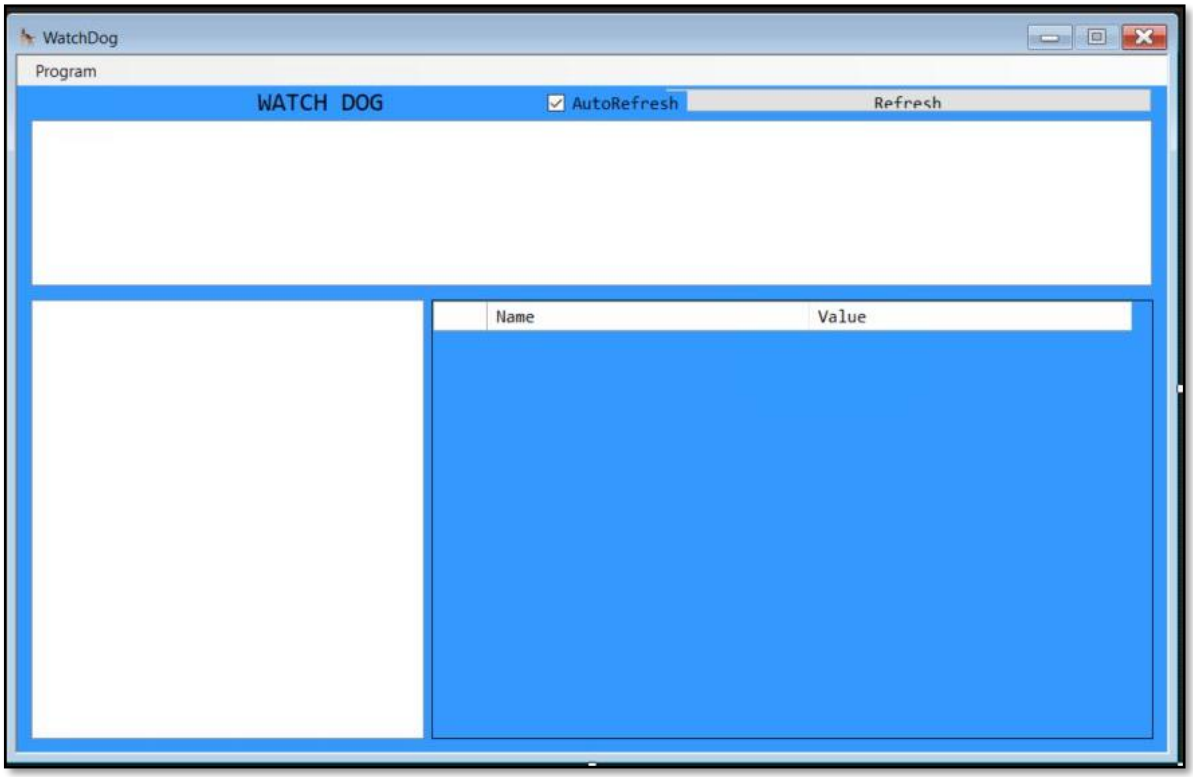


Рисунок 3.1 – Графічний інтерфейс додатку

3.1.2 Реалізація подій вікна за допомогою методів listBox_events_DrawItem та updateTreeView

Метод listBox_events_DrawItem використовується для настроювання візуального відображення кожного елемента в listBox_events.

Він приймає наступні параметри:

sender є об'єктом, який викликає подію (в цьому випадку, listBox_events)

e - це інстанція класу DrawItemEventArgs, яка надає інформацію про подію малювання.

Метод updateTreeView оновлює дерево процесів, яке відбувається синхронно при кожному автоматичному оновленні інтерфейсу користувача.

```
private void listBox_events_DrawItem(object sender,
DrawItemEventArgs e)
{
    MyListBoxItem item = listBox_events.Items[e.Index] as
MyListBoxItem;
    if (item != null)
    {
        e.DrawBackground();
        Brush myBrush = item.ItemColor;
        e.Graphics.DrawString(item.Message, e.Font, myBrush,
e.Bounds, StringFormat.GenericDefault);
        e.DrawFocusRectangle();
    }
    else
    {
        //
    }
}
```

У цьому фрагменті коду listBox_events_DrawItem використовує MyListBoxItem - власний клас, що розробляється для представлення елементів у listBox_events. Кожен елемент має колір (ItemColor) та текст (Message), що відображається. Якщо елемент не є екземпляром MyListBoxItem, він пропускається. csharp

```
private void updateTreeView() {
    ...
    foreach (EventRecord er in this.eventRecordList) {
        ...
    }}
```

3.1.3 Реалізація функції autoRefresh для автоматичного оновлення інтерфейсу користувача

Функція autoRefresh - запускає вічний цикл, який оновлює вміст treeView_process кожну секунду, якщо встановлено прапорець автооновлення. Важливо зазначити, що оновлення проводиться асинхронно, тому що ця операція може бути часом вимогливою, а ми не хочемо блокувати головний потік виконання програми.

```
private void autoRefresh() {
    while(!this.IsDisposed) {
        if (!treeView_process.IsHandleCreated ||
!checkBox_autorefresh.IsHandleCreated) {
            Thread.Sleep(1000);
            continue;
        }
        if (checkBox_autorefresh.Checked) {
            if (treeView_process.IsHandleCreated)  {
                treeView_process.Invoke((MethodInvoker)delegate
()
                {
                    updateTreeView();
                });
            }
        }
        Thread.Sleep(1000);
    }
}
```

У цьому фрагменті коду перевіряється статус "Disposed" поточної форми, щоб забезпечити, що додаток не спробує оновити інтерфейс, якщо форма була вже закрита. Далі перевіряється, чи створені відповідні важелі (Handles) для treeView_process та checkBox_autorefresh.

Якщо це не так, потік спить на секунду, а потім продовжує цикл. Якщо прапорець автооновлення встановлено, цикл виконує дію Invoke для treeView_process, викликаючи метод updateTreeView. Invoke гарантує, що updateTreeView виконується в потоці, в якому було створено treeView_process (це важливо, бо WinForms вимагає, щоб вся взаємодія з контролями відбувалася в головному потоці). Після цього потік знову "спить" на секунду, перед тим як продовжити цикл.

3.2 Використання класів для обробки подій системи

Цей підрозділ призначений для детального аналізу ключових класів, що використовуються для обробки подій системи в додатку "Windows WatchDog". Класи, вироблені в даному проекті, включають Utils, EventRecord та Log.

Клас Utils містить набір утилітарних методів, що допомагають з опрацюванням подій. Клас EventRecord відповідає за представлення інформації про події, отриманої з журналу подій Windows. Він надає методи для отримання детальної інформації про конкретну подію. Нарешті, клас Log дозволяє записувати інформацію про події до журналу для подальшого аналізу.

В цьому розділі будуть розглянуті важливі методи цих класів, а також приклади їх використання в коді проекту. Крім того, будуть наведені фрагменти коду, що демонструють, як ці класи використовуються для обробки подій системи.

3.2.1 Клас Utils і його методи

Клас Utils та його методи. Цей клас містить статичні методи, що використовуються в різних частинах програми для виконання загальних завдань. Основні методи та їх призначення можна проілюструвати за допомогою фрагментів коду:

```
public static class Utils {
    public static string FormatGUID(string guid) {...}
    public static string GetEventNameFromID(string id) {...}
    public static string FormatEventTime(string time) {...}
}
```

Метод FormatGUID(string GUID) призначений для форматування рядків, що представляють унікальні ідентифікатори (GUID). Це зазвичай використовується при роботі з API, які вимагають GUID в конкретному форматі.

Метод GetEventNameFromID(string id) конвертує ідентифікатор події в рядок, що представляє назву події. Це може бути корисним для виведення інформативних повідомлень про події.

Метод FormatEventTime(string time) приймає рядок, що представляє час у вигляді, що повертається API журналу подій, і перетворює його в більш зручний для людини формат.

Ці методи використовуються у різних частинах програми для виконання загальних завдань, що забезпечують коректну роботу програми.

3.2.2 Клас EventRecord і його методи

EventRecord та його методи. Його головне завдання полягає у представленні інформації про події, отриманої з журналу подій Windows. Для цього в класі EventRecord реалізовано ряд методів. Розглянемо основні методи та їх призначення на прикладі фрагментів коду:

```
class EventRecord
{
    public string GetEventID() {...}
    public string GetEventRecordID() {...}
    public string GetEventTimeCreated() {...}
    public string GetEventProcessID() {...}
    public string GetEventThreadID() {...}
    public string GetEventImage() {...}
    public string GetEventMessage() {...}}
```

- Метод GetEventID() використовується для отримання ідентифікатора події, який визначає тип системної події.
- Метод GetEventRecordID() повертає унікальний ідентифікатор запису події у журналі подій.
- GetEventTimeCreated() дозволяє отримати час створення події.
- GetEventProcessID() та GetEventThreadID() повертають ідентифікатори процесу та потоку відповідно, в яких відбулась подія.

- GetEventImage() повертає назву образу процесу, який створив подію.
- GetEventMessage() відповідає за відтворення повідомлення події. Ці методи разом забезпечують виведення детальної інформації про кожну системну подію, що дозволяє адекватно реагувати на неї.

3.2.3 Клас Log і його методи

Клас Log і його методи. Він виводить події у ListBox в головному вікні додатку та зберігає всі події, що були отримані на протязі виконання програми, а також використовується для моніторингу стану програми.

```
public class Log {
    public static void setListBox(ListBox lb) {...}
    public static void appendEvent(string msg, Color color)
{...}
}
```

Метод setListBox(ListBox lb) встановлює об'єкт ListBox, в який будуть записуватися повідомлення. Цей метод використовується для ініціалізації журналу подій.

Метод appendEvent(string msg, Color color) додає нове повідомлення в журнал подій. Він приймає рядок, який представляє повідомлення, і колір, що використовується для його відображення. Цей метод використовується при виникненні нової події для її запису в журнал.

Опираючись на ці методи, клас Log надає простий інтерфейс для ведення журналу подій, що допомагає відстежувати роботу програми – рис

3.3 Реалізація основного циклу відстеження подій

Цей розділ присвячений реалізації основного циклу відстеження подій у системі, який є ключовим для функціональності програми.

Цикл включає ініціалізацію об'єкта події для перевірки чи відносяться нові логи до вже існуючої групи подій.

Процес відстеження відбувається за допомогою декількох функцій та методів.. Ці функції включають методи, які реагують на певні події, та методи, які виконують періодичне оновлення для відстеження змін.

Основний цикл відстеження подій - це серце програми, і він забезпечує важливість такого застосунку для системного моніторингу.

3.3.1 Реалізація функції start для запуску відстеження подій системи

Функція start є ключовою частиною програми Windows WatchDog. Вона викликається при запуску програми та ініціює процес відстеження системних подій. Функція start описана у наведеному нижче вирізку коду:

```
private void start() {
    Log.setListBox(this.listBox_events);

    var obj = new EventLogQuery("Microsoft-Windows-
Sysmon/Operational", PathType.LogName);
    var watcher = new EventLogWatcher(obj);
    watcher.EventRecordWritten += new
EventHandler<EventRecordWrittenEventArgs>(handleEvent);
    watcher.Enabled = true;

    while (!this.IsDisposed) {
        System.Threading.Thread.Sleep(1000);
    }
}
```

В цьому методі спочатку встановлюється listBox_events як місце, де будуть відображатися системні події. Потім створюється об'єкт EventLogQuery, що слідкує за діями оперативних подій Sysmon у Windows. Об'єкт EventLogWatcher створюється для відстеження об'єкта EventLogQuery. Далі обробник подій EventRecordWritten додається до об'єкта watcher, що викликає функцію handleEvent, коли запис події буде записано.

Метод start активує watcher і потім входить у цикл, що виконується доки форма не буде відхилена, тим самим забезпечуючи постійне відстеження системних подій.

3.3.2 Реалізація функції handleEvent для обробки подій системи

Центральним моментом в системі Windows WatchDog є обробка системних подій. Це досягається за допомогою функції handleEvent, яка викликається при кожному записі події в системному журналі. Функція handleEvent описана в додатку В.

Функція handleEvent є методом, що обробляє події, визначені в системному журналі подій Windows. Ця функція отримує подію у вигляді параметра e, перетворює її в XML формат, а потім аналізує її, щоб отримати метадані, такі як ідентифікатор процесу, ідентифікатор батьківського процесу, і так далі.

Код починається з того, що визначає, чи є отримана подія дійсною. Якщо ні, він записує повідомлення про помилку та припиняє обробку.

Потім йде перетворення події в XML формат та аналіз метаданих з цього XML. Це включає визначення таких деталей, як GUID процесу, ID процесу, GUID батьківського процесу та ID батьківського процесу.

У випадку, якщо ідентифікатор процесу або GUID процесу відсутні, метод просто повертає контроль, записуючи попередження.

Наступний блок коду обробляє ситуацію, коли ідентифікатор батьківського процесу або GUID батьківського процесу відсутні. У такому випадку він шукає батьківський запис події, і якщо його не знайдено, додає новий запис події в корінь. Якщо батьківський запис події знайдено, він додає новий запис події як дочірній елемент до цього батьківського запису.

Якщо ідентифікатор батьківського процесу та GUID батьківського процесу присутні, він знову шукає батьківський запис події. Якщо його не знайдено, додає новий запис події в корінь, а в іншому випадку - додає новий запис події як дочірній до батьківського запису.

У всіх випадках, коли створюється новий запис події, його деталі заносяться в журнал для відстеження.

Останній блок коду обробляє будь-які винятки, що виникають під час виконання цього методу, записуючи їх в журнал помилок.

3.4 Використання класу Program для ініціалізації додатку

Важливим аспектом розробки програмного забезпечення є ініціалізація додатку. Це перший крок, який відбувається при запуску програми. У контексті додатку Windows WatchDog, ініціалізація включає створення об'єкта головної форми та запуск застосунку.

Процес ініціалізації розглянуто докладніше в підрозділах цього розділу, які включають використання класу Program для ініціалізації додатку та розробку віконних подій. Деталізація цих підрозділів надає чітке розуміння того, як програма ініціалізується та запускається.

В контексті програмування на .NET Framework, клас "Program" служить входом до додатку. Його основне завдання - запуск основної форми додатка. Для програми Windows WatchDog цей клас ініціалізує форму MainForm, як і у багатьох Windows Forms додатках.

```
static class Program {
    [STAThread]
    static void Main() {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new FormMain());
    }}
```

Метод Main - це вхідна точка додатка. Він встановлює декілька властивостей, які впливають на візуальний стиль та сумісність відображення тексту, а потім запускає головну форму за допомогою методу Application.Run. Параметром цього методу є екземпляр FormMain, що ініціює створення головного вікна програми.

3.5 Тестування програмного додатку

Для проведення тестування було створено batch скрипт, який виконує набір команд в командному рядку Windows (cmd.exe) для перевірки побудови дерев процесів, тобто звернення до компонентів ОС буде відбуватися в рамках процесу командного рядка. Розглянемо кожну команду скрипта

```
@echo off
echo Running cmd.exe test commands
net user /add TestUser TestPassword123
net localgroup administrators TestUser /add
net user /del TestUser
echo cmd.exe test commands finished
pause
```

@echo off: Ця команда призупиняє вивід всіх наступних команд в консолі. Це зазвичай робиться, щоб консоль виглядала чистіше, без зайвих даних.

1. net user /add TestUser TestPassword123: Команда net user використовується для управління обліковими записами користувачів.

2. net localgroup administrators TestUser /add: Команда net localgroup administrators додає користувача TestUser до локальної групи administrators.

3. net user /del TestUser: Команда net user з ключем /del видаляє обліковий запис TestUser.

4. pause: команда зупиняє виконання скрипту, що потрібно для перевірки правильності виконання скрипта перед закриттям терміналу.

Також для проведення дослідження були внесені зміни в конфігураційний файл в якому залишилися наступні типи подій для моніторингу (Process Create, Process Terminate, Registry Event, File Create, File Delete) та який зображено на рисунку 3.2.

```
<Sysmon schemaversion="4.82">
  <HashAlgorithms>MD5</HashAlgorithms>
  <EventFiltering>

    <ProcessCreate onmatch="exclude">

    </ProcessCreate>

    <ProcessTerminate onmatch="exclude">

    </ProcessTerminate>

    <RegistryEvent onmatch="exclude">

    </RegistryEvent>

    <FileCreate onmatch="exclude">

    </FileCreate>

    <FileDelete onmatch="exclude">

    </FileDelete>

  </EventFiltering>
</Sysmon>
```

Рисунок 3.2 – Конфігураційний файл

Після застосування конфігураційного файлу в Sysmon запускаємо програму, і після запускаємо batch скрипт для симуляції зловмисної діяльності, виконання якої представлено на рисунку 3.3.

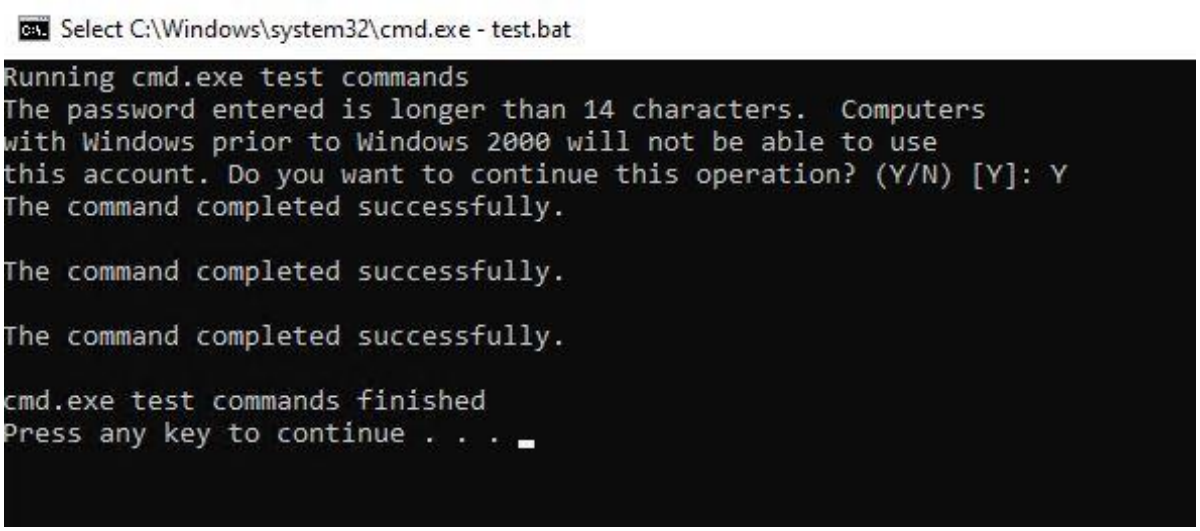


Рис.3.3 – Запуск batch скрипта

Програма відобразила наступний результат, що зображений на рисунку 3.4.

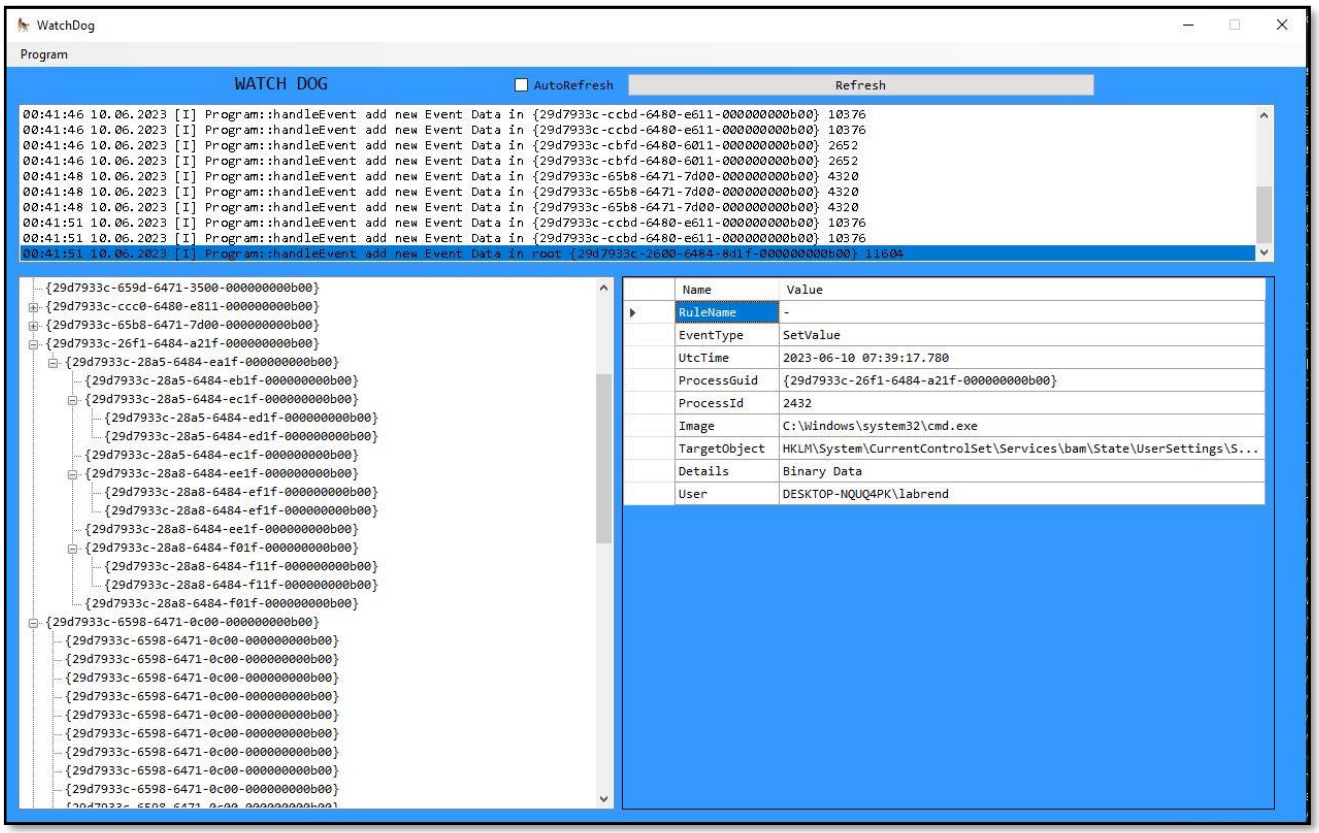


Рисунок 3.4 – Вивід програми

Як бачимо, було вибудувано дерево процесів, що зображено на рисунку 3.5.

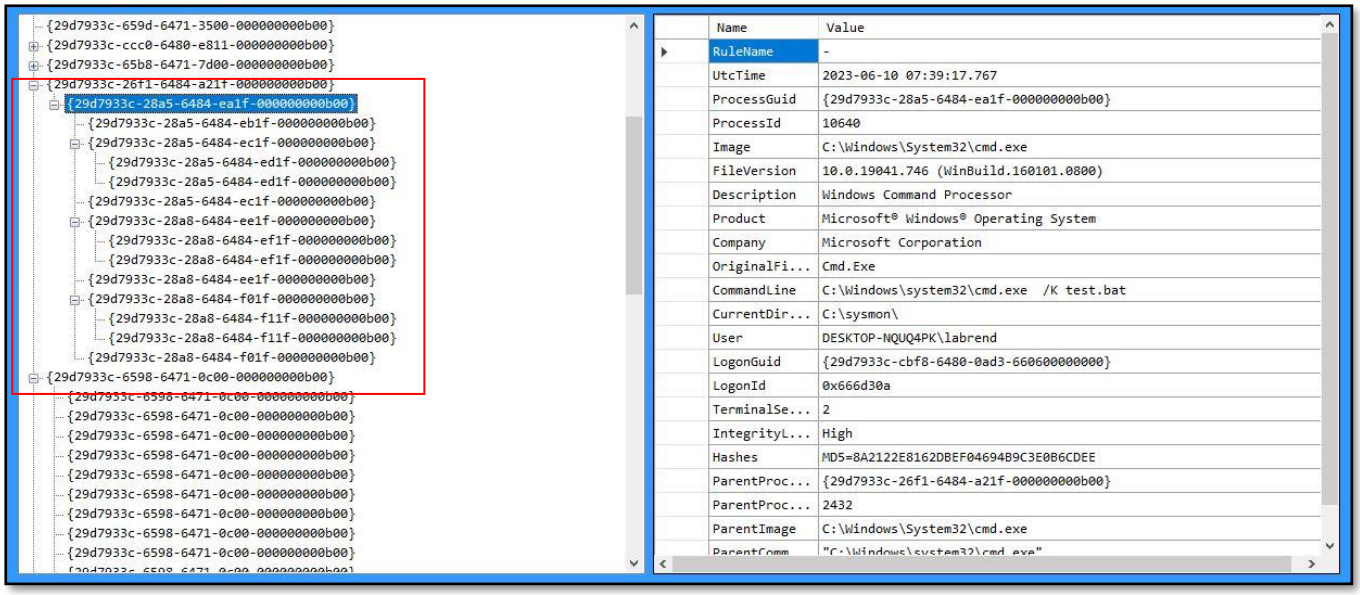


Рисунок 3.5 – Дерево процесів

Запуск test.bat з директорії “C:\sysmon”, а також подальше послідовне виконання команд batch скрипта та створення нового користувача через net1.exe, що зображено на рисунку 3.6.

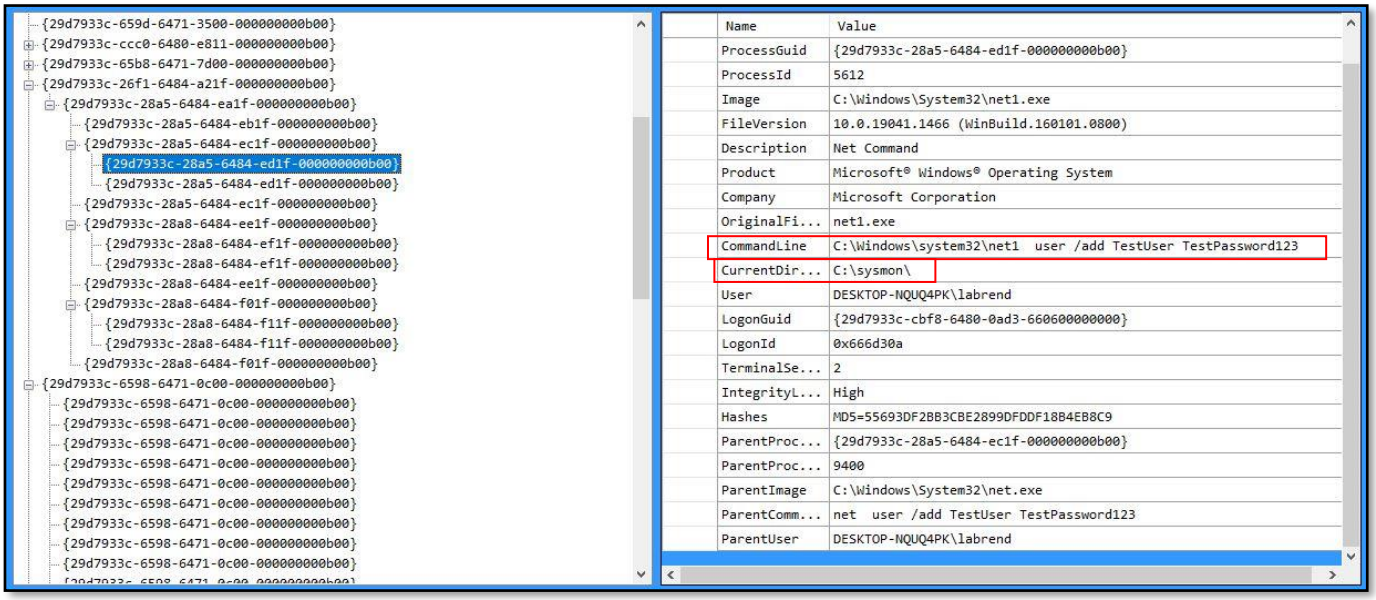


Рисунок 3.6 – Лог створення нового користувача через net.exe

Лог виконання команди localgroup /add для додавання створеного користувача до групи через net1.exe зображено на рисунку 3.7.

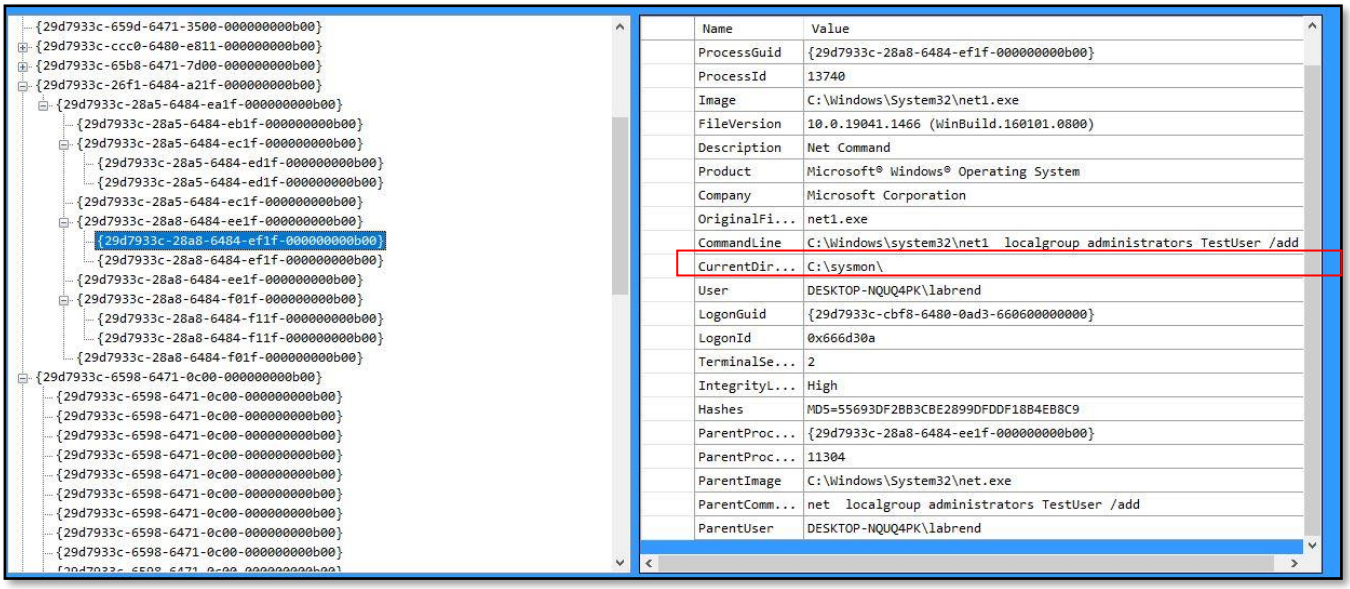


Рисунок 3.7 – Лог виконання команди додавання нового користувача в групу

Лог видалення користувача через net1.exe зображено на рисунку 3.8.

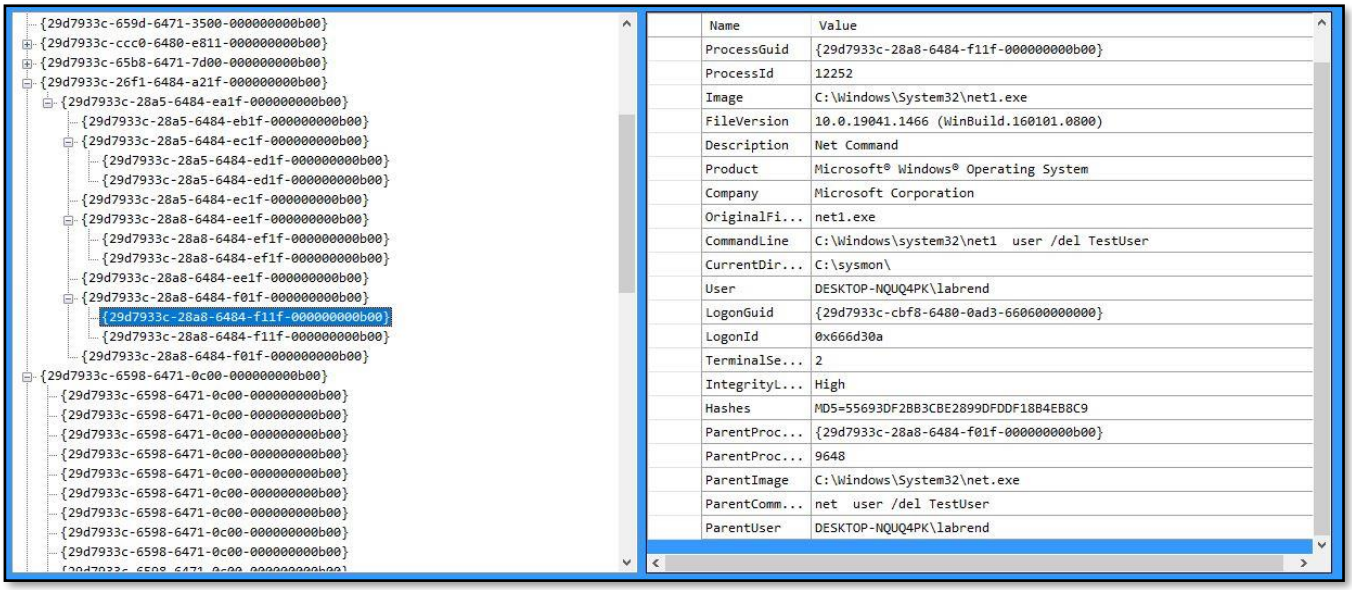


Рисунок 3.8 – Лог видалення користувача

А також супутні модифікації гілок регістру виконуваним файлом lsass.exe

lsass.exe (Local Security Authority Subsystem Service) - це процес ОС Windows, що виконує функцію оркестратора підсистем аутентифікації та відповідає за перевірку введених облікових даних, звіряє дії користувача з встановленими

політиками аутентифікації, обробляє різні варіанти входу в систему, генерує токени безпеки. Його лог зображено на рисунку 3.9.

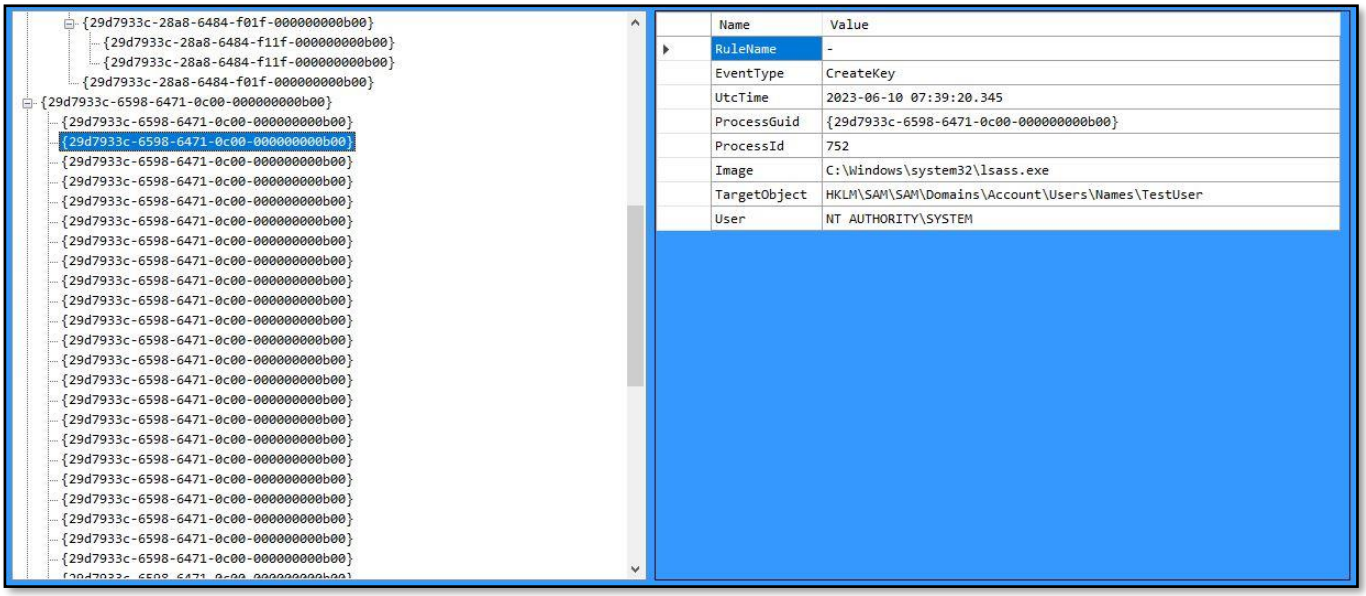


Рисунок 3.9 – Логи процесу lsass.exe

Також необхідно дослідити журнал Security. Інформація, що подається в ньому зображена на рисунку 3.10

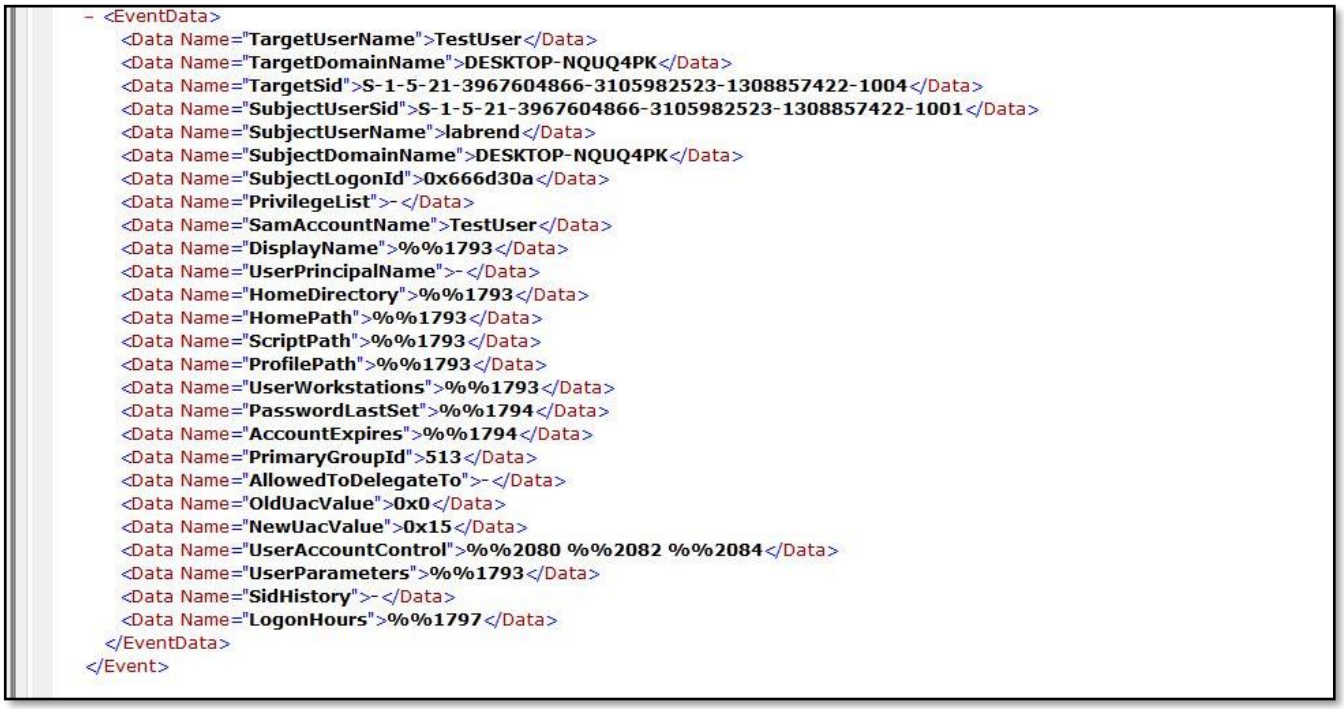


Рисунок 3.10 - Xml відображення логу Security.evtx

Як видно в журналі відображається факт створення нового користувача, але не відображається як саме це сталося.

В подальшому досліджуючи такі випадки можна модифікувати файл конфігурації створюючи фільтри по ключовим файлам та діям, наприклад взаємодія з net1.exe через cmd.exe з виконанням команд user /add. Це і є реалізація поведінкового аналізу в якому можна визначити результатом яких дій було здійснено операцію в рамках системи. А завдяки дуже деталізованому опису всіх дій можливо швидко та дуже детально провести розслідування та точно дізнатися що сталося, хто це зробив і в якій частині системи знаходиться джерело цих дій. Після розслідування доповнити конфігураційний файл для подальшої підсвітки подібних дій.

Висновки за розділом 3

В розділі 3 була розглянута реалізація програмного модуля для виявлення шкідливого додатку. Були детально вивчені методи взаємодії з користувачем, а також використання різних класів для обробки подій системи. Було розглянуто процес реалізації основного циклу відстеження подій системи та процес ініціалізації додатку.

Розроблений додаток демонструє високу ефективність в отриманні та аналізі системних подій, що дозволяє виявляти потенційно шкідливі додатки з високим рівнем точності. Також було проведено детальне тестування роботи програмного додатку. Було виконано порівняльний аналіз логування за допомогою звичайного журналу та за допомогою Sysmon.

Результати дослідження показали, що програмний модуль ефективно вирішує поставлені завдання та повністю задовольняє перелік вимог, що був сформований у другому розділі цієї роботи.

В перспективі логіку додатку можна використовувати в SIEM та EDR системах, реалізовуючи клієнт серверну архітектуру, в якій навантаження системи при зберіганні, обробці та візуалізації отриманої інформації перейде на серверну частину.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено та проведено тестування програмного модуля виявлення шкідливого ПЗ на основі поведінкового аналізу.

У першому розділі була проведена детальна оцінка методології виявлення шкідливого програмного забезпечення. Були досліджені основні та допоміжні механізми виявлення шкідливого ПЗ, такі як маркери компрометації та поведінкові сигнатури. та аналіз особливостей ОС Windows дозволили розробити вимоги до програмного. Були також розглянуті різні джерела маркерів компрометації та поведінкових сигнатур в ОС Windows. Детальний огляд існуючих рішень для виявлення шкідливого ПЗ

У другому розділі було спроєктовано архітектуру програмного модуля. Основними розділами цього етапу були вибір C# як основної мови програмування, використання Sysmon для збирання системних подій, та механізм поєднання логів в інциденти безпеки.

У третьому розділі була здійснена реалізація та тестування програмного модуля. Були вивчені різні аспекти взаємодії з користувачем, використання класів для обробки подій системи, ініціалізація додатку та його тестування. Зокрема було проведено порівняння роботи модуля з журналом подій Windows і Sysmon, результати якого показали вищу ефективність використання Sysmon.

Результати розробки та тестування свідчать про те, що програмний модуль ефективно вирішує поставлені завдання, демонструючи високий рівень надійності та ефективності в процесі виявлення шкідливих додатків. Це підтверджує високу практичну цінність розробленого додатку.

В перспективі саме така логіка побудови поведінкового аналізу для виявлення шкідливого програмного забезпеченні, що базується на його поведінці може використовуватися в інших безпекових рішеннях таких як SIEM та EDR

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What Is the Windows Event Viewer, and How Can I Use It? [Electronic resource]:https://www.howtogeek.com/123646/htg-explains-what-the-windows-event-viewer-is-and-how-you-can-use-it/

2. Windows Security Log Events [Electronic resource]: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx

3. Windows event log [Electronic resource]: https://www.techtarget.com/searchwindowsserver/definition/Windows-event-log

4. SYSMON [Electronic resource]: https://learn.microsoft.com/pl-pl/sysinternals/downloads/sysmon

5. Threat-hunting using Sysmon – Advanced Log Analysis for Windows [Electronic resource]: https://www.socinvestigation.com/threat-hunting-using-sysmon-advanced-log-analysis-for-windows/

6. Getting Started With Sysmon [Electronic resource]: https://www.blackhillsinfosec.com/getting-started-with-sysmon/

7. How to use Event Viewer on Windows 10 [Electronic resource]: https://www.windowscentral.com/how-use-event-viewer-windows-10

8. Using Wazuh to monitor Sysmon events [Electronic resource]: https://wazuh.com/blog/using-wazuh-to-monitor-sysmon-events/

9. Windows operating system security [Electronic resource]: https://learn.microsoft.com/en-us/windows/security/operating-system-security/

10. Operating Systems Architecture [Electronic resource]: http://cis2.oc.ctc.edu/oc_apps/Westlund/xbook/xbook.php?unit=04&proc=page&numb=1

11. THE NT ARCHITECTURE OF WINDOWS [Electronic resource]: https://blog.certcube.com/the-nt-architecture-of-windowss/

12. Architecture of Operating System [Electronic resource]: https://www.scaler.com/topics/architectures-of-operating-system/

13. What is PowerShell and how to use it: The ultimate tutorial [Electronic resource]: https://www.techtarget.com/searchwindowsserver/definition/PowerShell

14. Windows Management Instrumentation [Electronic resource]: https://learn.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page

15. How User Account Control works [Electronic resource]: https://learn.microsoft.com/en-us/windows/security/application-security/application-control/user-account-control/how-it-works

16. Difference between COM and DCOM [Electronic resource]: https://www.geeksforgeeks.org/difference-between-com-and-dcom/

17. Windows API index [Electronic resource]: https://learn.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

18. How to Use Windows API Knowledge to Be a Better Defender [Electronic resource]: https://redcanary.com/blog/windows-technical-deep-dive/

19. Task Scheduler for developers [Electronic resource]: https://learn.microsoft.com/en-us/windows/win32/taskschd/task-scheduler-start-page

20. Behavioral Analytics in Cybersecurity: Does It Work as Advertised? [Electronic resource]: https://www.esecurityplanet.com/applications/behavioral-analytics-cybersecurity-does-it-work/

21. What is Behavioral Analytics & How Does it Apply to Cybersecurity? [Electronic resource]: https://www.cybraics.com/blog/user-behavioral-analytics-the-new-cybersecurity-approach

22. FIRST-Shanghai-Sysmon-Search-Wataru-Takahashi.pdf [Electronic resource]: https://www.first.org/resources/papers/shanghai2018/FIRST-Shanghai-Sysmon-Search-Wataru-Takahashi.pdf

23. Sysmon Internals [Electronic resource]: https://exatrack.com/public/SysmonInternals.pdf

24. How to log data to the Windows Event Log in C# [Electronic resource]: https://www.infoworld.com/article/3598750/how-to-log-data-to-the-windows-event-log-in-csharp.html

25. Writing to the Windows Event Log with C# .NET [Electronic resource]: https://dotnetcodr.com/2017/10/13/writing-to-the-windows-event-log-with-c-net-2/

26. BitLocker overview [Electronic resource]: https://learn.microsoft.com/en-us/windows/security/operating-system-security/data-protection/bitlocker/

27. NTFS overview [Electronic resource]: https://learn.microsoft.com/en-us/windows-server/storage/file-server/ntfs-overview

28. Windows registry information for advanced users [Electronic resource]: https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users

29. 25 Microsoft Forms Tips and Tricks [Electronic resource]: https://www.powertechtips.com/microsoft-forms-tips/

30. Processes and Threads [Electronic resource]: https://learn.microsoft.com/en-us/windows/win32/procthread/processes-and-threads

31. Windows System Processes — An Overview For Blue Teams [Electronic resource]: https://nasbench.medium.com/windows-system-processes-an-overview-for-blue-teams-42fa7a617920

32. Cybersecurity Blue Team Guide [Electronic resource]: https://medium.com/@joshuaspeshock/cybersecurity-blue-team-guide-d6f51ba5c437

33. 4 Ways to Create a Local User Account in Windows 10 [Electronic resource]: https://www.makeuseof.com/ways-to-create-local-user-account-windows

34. Introduction to Windows Service Applications [Electronic resource]: https://learn.microsoft.com/en-us/dotnet/framework/windows-services/introduction-to-windows-service-applications

35. What is System.dll? [Electronic resource]: https://www.file.net/process/system.dll.html

36. АРХІТЕКТУРА WINDOWS: ОПИС, ВИДИ, СТРУКТУРА [Electronic resource]: https://government.com.ua/didzhytalizatsiia/arkhitektura-windows-opis-vidi-struktura.html

ДОДАТОК А

Код програмного модуля

Utils.cs

```
using System.Collections.Generic;
using System.Xml;

namespace WWatchDog {
   internal class Utils {
      public static Dictionary<string, string> getMetaDataFromXml(string xml) {
         Dictionary<string, string> ret = new Dictionary<string, string>();
         XmlDocument xDoc = new XmlDocument();
         xDoc.LoadXml(xml);
         XmlElement xRoot = xDoc.DocumentElement;
         foreach (XmlElement xnode in xRoot) {
            if (xnode.Name != "EventData") {
               continue;
            }
            foreach (XmlNode node in xnode.ChildNodes) {
               if (node.Attributes == null || node.Attributes.Count <= 0) {
                  continue;
               }
               ret.Add(node.Attributes[0].InnerText, node.InnerText);
            }
         }
         return ret;
      }
   }
}
```

EventRecord.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WWatchDog
{
   public class EventRecord
   {
      private EventRecord parent = null;
      private List<EventRecord> childred = new List<EventRecord>();
      private string guid = string.Empty;
      private int processId = -1;
      private int LEVEL { get; set; }
      private Dictionary<string, string> eventData = new Dictionary<string, string>();
```

```
    public EventRecord(string guid, int processId, Dictionary<string, string> eventData, EventRecord
parent)
    {
        this.guid = guid;
        this.processId = processId;
        this.parent = parent;
        this.eventData = eventData;
    }

    public string getGUID()
    {
        return this.guid;
    }

    public int getProcessID()
    {
        return this.processId;
    }

    public bool isEqual(int processId, string guid)
    {
        return processId == this.processId && this.guid == guid;
    }

    public List<EventRecord> getChildren()
    {
        return this.childred;
    }

    public void setParent(EventRecord record)
    {
        this.parent = record;
    }

    public EventRecord getParent()
    {
        return this.parent;
    }

    public void appendChild(EventRecord record)
    {
        this.childred.Add(record);
    }

    public Dictionary<string, string> getEventData()
    {
        return this.eventData;
    }

    public static EventRecord getParentRecord(List<EventRecord> root, int parentProcessId, string
parentGUID)
```

```
        {
            foreach (EventRecord record in root)
            {
                if (record.isEqual(parentProcessId, parentGUID))
                {
                    return record;
                }
                EventRecord ret = getParentRecord(record.getChildren(), parentProcessId, parentGUID);
                if (ret == null)
                {
                    continue;
                }
                return ret;
            }
            return null;
        }
    }
}
```

Logs.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WWatchDog {
    public class Log {
        private static ListBox listBox_events = null;

        public static void setListBox(ListBox listBox) {
            Log.listBox_events = listBox;
        }

        public static void info(string value) {
            message(Brushes.Black, "[I]", value);
        }

        public static void warning(string value) {
            message(Brushes.SeaGreen, "[W]", value);
        }

        public static void error(string value) {
            message(Brushes.Red, "[E]", value);
        }

        private static void message(Brush color, string marker, string value) {
            if (Log.listBox_events != null) {
                listBox_events.Invoke((MethodInvoker)delegate () {
                    while(listBox_events.Items.Count > 20) {
                        listBox_events.Items.RemoveAt(0);
                    }
```

```
                listBox_events.Items.Add(new MyListBoxItem(color, string.Format("{0} {1} {2}",
DateTime.Now.ToString("HH:mm:ss dd.MM.yyyy"), marker, value)));
                listBox_events.SelectedIndex = listBox_events.Items.Count - 1;
            });
        }
    }
}
```

FormMain.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics.Eventing.Reader;
using System.Threading;
using static System.Windows.Forms.VisualStyles.VisualStyleElement;
using System.Xml.Linq;

namespace WWatchDog
{
    public partial class FormMain : Form
    {
        public FormMain()
        {
            InitializeComponent();
            new Thread(delegate () {
                start();
            }).Start();
            new Thread(delegate () {
                autoRefresh();
            }).Start();
        }

        private void start() {
            Log.setListBox(this.listBox_events);
            var obj = new EventLogQuery("Microsoft-Windows-Sysmon/Operational", PathType.LogName);
            var watcher = new EventLogWatcher(obj);
            watcher.EventRecordWritten += new EventHandler<EventRecordWrittenEventArgs>(handleEvent);
            watcher.Enabled = true;
            while (!this.IsDisposed) {
                System.Threading.Thread.Sleep(1000);
            }
        }
```

```
private void autoRefresh() {
   while(!this.IsDisposed) {
      if (!treeView_process.IsHandleCreated || !checkBox_autorefresh.IsHandleCreated) {
         Thread.Sleep(1000);
         continue;
      }
      if (checkBox_autorefresh.Checked) {
         if (treeView_process.IsHandleCreated)  {
            treeView_process.Invoke((MethodInvoker)delegate ()
            {
               updateTreeView();
            });
         }
      }
      Thread.Sleep(1000);
   }
}

private void exitToolStripMenuItem_Click(object sender, EventArgs e) {
   this.Close();
}

private static List<EventRecord> ROOT_EVENTS = new List<EventRecord>();

private void updateTreeView() {
   treeView_process.Nodes.Clear();
   dataGridView_metaData.Rows.Clear();
   updateTreeViewNode(treeView_process.Nodes, ROOT_EVENTS);
}

private void updateTreeViewNode(TreeNodeCollection root, List<EventRecord> records) {
   foreach (EventRecord record in records) {
      int index = root.Add(new TreeNode(record.getGUID()));
      updateTreeViewNode(root[index].Nodes, record.getChildren());
   }
}

private void handleEvent(object obj, EventRecordWrittenEventArgs e)
{
   try
   {
      var record = e.EventRecord;
      if (record != null)
      {
         string xml = record.ToXml();
         Dictionary<string, string> dict = Utils.getMetaDataFromXml(xml);
         var processGuid = string.Empty;
         var processId = string.Empty;
         var parentProcessGuid = string.Empty;
         var parentProcessId = string.Empty;
         if (dict.ContainsKey("ProcessGuid"))
         {
```

```
                processGuid = dict["ProcessGuid"].ToString();
            }
            if (dict.ContainsKey("ProcessId"))
            {
                processId = dict["ProcessId"].ToString();
            }
            if (dict.ContainsKey("ParentProcessGuid"))
            {
                parentProcessGuid = dict["ParentProcessGuid"].ToString();
            }
            if (dict.ContainsKey("ParentProcessId"))
            {
                parentProcessId = dict["ParentProcessId"].ToString();
            }
            if (string.IsNullOrEmpty(processGuid) || string.IsNullOrEmpty(processId))
            {
                Log.warning("Program::handleEvent skip event because no process id");
                return;
            }
            if (string.IsNullOrEmpty(parentProcessGuid) || string.IsNullOrEmpty(parentProcessId))
            {
                EventRecord parentRecord = EventRecord.getParentRecord(ROOT_EVENTS,
Convert.ToInt32(processId), processGuid);
                if (parentRecord == null)
                {
                    ROOT_EVENTS.Add(new EventRecord(processGuid, Convert.ToInt32(processId), dict,
null));
                    Log.info(string.Format("Program::handleEvent add new Event Data in root {0} {1}",
processGuid, processId));
                    return;
                }
                EventRecord perentPerentRecord = parentRecord.getParent();
                if (perentPerentRecord == null)
                {
                    parentRecord.appendChild(new EventRecord(processGuid, Convert.ToInt32(processId),
dict, parentRecord));
                    Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId));
                    return;
                }
                perentPerentRecord.appendChild(new EventRecord(processGuid,
Convert.ToInt32(processId), dict, perentPerentRecord));
                Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId));
                return;
            }
            else
            {
                EventRecord parentRecord = EventRecord.getParentRecord(ROOT_EVENTS,
Convert.ToInt32(parentProcessId), parentProcessGuid);
                if (parentRecord == null)
                {
```

```
                    ROOT_EVENTS.Add(new EventRecord(processGuid, Convert.ToInt32(processId), dict,
null));
                    Log.info(string.Format("Program::handleEvent add new Event Data in root {0} {1}",
processGuid, processId));
                }
                else
                {
                    parentRecord.appendChild(new EventRecord(processGuid, Convert.ToInt32(processId),
dict, parentRecord));
                    Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId, parentRecord.getGUID(), parentRecord.getProcessID()));
                }
            }
        }
        else
        {
            Log.error("Program::handleEvent the event instance was null.");
        }
    }
    catch (Exception ex)
    {
        Log.error("Program::handleEvent " + ex.ToString());
    }
}

private void listBox_events_DrawItem(object sender, DrawItemEventArgs e)
{
    MyListBoxItem item = listBox_events.Items[e.Index] as MyListBoxItem;
    if (item != null)
    {
        e.DrawBackground();
        Brush myBrush = item.ItemColor;
        e.Graphics.DrawString(item.Message, e.Font, myBrush, e.Bounds, StringFormat.GenericDefault);
        e.DrawFocusRectangle();
    }
}

private void button_refresh_Click(object sender, EventArgs e) {
    updateTreeView();
}

private List<int> getIndexNode(TreeNode node) {
    List<int> ret = new List<int> { node.Index };
    while(node.Parent != null) {
        node = node.Parent;
        ret.Add(node.Index);
    }
    ret.Reverse();
    return ret;
}

private EventRecord getEventRecord(List<EventRecord> root, List<int> nodesIndex) {
```

```
            EventRecord record = null;
            List<EventRecord> records = root;
            foreach (int index in nodesIndex) {
                record = records[index];
                records = record.getChildren();
            }
            return record;
        }

        private void treeView_process_AfterSelect(object sender, TreeViewEventArgs e)
        {
            dataGridView_metaData.Rows.Clear();
            var node = treeView_process.SelectedNode;
            if (node == null) {
                return;
            }
            EventRecord record = getEventRecord(ROOT_EVENTS, getIndexNode(node));
            if (record == null) {
                return;
            }
            foreach(KeyValuePair<string, string> eventData in record.getEventData()) {
                dataGridView_metaData.Rows.Add(eventData.Key, eventData.Value);
            }
        }

        private void treeView_process_DrawNode(object sender, DrawTreeNodeEventArgs e)
        {

        }
    }
}
```

MyListBoxItem.cs

```
using System.Drawing;

namespace WWatchDog {
    public class MyListBoxItem {
        public MyListBoxItem(Brush c, string m) {
            ItemColor = c;
            Message = m;
        }
        public Brush ItemColor { get; set; }
        public string Message { get; set; }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WWatchDog
{
   internal static class Program
   {

      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new FormMain());
      }
   }
}
```

FormMain.Designer.cs

```
namespace WWatchDog
{
   partial class FormMain
   {
      private System.ComponentModel.IContainer components = null;
      protected override void Dispose(bool disposing)
      {
         if (disposing && (components != null))
         {
            components.Dispose();
         }
         base.Dispose(disposing);
      }

      private void InitializeComponent()
      {
         System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(FormMain));
         this.treeView_process = new System.Windows.Forms.TreeView();
         this.label1 = new System.Windows.Forms.Label();
         this.dataGridView_metaData = new System.Windows.Forms.DataGridView();
         this.MDName = new System.Windows.Forms.DataGridViewTextBoxColumn();
         this.MDValue = new System.Windows.Forms.DataGridViewTextBoxColumn();
         this.button_refresh = new System.Windows.Forms.Button();
         this.menuStrip1 = new System.Windows.Forms.MenuStrip();
```

```
            this.programToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.exitToolStripMenuItem = new System.Windows.Forms.ToolStripMenuItem();
            this.checkBox_autorefresh = new System.Windows.Forms.CheckBox();
            this.listBox_events = new System.Windows.Forms.ListBox();
            ((System.ComponentModel.ISupportInitialize)(this.dataGridView_metaData)).BeginInit();
            this.menuStrip1.SuspendLayout();
            this.SuspendLayout();
            this.treeView_process.Location = new System.Drawing.Point(13, 227);
            this.treeView_process.Name = "treeView_process";
            this.treeView_process.Size = new System.Drawing.Size(578, 517);
            this.treeView_process.DrawNode += new
System.Windows.Forms.DrawTreeNodeEventHandler(this.treeView_process_DrawNode);
            this.treeView_process.AfterSelect += new
System.Windows.Forms.TreeViewEventHandler(this.treeView_process_AfterSelect);
            this.label1.AutoSize = true;
            this.label1.Font = new System.Drawing.Font("Consolas", 14.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.label1.Location = new System.Drawing.Point(219, 29);
            this.label1.Name = "label1";
            this.label1.Size = new System.Drawing.Size(129, 28);
            this.label1.TabIndex = 1;
            this.label1.Text = "WATCH DOG";
            this.dataGridView_metaData.AllowUserToAddRows = false;
            this.dataGridView_metaData.AllowUserToDeleteRows = false;
            this.dataGridView_metaData.BackgroundColor = System.Drawing.SystemColors.MenuHighlight;
            this.dataGridView_metaData.ColumnHeadersHeightSizeMode =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
            this.dataGridView_metaData.Columns.AddRange(new
System.Windows.Forms.DataGridViewColumn[] {
            this.MDName,
            this.MDValue});
            this.dataGridView_metaData.Location = new System.Drawing.Point(598, 227);
            this.dataGridView_metaData.Name = "dataGridView_metaData";
            this.dataGridView_metaData.ReadOnly = true;
            this.dataGridView_metaData.RowHeadersWidth = 51;
            this.dataGridView_metaData.Size = new System.Drawing.Size(633, 517);
            this.MDName.HeaderText = "Name";
            this.MDName.MinimumWidth = 6;
            this.MDName.Name = "MDName";
            this.MDName.ReadOnly = true;
            this.MDName.Width = 300;
            this.MDValue.HeaderText = "Value";
            this.MDValue.MinimumWidth = 6;
            this.MDValue.Name = "MDValue";
            this.MDValue.ReadOnly = true;
            this.MDValue.Width = 300;
            this.button_refresh.Location = new System.Drawing.Point(603, 30);
            this.button_refresh.Name = "button_refresh";
            this.button_refresh.Size = new System.Drawing.Size(454, 23);
            this.button_refresh.TabIndex = 3;
            this.button_refresh.Text = "Refresh";
            this.button_refresh.UseVisualStyleBackColor = true;
```

```
            this.button_refresh.Click += new System.EventHandler(this.button_refresh_Click);
            this.menuStrip1.ImageScalingSize = new System.Drawing.Size(20, 20);
            this.menuStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {
            this.programToolStripMenuItem});
            this.menuStrip1.Location = new System.Drawing.Point(0, 0);
            this.menuStrip1.Name = "menuStrip1";
            this.menuStrip1.Size = new System.Drawing.Size(1261, 28);
            this.menuStrip1.TabIndex = 4;
            this.menuStrip1.Text = "menuStrip1";
            this.programToolStripMenuItem.DropDownItems.AddRange(new
System.Windows.Forms.ToolStripItem[] {
            this.exitToolStripMenuItem});
            this.programToolStripMenuItem.Name = "programToolStripMenuItem";
            this.programToolStripMenuItem.Size = new System.Drawing.Size(80, 24);
            this.programToolStripMenuItem.Text = "Program";
            this.exitToolStripMenuItem.Name = "exitToolStripMenuItem";
            this.exitToolStripMenuItem.Size = new System.Drawing.Size(116, 26);
            this.exitToolStripMenuItem.Text = "Exit";
            this.exitToolStripMenuItem.Click += new
System.EventHandler(this.exitToolStripMenuItem_Click);
            this.checkBox_autorefresh.AutoSize = true;
            this.checkBox_autorefresh.Checked = true;
            this.checkBox_autorefresh.CheckState = System.Windows.Forms.CheckState.Checked;
            this.checkBox_autorefresh.Location = new System.Drawing.Point(494, 33);
            this.checkBox_autorefresh.Name = "checkBox_autorefresh";
            this.checkBox_autorefresh.Size = new System.Drawing.Size(130, 24);
            this.checkBox_autorefresh.TabIndex = 5;
            this.checkBox_autorefresh.Text = "AutoRefresh";
            this.checkBox_autorefresh.UseVisualStyleBackColor = true;
            this.listBox_events.DrawMode = System.Windows.Forms.DrawMode.OwnerDrawFixed;
            this.listBox_events.FormattingEnabled = true;
            this.listBox_events.ItemHeight = 15;
            this.listBox_events.Location = new System.Drawing.Point(13, 60);
            this.listBox_events.Name = "listBox_events";
            this.listBox_events.Size = new System.Drawing.Size(1218, 154);
            this.listBox_events.TabIndex = 6;
            this.listBox_events.DrawItem += new
System.Windows.Forms.DrawItemEventHandler(this.listBox_events_DrawItem);
            this.AutoScaleDimensions = new System.Drawing.SizeF(9F, 19F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.BackColor = System.Drawing.SystemColors.MenuHighlight;
            this.ClientSize = new System.Drawing.Size(1261, 756);
            this.Controls.Add(this.listBox_events);
            this.Controls.Add(this.checkBox_autorefresh);
            this.Controls.Add(this.button_refresh);
            this.Controls.Add(this.dataGridView_metaData);
            this.Controls.Add(this.label1);
            this.Controls.Add(this.treeView_process);
            this.Controls.Add(this.menuStrip1);
            this.Font = new System.Drawing.Font("Consolas", 9.75F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(0)));
            this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D;
```

```
        this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
        this.MainMenuStrip = this.menuStrip1;
        this.MaximizeBox = false;
        this.Name = "FormMain";
        this.StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
        this.Text = "WatchDog";
        ((System.ComponentModel.ISupportInitialize)(this.dataGridView_metaData)).EndInit();
        this.menuStrip1.ResumeLayout(false);
        this.menuStrip1.PerformLayout();
        this.ResumeLayout(false);
        this.PerformLayout();
    }

    private System.Windows.Forms.TreeView treeView_process;
    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.DataGridView dataGridView_metaData;
    private System.Windows.Forms.Button button_refresh;
    private System.Windows.Forms.MenuStrip menuStrip1;
    private System.Windows.Forms.ToolStripMenuItem programToolStripMenuItem;
    private System.Windows.Forms.ToolStripMenuItem exitToolStripMenuItem;
    private System.Windows.Forms.CheckBox checkBox_autorefresh;
    private System.Windows.Forms.ListBox listBox_events;
    private System.Windows.Forms.DataGridViewTextBoxColumn MDName;
    private System.Windows.Forms.DataGridViewTextBoxColumn MDValue;
    }
}
```

ДОДАТОК Б

FormMain.cs

```
private void handleEvent(object obj, EventRecordWrittenEventArgs e)
    {
        try
        {
            var record = e.EventRecord;
            if (record != null)
            {
                string xml = record.ToXml();
                Dictionary<string, string> dict = Utils.getMetaDataFromXml(xml);

                var processGuid = string.Empty;
                var processId = string.Empty;
                var parentProcessGuid = string.Empty;
                var parentProcessId = string.Empty;

                if (dict.ContainsKey("ProcessGuid"))
                {
                    processGuid = dict["ProcessGuid"].ToString();
                }
                if (dict.ContainsKey("ProcessId"))
                {
                    processId = dict["ProcessId"].ToString();
                }
                if (dict.ContainsKey("ParentProcessGuid"))
                {
                    parentProcessGuid = dict["ParentProcessGuid"].ToString();
                }
                if (dict.ContainsKey("ParentProcessId"))
                {
                    parentProcessId = dict["ParentProcessId"].ToString();
                }

                if (string.IsNullOrEmpty(processGuid) || string.IsNullOrEmpty(processId))
                {
                    Log.warning("Program::handleEvent skip event because no process id");
                    return;
                }

                if (string.IsNullOrEmpty(parentProcessGuid) || string.IsNullOrEmpty(parentProcessId))
                {
                    EventRecord parentRecord = EventRecord.getParentRecord(ROOT_EVENTS,
Convert.ToInt32(processId), processGuid);
                    if (parentRecord == null)
                    {
                        ROOT_EVENTS.Add(new EventRecord(processGuid, Convert.ToInt32(processId), dict,
null));
                        Log.info(string.Format("Program::handleEvent add new Event Data in root {0} {1}",
processGuid, processId));
```

```
                    return;
                }
                EventRecord perentPerentRecord = parentRecord.getParent();
                if (perentPerentRecord == null)
                {
                    parentRecord.appendChild(new EventRecord(processGuid, Convert.ToInt32(processId),
dict, parentRecord));
                    Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId));
                    return;
                }
                perentPerentRecord.appendChild(new EventRecord(processGuid,
Convert.ToInt32(processId), dict, perentPerentRecord));
                Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId));
                return;
            }
            else
            {
                EventRecord parentRecord = EventRecord.getParentRecord(ROOT_EVENTS,
Convert.ToInt32(parentProcessId), parentProcessGuid);
                if (parentRecord == null)
                {
                    ROOT_EVENTS.Add(new EventRecord(processGuid, Convert.ToInt32(processId), dict,
null));
                    Log.info(string.Format("Program::handleEvent add new Event Data in root {0} {1}",
processGuid, processId));
                }
                else
                {
                    parentRecord.appendChild(new EventRecord(processGuid, Convert.ToInt32(processId),
dict, parentRecord));
                    Log.info(string.Format("Program::handleEvent add new Event Data in {0} {1}",
processGuid, processId, parentRecord.getGUID(), parentRecord.getProcessID()));
                }
            }
        }
        else
        {
            Log.error("Program::handleEvent the event instance was null.");
        }
    }
    catch (Exception ex)
    {
        Log.error("Program::handleEvent " + ex.ToString());
    }
}
```

ДОДАТОК В

Конфігураційний файл Sysmon.xml

```
<Sysmon schemaversion="4.82"> <!-- Версія схеми конфігураційного файлу -->
  <!-- Визначає алгоритми хешування, які будуть застосовуватися -->
  <HashAlgorithms>MD5</HashAlgorithms> <!-- Використовується MD5 алгоритм хешування -->
  <EventFiltering> <!-- Початок блоку фільтрації подій -->
      <!-- Виключає логування подій створення процесу -->
      <ProcessCreate onmatch="exclude">

      </ProcessCreate>
      <!-- Виключає логування подій зміни часу створення файлу -->
      <FileCreateTime onmatch="exclude">

      </FileCreateTime>
      <!-- Виключає логування подій мережевого з'єднання -->
      <NetworkConnect onmatch="exclude">

      </NetworkConnect>
      <!-- Виключає логування подій завершення процесу -->
      <ProcessTerminate onmatch="exclude">

      </ProcessTerminate>
      <!-- Виключає логування подій завантаження драйвера -->
      <DriverLoad onmatch="exclude">

      </DriverLoad>
      <!-- Виключає логування подій завантаження зображення -->
      <ImageLoad onmatch="exclude">

      </ImageLoad>
      <!-- Виключає логування подій створення віддаленого потоку -->
      <CreateRemoteThread onmatch="exclude">

      </CreateRemoteThread>
      <!-- Виключає логування подій зчитування сирих даних -->
      <RawAccessRead onmatch="exclude">

      </RawAccessRead>
      <!-- Виключає логування подій доступу до процесу -->
      <ProcessAccess onmatch="exclude">

      </ProcessAccess>
      <!-- Виключає логування подій створення файлу -->
      <FileCreate onmatch="exclude">

      </FileCreate>
      <!-- Виключає логування подій реєстру -->
      <RegistryEvent onmatch="exclude">
```

```
    </RegistryEvent>
    <!-- Виключає логування подій створення потоку хешу файлу -->
    <FileCreateStreamHash onmatch="exclude">

    </FileCreateStreamHash>
    <!-- Виключає логування подій каналу -->
    <PipeEvent onmatch="exclude">

    </PipeEvent>
    <!-- Виключає логування подій WMI -->
    <WmiEvent onmatch="exclude">

    </WmiEvent>
    <!-- Виключає логування подій DNS-запитів -->
    <DnsQuery onmatch="exclude">

    </DnsQuery>
    <!-- Виключає логування подій видалення файлів -->
    <FileDelete onmatch="exclude">

    </FileDelete>

  </EventFiltering> <!-- Кінець блоку фільтрації подій -->
</Sysmon>
```

ДОДАТОК Г
СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Тези-наукових конференцій

Нечипоренко І. Використання інструментів OSINT для виявлення технічних характеристик та конфігурації інфраструктури цільової організації / Іван Пархоменко, Лариса Мирутенко, Юлія Власюк, Нечипоренко Іван / VI Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-телекомунікаційних систем» (PCSITS) 27 квітня 2023, Київ, Україна, стр. 80-82.