Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка програми виявлення шкідливого програмного забезпечення на основі поведінкавого аналізу

Виконав: студент 2(4) курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Линдюк А.А.
(прізвище та ініціали)

Керівник: к.ф.- м.н., доц., доцент каф. МБІС
Шиян А. А.
(прізвище та ініціали)

«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Крупельницький Л.В.
(прізвище та ініціали)

«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“_____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Линдюку Артьому Анатолійовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи "Розробка програми виявлення шкідливого програмного забезпечення на основі поведінкавого аналізу"

Керівник роботи к.ф.- м.н., доц., доцент каф. МБІС Шиян А. А.
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від " 02 " червня 2025року № 80
2. Термін подання студентом роботи 28.05.2025 р.
3. Вихідні дані до роботи Набір поведінкових характеристик програмного забезпечення, журнал дій процесів у системі.

4. Зміст текстової частини:
1. Теоретичні основи виявлення шкідливого програмного забезпечення
2. Проєктування системи виявлення шкідливого ПЗ
3. Реалізація та тестування програми

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Презентація у форматі PowerPoint з ілюстраціями структури програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	доцент Шиян А. А		
Розділ II	доцент Шиян А. А		
Розділ III	доцент Шиян А. А		

7. Дата видачі завдання 18 лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент _____ **Линдюк А.А.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Шиян А.А.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У даній бакалаврській дипломній роботі розглянуто процес розробки програмного засобу для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу. Основну увагу приділено аналізу динамічної поведінки програм, яка дозволяє ідентифікувати нові та модифіковані види шкідливого ПЗ, що можуть залишатися невиявленими при використанні традиційних сигнатурних методів. У роботі проаналізовано існуючі підходи до поведінкового аналізу, розроблено архітектуру системи, реалізовано програму з використанням сучасних інструментів, а також проведено тестування для оцінки ефективності. Результати показали, що запропоноване рішення може успішно виявляти підозрілу активність у системі та слугувати основою для подальшої розробки інтелектуальних систем захисту.

ANNOTATION

This bachelor's thesis describes the process of developing a software tool for detecting malware based on behavioral analysis. The main attention is paid to the analysis of dynamic program behavior, which allows identifying new and modified types of malware that may remain undetected when using traditional signature methods. The paper analyzes existing approaches to behavioral analysis, develops the system architecture, implements the program using modern tools, and conducts testing to evaluate its effectiveness. The results showed that the proposed solution can successfully detect suspicious activity in the system and serve as a basis for further development of intelligent security systems.

ЗМІСТ

ВСТУП

Актуальність задачі. У сучасному цифровому світі кіберзагрози стають дедалі витонченішими, а шкідливе програмне забезпечення (malware) — все більш небезпечним. Традиційні методи виявлення, засновані на сигнатурному аналізі, вже не здатні забезпечити належний рівень безпеки, оскільки нові зразки шкідливого ПЗ з’являються щодня, часто з незначними змінами, які дозволяють їм обходити стандартні антивірусні механізми. У зв’язку з цим особливої актуальності набувають методи поведінкового аналізу, що дають змогу виявляти потенційно небезпечні програми не за їх зовнішніми ознаками, а за характерними діями у системі.

Поведінковий аналіз активно досліджується у сфері кібербезпеки як універсальний підхід до виявлення Zero-Day атак та поліморфного шкідливого ПЗ. Провідні світові компанії (наприклад, CrowdStrike, Palo Alto Networks, FireEye) та академічні установи розробляють системи, що використовують машинне навчання, евристику та аналіз активності для ідентифікації аномальної поведінки в інформаційних системах. На відміну від сигнатурних методів, поведінковий підхід дозволяє виявити загрозу ще до того, як вона буде класифікована та додана до бази.

Попри значний прогрес, у поведінковому аналізі залишаються нерозв’язані проблеми: високе навантаження на ресурси, складність відокремлення шкідливої активності від легітимної, можливість обходу sandbox-аналізу за допомогою антианалітичних технік. Відтак, постає потреба в розробці більш ефективних, точних та продуктивних рішень, які здатні працювати в реальному часі та адаптуватися до змін у поведінці загроз.

Мета і задачі дослідження є розробка програмного інструменту, що здійснює виявлення шкідливого програмного забезпечення на основі поведінкового аналізу.

- **Задачі дослідження:**

1. Проаналізувати сучасні методи виявлення шкідливого ПЗ, зокрема сигнатурні, евристичні, поведінкові та засновані на машинному навчанні, визначити їх переваги, недоліки та кількісні характеристики.

2. Розробити алгоритм поведінкового аналізу, що враховує характерні дії програм (доступ до реєстру, мережеву активність, зміну системних файлів тощо), з урахуванням обмежень продуктивності та точності.

3. Побудувати програмний засіб, що реалізує запропонований підхід: збір поведінкових ознак, їх попередню обробку, класифікацію активності та відображення результатів користувачу.

Об'єктом дослідження є процес виявлення шкідливого програмного забезпечення в комп’ютерних системах.

Предмет дослідження є методи та засоби поведінкового аналізу програмної активності з метою ідентифікації потенційно шкідливих дій.

Новизна дослідження полягає у розробці додатку для поведінкового моніторингу процесів, який дозволяє виявляти потенційно небезпечне програмне забезпечення без необхідності використовувати базу сигнатур. На відміну від традиційних антивірусних систем, запропонований підхід базується на аналізі дій програм у реальному часі.

Практична цінність роботи полягає у розробці ефективного програмного засобу для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу, який може бути інтегрований у системи кіберзахисту для підвищення їхньої здатності протидіяти новим та модифікованим загрозам, включаючи Zero-day атаки.

1 ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглядаються базові поняття, пов’язані з шкідливим програмним забезпеченням (ПЗ), його видами та методами виявлення. Дається класифікація шкідливих програм (віруси, трояни, руткіти, шпигунське ПЗ тощо) та аналізуються основні підходи до їх детектування: сигнатурний аналіз, евристичний підхід, машинне навчання та, особливо, поведінковий аналіз.

Також у розділі проводиться огляд сучасних систем захисту, які використовують поведінковий підхід, описуються їхні переваги та недоліки. Визначаються основні виклики, з якими стикаються дослідники й розробники в процесі створення ефективних засобів виявлення загроз.

1.1 Класифікація шкідливого програмного забезпечення

Шкідливе програмне забезпечення (ПЗ), або malware, — це узагальнена назва для будь-яких програм, які спеціально розроблені для несанкціонованого втручання в роботу комп’ютерної системи, збору конфіденційної інформації або порушення її функціонування[1]. Основні типи шкідливого ПЗ наведено в таблиці 1.1.

Таблиця 1.1 - Основні типи шкідливого програмного забезпечення[2]

Тип шкідника	Опис
Вірус	Самовідтворюваний код, що приєднується до файлів
Черв’як	Поширюється через мережу без втручання користувача
Троян	Маскується під легітимне ПЗ, виконує шкідливі дії
Шпигунське ПЗ	Збирає дані про користувача без його згоди
Кейлогер	Реєструє натискання клавіш

Продовження таблиці 1.1

Руткіт	Приховує наявність інших шкідливих програм
Рансомвар	Блокує систему або дані, вимагаючи викуп

Сучасні шкідливі програми часто поєднують кілька функцій, що ускладнює їх виявлення та класифікацію. Принцип роботи шкідливих програм залежить від їх типу і призначення. Наприклад віруси чіпляються до звичайних файлів і потребують взаємодії з користувачем для свого розповсюдження[2].

З огляду на постійний розвиток IT-середовища, класифікація шкідливого ПЗ є критично важливою для розробки ефективних методів захисту.

Основні критерії класифікації:

- **За способом зараження[3]**
  1. Файлові віруси — прикріплюються до виконуваних файлів (.exe, .dll).
  2. Завантажувальні віруси — уражають завантажувальний сектор.
  3. Сценарні віруси — використовують скрипти (наприклад, VBA макроси).
- **За механізмом дії[4]**
  1. Трояни — маскуються під легітимне ПЗ, виконуючи приховані дії.
  2. Хробаки — самореплікація через мережі.
  3. Spyware — шпигунські модулі для крадіжки даних.
  4. Ransomware — шифрування даних із вимогою викупу.
  5. Rootkit — приховує шкідливу активність на низькому рівні.
  6. Keylogger — фіксує натискання клавіш.
  7. Backdoor — створює обхід автентифікації.
  8. Adware — показує нав’язливу рекламу.

- **За мотивацією атаки [5]**
  1. Фінансова (крадіжка коштів, криптовалют).
  2. Шпигунська (державні або корпоративні цілі).
  3. Деструктивна (знищення даних або сервісів).
- **За ціллю впливу [6]**
  1. Botnet — зараження систем для DDoS-атак.
  2. Спеціалізоване ПЗ для знищення промислових систем (Stuxnet тощо).

Рисунок 1.1 – Класифікація шкідливих програм за принципами розповсюдження та функціонування[7].

Як і інші шкідливі програми, троянські програми можуть виконувати зазначені вище деструктивні дії, але в основному їх використовують для виконання шпигунських дій(рис. 1.1).

Значна частина шкідливих програм у початкові періоди зараження не виконують деструктивних дій, а лише розмножуються. Це так звана пасивна фаза їхнього існування. Через певний час, у визначений день або по команді з комп'ютера в мережі шкідливі програми починають виконувати деструктивні дії – переходять в активну фазу свого існування.

Сучасні шкідливі програми призначені для підключення до мережі для виконання різних функцій. Найвідоміший – маяк, коли нещодавно

скомпрометована система передає сигнал на сервери зловмисника, щоб підтвердити, що вона є новою жертвою, готова до виконання команд[7].

Шпигунське використовується людьми, які хочуть стежити за діями на комп'ютері людей, особисто відомих їм. Шпигунське програмне забезпечення легко видалити[8].

Таблиця 1.2 – Сучасні шкідливі програми.

Зразок кампанія	Клас загрози	Ціль і тактика	Рік(и) активності	Ключові особливості	Джерела
LockBit 3.0 / 4.0	Ransom-as-a-Service	Шифрує дані та публікує їх у разі несплати викупу; після масштабної спец-операції NCA/FBI у лютому 2024 група швидко випустила новий самопоширюваний варіант	2020-2025	Автоматичне поширення в мережі, 2-й канал зв’язку через соцмережі	[9]
BlackCat (ALPHV)	Рансомвар + крадіжка даних	Націлюється на критичні галузі (охорона здоров’я, енергетика); крадіжка та подвійний шантаж	2021-2025	Перший RaaS на Rust, власний «вітринний» сайт-донос для жертв	[10]
Industroyer2	ICS SCADA-шкідник	Виведення з ладу енергомереж; застосований проти об’єктів енергетики України	2022	Прямі команди IEC-104 до польових пристроїв, мінімальний «шум»	[11]

Продовження таблиці 1.2

SUNBURST (SolarWinds)	Supply-chain backdoor	Змінений апдейт Orion IT-платформи давав доступ урядовим мережам США	2020–2021	Селект-список жертв, багаторівневий C2-ланцюжок	[12]
TrickBot AnchorBazar	Банківський троян модульний бекдор	Фінансова крадіжка + лінія до Conti/Quantum RaaS	2016-2024	Динамічне завантаження модулів, переходить у fileless-режим	[13]
Mirai (суч. GeoVision варіант)	IoT-ботнет	DDoS через камери / NAS, нові CVE-2024-6047 & -11120	2016-2025	Автоматичний експлойт CGI-сценаріїв із Hard-coded паролями	[14]
FluBot	Android SMS-черв’як	Фішингове SMS-посилання крадіжка банківських реквизитів	2020-2023	Overlay-екрани поверх банківських застосунків	[15]

- **Головні тенденції 2023-2025 рр.**

1. Комерціалізація шкідливого ПЗ. RaaS-і та PaaS-платформи знижують «поріг входу» для кіберзлочинців і пришвидшують еволюцію зразків [9].

2. Shift-Left-Attack: атаки ланцюга постачання (supply-chain) — компрометація легітимних оновлень ПО, як у випадку з SolarWinds [12].

3. Fileless-та ін-процес-малвар. Акцент на пам’яткові навантаження (in-memory) і Living-off-the-Land Binary (LoLBin) техніки, що ускладнює сигнатурний детект.

4. Націлювання на OT/ICS. Поява Industroyer2 та Triton-подібних шкідників підтверджує зростання ризиків для критичної інфраструктури [11].

5. Мобільні та IoT-екосистеми. FluBot, SharkBot та Mirai-варіанти активно експлуатують слабку безпеку смарт-пристроїв та смартфонів [14],[15].

6. AI-асистоване шкідливе ПЗ. Виявляються PoC-малвари, що застосовують ML-моделі для ухилення від детекції або автоматичного підбору фішингових тем.

Багато сучасних шкідливих програм поєднують кілька характеристик одночасно. Наприклад, одна програма може містити троян, руткіт і бекдор, одночасно викрадати дані і відкривати доступ до системи для інших атак. Це ускладнює виявлення та потребує застосування поведінкового аналізу та методів машинного навчання[13].

Для об'єктивного порівняння ефективності різних методів виявлення шкідливого програмного забезпечення доцільно використовувати кількісні характеристики. Такі показники, як точність виявлення, рівень хибнопозитивних/хибнонегативних спрацьовувань, час реакції, а також продуктивність у режимі реального часу, дозволяють оцінити практичну придатність методів у конкретних умовах експлуатації.

- **Точність виявлення (Detection Accuracy)** [16]

Один із ключових показників ефективності системи. Згідно з даними AV-TEST, точність виявлення для різних методів становить:

1. Сигнатурний аналіз — 85–90%
1. Поведінковий аналіз — 92–96%
2. Методи на базі ML — 96–99%

Ці показники базуються на тестуванні з використанням масиву з 30 тис. зразків для Windows та Linux .

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (1.1)$$

де:

TP — кількість правильно виявлених шкідливих зразків;

TN — кількість правильно виявлених нешкідливих зразків;

FP — кількість хибнопозитивних спрацьовувань;

FN — кількість хибнонегативних спрацьовувань[16];

Формула (1.1) використовується для оцінки загальної ефективності системи виявлення загроз. Чим ближче значення до 1 (або 100%), тим точніше працює система.

- **Час реакції (Response Time)** [17]

Середній час від запуску підозрілого процесу до його виявлення.

1. Сигнатурні методи — миттєво (0.1–0.3 с), але тільки відомі загрози;
2. Поведінкові методи — 2–10 секунд (залежно від порогу активації);
3. ML класифікатори — 1–2 с при оптимізації;

Ці дані отримані з досліджень, що аналізують час реакції систем виявлення шкідливого ПЗ.

- **Рівень хибнопозитивних/хибнонегативних спрацьовувань)** [17]

1. FP (False Positive) – нормальний процес позначений як шкідливий;
2. FN (False Negative) – шкідливий процес не виявлено;

Згідно з дослідженнями:

1. Сигнатурні системи: FP ≈ 1–3%, FN ≈ 10–15%;
2. Поведінкові: FP ≈ 5–8%, FN ≈ 3–5%;
3. ML: FP ≈ 2–4%, FN ≈ 2–3%;

Ці показники базуються на аналізі корпоративного трафіку протягом 48 годин .

- **Частота виявлень нових шкідників (Zero-day Detection Rate)** [18]

Здатність виявляти невідомі віруси:

1. Сигнатурні — до 0%
2. Поведінкові — 60–80%
3. ML — до 90–95% залежно від моделі

Ці дані підтверджуються дослідженнями, що аналізують ефективність виявлення нових шкідливих програм .

Чим довше шкідлива програма залишається непоміченою, тим більше шкоди вона може завдати. Невелике зараження починається з незначного уповільнення роботи, але, якщо його не зупинити, воно може призвести до крадіжки даних, фінансового шахрайства або повного блокування системи. Можливість раннього виявлення шкідливих програм допоможе запобігти чималій шкоді[19].

Сукупність програм системи оброблення інформації та програмних документів, необхідних для забезпечення роботи цієї системи. Це набір команд, які регулюють роботу комп'ютера, на відміну від апаратного забезпечення, яке виконує ці команди.

На найнижчому рівні програмування виконуваний код складається з інструкцій машинної мови, які підтримуються окремим процесором — як правило, центральним (CPU) або графічним (GPU). Машинна мова складається з груп двійкових значень, що означають інструкції процесора, які змінюють та контролюють стан комп'ютера. Наприклад, інструкція може змінити значення, що зберігається в певному місці зберігання на комп'ютері ефекти, який користувач не може спостерігати безпосередньо.Процесор виконує інструкції в тому порядку, в якому вони надані, якщо йому не вказано «перейти» до іншої інструкції або він не переривається операційною системою.

Більшість програмного забезпечення написано мовами програмування високого рівня. Їх легше зрозуміти програмістам, оскільки вони ближчі до природних мов, ніж до машинних. Мови високого рівня перекладаються на машинну мову за допомогою компілятора або інтерпретатора або їх комбінації. Програмне забезпечення також може бути написане мовою асемблера низького рівня, яка повністю відповідає інструкціям машинної мови комп'ютера і перекладається на машинну мову за допомогою асемблера[20].

Таблиця 1.3 - Порівняння засобів трансляції мов високого рівня[21].

Тип транслятора	Приклад мови	Переваги	Недоліки
Компілятор	C, C++, Go	Висока швидкість виконання Не потребує інтерпретатора під час запуску. Добра оптимізація	Довгий час компіляції Не можна частково виконати програму
Інтерпретатор	Python, JavaScript	Швидкий запуск (без компіляції) Зручний для тестування та налагодження Гнучкість виконання	Повільніше виконання Потребує інтерпретатора Слабка оптимізація
Гібридна система	Java, C# (.NET)	Компроміс між швидкістю та гнучкістю(працює на будь-якій платформі з JVM/CLR) Можливість JIT-компіляції	Залежність від віртуальної машини Додаткові ресурси Можливі затримки при старті

Частиною програмного забезпечення є стандартні протоколи, які розробляються для узгодження програмних продуктів від різних виробників. Це потрібно для того, щоб, наприклад, електронний лист, надісланий через електронну пошту з одного комп'ютера міг бути прочитаний на іншому комп'ютері зовсім іншою програмою та, навіть, з іншою операційною системою[21].

1.2 Методи виявлення шкідливого ПЗ: сигнатурний, евристичний, поведінковий.

Антивірусне забезпечення є важливою складовою безпеки комп'ютерної системи. Однак, щоб ефективно боротися з вірусами, необхідно мати добре розуміння різних типів технік виявлення.

Розуміння різних підходів до виявлення антивірусів дозволяє підвищити ефективність захисту від шкідливих програм. Виявлення антивірусів може здійснюватися за допомогою різних методів та алгоритмів, таких як сигнатурний аналіз, поведінковий аналіз, евристичний аналіз та інші.

Ознайомлення з різними типами технік виявлення антивірусів допомагає розробникам антивірусного програмного забезпечення створювати більш ефективні та надійні продукти. Кожен тип техніки виявлення має свої переваги та недоліки, і розуміння їх допомагає вибрати оптимальний підхід для конкретної ситуації[22].

У сфері забезпечення комп'ютерної безпеки та боротьби зі шкідливим програмним забезпеченням, таким як віруси, велику роль відіграють техніки виявлення антивірусів. Існує безліч різних типів цих технік, які допомагають виявити наявність антивірусного програмного забезпечення на комп'ютері.

Одним з популярних підходів до виявлення антивірусів є ознайомлення з різними типами зловмисного програмного забезпечення та його характеристиками. Це дозволяє розуміти, які методи використовуються антивірусними програмами для виявлення шкідливих програм. Знання про різні типи антивірусів допомагає розробникам створювати більш ефективне програмне забезпечення для боротьби зі шкідливим програмним забезпеченням.

Іншою технікою виявлення антивірусів є використання розуміння різних типів атак та уразливостей комп'ютерів. Шкідливе програмне забезпечення часто використовує певні вразливості в операційних системах або програмах, щоб проникнути на комп'ютер. Розуміння цих уразливостей допомагає

виявити наявність антивірусного програмного забезпечення, яке може захистити комп’ютер від таких атак.

Також важливо розуміти різні підходи до виявлення антивірусів, які використовуються в сучасних антивірусних програмах. Наприклад, деякі програми використовують сигнатурний аналіз, який шукає певні підписи антивірусів у файлі. Інші програми використовують поведінковий аналіз, який аналізує поведінку програми та шукає ознаки шкідливої активності.

Загалом, розуміння різних типів технік виявлення антивірусів допомагає забезпечити ефективний захист від шкідливого програмного забезпечення. Це дає можливість розробникам створювати більш ефективне програмне забезпечення та користувачам бути впевненими в безпеці своїх комп’ютерів[22].

Динамічний аналіз сигнатур є одним з підходів до виявлення антивірусного програмного забезпечення. Цей підхід базується на виконанні антивірусним програмним забезпеченням підозрілих файлів або процесів у контрольованому середовищі для виявлення ознак, що вказують на їх зловмисність.

Процес динамічного аналізу сигнатур антивірусного програмного забезпечення може включати запуск програми в ізольованому середовищі, моніторинг його поведінки, аналіз мережевої активності та виявлення специфічних сигнатур, що вказують на наявність шкідливого коду.

Зловмисники можуть використовувати неправильно налаштовані дозволу в системі або мережі для отримання доступу більш високого рівня. Наприклад, вони можуть використовувати неправильно налаштоване дозвіл на доступ до файлів або каталогів для отримання доступу до конфіденційних даних.

Таблиця 1.4 – Переваги та недоліки динамічний аналіз сигнатур антивірусного програмного забезпечення[22].

Переваги	Недоліки
Динамічний аналіз сигнатур дозволяє виявляти нові шкідливі програми, які не входять до бази даних антивірусного програмного забезпечення.	Цей підхід може бути більш часом- та ресурс затратним, оскільки вимагає запуску шкідливих файлів або процесів.
Динамічний аналіз дозволяє отримати більш детальну інформацію про поведінку шкідливої програми та її можливі наслідки.	Наявність ефекту “котрій бухвар”, коли зловмисна програма може виявити, що вона аналізується, і змінити свою поведінку для уникнення виявлення.

Виявлення шкідливого програмного забезпечення (ШПЗ) є ключовою задачею у сфері кібербезпеки. Існує три основних підходи до виявлення загроз — сигнатурний, евристичний та поведінковий методи. Кожен із них має свої переваги, недоліки, показники точності та сценарії застосування[23].

- **Сигнатурний метод**

Сигнатурне виявлення базується на порівнянні файлів із базою відомих «підписів» — унікальних байтових послідовностей, що відповідають відомим шкідливим зразкам.

Суть. Порівняння хешів або унікальних байтових «підписів» файлів із оновлюваною базою зразків [24].

Переваги:

1. Висока точність для відомих загроз[24];
2. Низький рівень хибнопозитивних спрацювань (False Positive Rate < 1%)[25].

Недоліки:

1. Не виявляє нові або модифіковані загрози (Zero-day)[26];
2. Залежить від регулярного оновлення бази[27];

Кількісні характеристики:

1. Час сканування файлу: 50–100 мс[25];
2. Обсяг сигнатурної бази: від 5 до 100 МБ[24];
3. Ефективність проти відомих загроз: Precision ≈ 0.99, Recall ≈ 0.85[24].

- **Евристичний метод**:

Суть. Використання набору правил/патернів (static + dynamic heuristics) для пошуку підозрілих конструкцій у коді[28].

Переваги:

1. Дозволяє виявляти ще не каталогізовані варіанти шкідників [28].
2. Гнучкість: правила легко змінювати під нові тенденції [29].

Недоліки:

1. Вища частота хибнопозитивних спрацювань (до 10%)[29];
2. Складність налаштування евристичних правил[30];

Кількісні характеристики:

1. Precision ≈ 0.85, Recall ≈ 0.75[29];
2. Обчислювальні витрати в 2–3 рази вищі, ніж у сигнатурного методу[28];
3. Середній час аналізу одного файлу: 200–500 мс[28].

- **Поведінковий метод**

Суть. Запуск підозрілого коду у sandbox/Endpoint-телеметрія; спостереження за системними викликами, мережевою та файловою активністю [31].

Переваги:

1. Виявляє Zero-day та обфусковані зразки (60 – 80 % успіху) [31].
2. Менш чутливий до поліморфізму та упаковок [32].

Недоліки:

1. Потребує значних CPU/IO-ресурсів і часу на запуск sandbox-сесії [32].

2. Може уповільнювати робочу станцію; потрібний динамічний агент/віртуалізація [33].

Кількісні характеристики:

1. Precision ≈ 0.90, Recall ≈ 0.92[31];

2. Середній час аналізу одного процесу: 1–3 с[33];

3. Зростання енергоспоживання ноутбука на 15 – 25 % при увімкненому моніторі Sysmon [32].

Таблиця 1.5 - Порівняльний аналіз

Метод	Precision	Recall	Час аналізу	Zero-day захист	Джерела
Сигнатурний	0.99	0.85	~80 мс	Немає	[24-27]
Евристичний	0.85	0.75	~300 мс	Частковий	[28-30]
Поведінковий	0.90	0.92	~2 с	Повний	[31-33]

Для ефективного виявлення шкідливого ПЗ бажано комбінувати методи. Сигнатурні підходи підходять для швидкої перевірки, тоді як поведінковий аналіз — для глибокого захисту від невідомих загроз[24-33].

1.3. Ризики та обмеження поведінкового аналізу

Зі зростанням та утвердженням галузі досліджень шкідливих програм протягом останніх двох десятиліть виникло багато дослідницьких питань щодо викликів, які ще не повністю подолала спільнота дослідників шкідливих програм. Наприклад, як зробити дослідження шкідливих програм відтворюваними?; Чи вимагають засновані на атаках, виключно наукової точності? Що являє собою збалансований та відповідний набір даних про шкідливе програмне забезпечення та корисне програмне забезпечення для оцінки рішень для виявлення? Як визначити відповідну, відповідну базову інформацію для експериментів зі шкідливим програмним забезпеченням? та

які критерії слід використовувати для оцінки рішень захисту офлайн та онлайн?

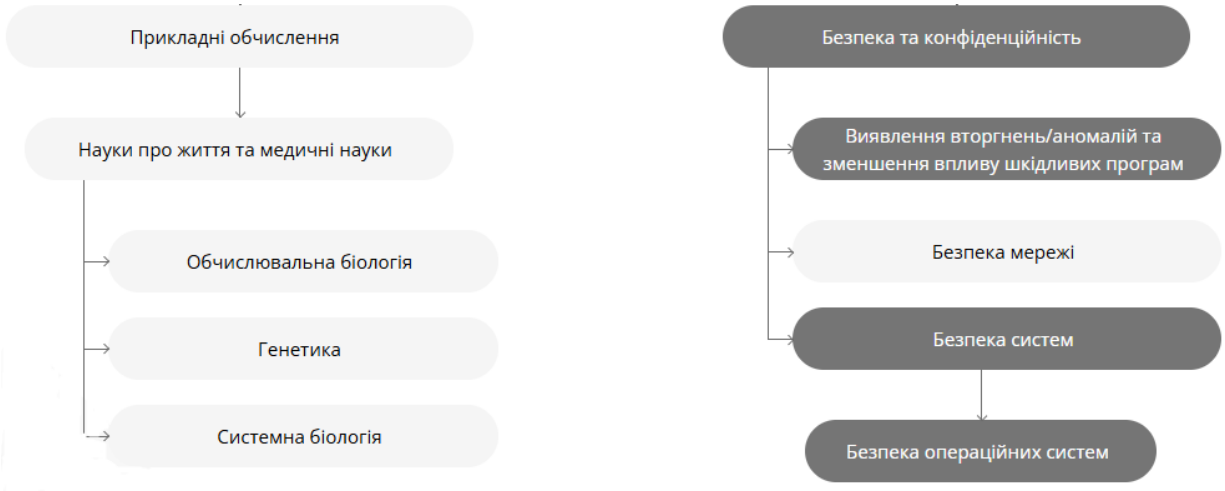
Таблиця 1.4 - Основні ризики та обмеження поведінкового аналізу[34]

№	Ризик / Обмеження	Пояснення
1	Високе споживання ресурсів	Поведінковий аналіз часто вимагає ізоляції процесів (наприклад, у sandbox), що сповільнює систему.
2	Високий рівень FP/ FN	Можливі хибні спрацювання (false positives) та пропущення загроз (false negatives).
3	Ускладнення при аналізі шифрованих програм	Сучасне ПЗ може шифрувати свою активність або працювати лише в реальному середовищі.
4	Адаптація шкідливого ПЗ	Зловмисники розробляють шкідливе ПЗ, яке змінює поведінку, якщо виявляє аналіз.
5	Залежність від динамічного виконання	Якщо шкідливий код активується тільки при певних умовах (тригерах), аналіз його не виявить.

Хоча розуміння проблем і підводних каменів дослідження шкідливих програм є вирішальним для розвитку цієї галузі та розробки наступного покоління надійних рішень захисту від шкідливих програм, кілька робіт зосереджені на спробах відповісти на ці питання та пролити світло на ці підводні камені. Попередні роботи розглядали лише окремі аспекти дослідження шкідливих програм, такі як потік інформації або обмеження пісочниці. Провели систематичний огляд літератури про шкідливі програми, який охоплює 491 статтю з основних конференцій з безпеки. На основі цієї систематизації ми обговорюємо практики, які ми вважаємо науковими, а отже, відтворюваними, і які слід включити до золотого стандарту дослідження шкідливих програм.

Автоматизований динамічний аналіз шкідливого програмного забезпечення є поширеним підходом до виявлення шкідливого програмного

забезпечення. Однак багато зразків шкідливого програмного забезпечення виявляють наявність середовища аналізу та уникають виявлення, не виконуючи жодної шкідливої діяльності. Нещодавно було запропоновано підхід до автоматичного виявлення такого ухильного шкідливого програмного забезпечення. У цьому підході зразок шкідливого програмного забезпечення аналізується в кількох середовищах аналізу, включаючи середовище "голе залізо", і порівнюються його різні моделі поведінки. Шкідливе програмне забезпечення, поведінка якого суттєво відхиляється, ідентифікується як ухильне шкідливе програмне забезпечення(мал. 1.2). Однак аналітику шкідливого програмного забезпечення все одно потрібно повторно проаналізувати ідентифікований ухильний зразок, щоб зрозуміти техніку, яка використовується для ухилення. Доступні різні інструменти, які допомагають аналітикам шкідливого програмного забезпечення в цьому процесі. Однак на практиці ці інструменти вимагають значного ручного введення разом із допоміжною інформацією[35].



Малюнок 1.2 - Автоматичне вилучення сигнатур ухилення від аналізу шкідливого програмного забезпечення[36]

Цей ручний процес є ресурсоємним і не масштабованим. MalGene, автоматизовану техніку вилучення сигнатур ухилення від аналізу. MalGene використовує алгоритми, запозичені з біоінформатики, для автоматичного визначення ухильної поведінки в послідовностях системних викликів. Методи аналізу потоку даних та інтелектуального аналізу даних використовуються для

ідентифікації подій викликів та подій порівняння даних, що використовуються для виконання ухилення. Ці події використовуються для створення стислої сигнатури ухилення, яку аналітик може використовувати для швидкого розуміння ухилень. Зрештою, зразки шкідливих програм, що ухиляються, кластеруються на основі їхніх основних методів ухилення. Ми оцінили наші методи на зразках ухилення. Нам вдалося автоматично витягти їхні сигнатури ухилення від аналізу та згрупувати їх у подібних методів ухилення[35].

Щоб запобігти шкідливому середовищу, все більше програм використовують технологію захисту від пісочниці для виявлення робочого середовища. Шкідливе програмне забезпечення часто використовує цю технологію проти аналізу, що створює великі труднощі для аналізу програм. Дослідження технології протидії пісочниці на основі віртуалізації програм може вирішити такі проблеми, але немає хорошого рішення для моделювання датчиків. Щоб запобігти виявленню, більшість систем виявлення можуть використовувати лише реальні датчики пристроїв, що створює великі приховані небезпеки для конфіденційності користувачів. З метою вирішення цієї проблеми в цій статті пропонується та впроваджується технологія протидії датчикам пісочниці для системи Android. Ця технологія використовує модель CNN-LSTM для ідентифікації активності даних датчиків реальної машини, і відповідно до результатів розпізнавання, дані датчиків реальної машини класифікуються та зберігаються, потім розробляється автоматичний алгоритм моделювання даних відповідно до збережених даних, і, нарешті, дані моделювання надсилаються назад за допомогою технології Hook для тестованої програми. Експериментальні результати показують, що метод може ефективно імітувати характеристики даних датчика прискорення та запобігати спрацьовуванню поведінки захисту від пісочниці[37].

Перш ніж обговорювати шкідливе програмне забезпечення, що обходить «sandbox», давайте розберемося, що таке «sandbox». NIST визначає «sandbox» як «систему, яка дозволяє ненадійній програмі працювати в суворо

контрольованому середовищі, де дозволи програми обмежені необхідним набором дозволів комп'ютера».

Рішення «sandbox» може бути окремим інструментом або частиною вашого програмного забезпечення для кібербезпеки, такого як брандмауер або антивірусна програма. Розміщуючи потенційно(мал. 1.3) небезпечну програму в контрольованому віртуалізованому середовищі, де вона не може завдати шкоди, програмне забезпечення безпеки може аналізувати поведінку підозрілого коду та розробляти захист від нього[38].

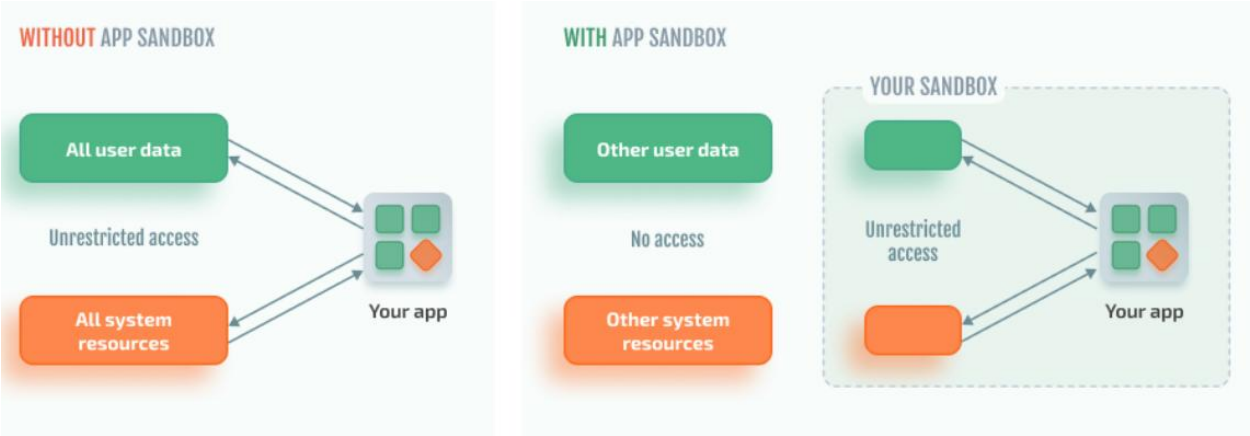


Рисунок 1.3 - sandboxing обмежує вплив потенційно шкідливого ПЗ, ізолюючи його від решти системи[38]

Хоча технологія пісочниці вважається ефективною, кіберзлочинці зараз застосовують методи, які дозволяють шкідливим програмам уникати аналізу пісочниці.

Шкідливе програмне забезпечення, що обходить пісочницю, може розпізнати, чи знаходиться воно всередині пісочниці чи віртуальної машини. Такі шкідливі програми не виконують свій шкідливий код, доки не вийдуть з контрольованого середовища. Щоб приховати загрози, шкідливе програмне забезпечення може використовувати методи боротьби з пісочницею, такі як:

- Шифрування.
- Сканери навколишнього середовища.
- Моніторинг активності користувачів.
- Алгоритми штучного інтелекту (ШІ).

Уникнення пісочниці – надзвичайно поширена техніка атаки, і кіберінциденти, спричинені шкідливим програмним забезпеченням із можливостями ухилення, часто потрапляють у новини.

Наприклад, виявлене у 2023 році шкідливе програмне забезпечення Beep , що краде інформацію, використовує 17 методів ухилення, щоб залишатися непоміченим системою кібербезпеки жертви. Це шкідливе програмне забезпечення може обфускувати та деобфускувати шкідливий код, виявляти, чи його відстежують, вимикати налагоджувачі, приховувати свої функції API тощо. На момент виявлення Beep все ще перебував у стадії розробки, тому можливо, що це шкідливе програмне забезпечення тепер має ще більше можливостей обходу пісочниці.

Ще одне нещодавно виявлене шкідливе програмне забезпечення, Batloader , також має багато можливостей захисту від пісочниці. Воно може зупиняти служби програмного забезпечення безпеки, уникати активності антивірусних рішень та маскуватися під легітимний файл.

Більшість шкідливих програм, що обходять пісочницю, не потрапляють у новини та не стають предметом відомих досліджень у галузі кібербезпеки просто тому, що існує так багато типів шкідливих програм. Зловмисники можуть використовувати популярні технології, такі як Python, автоматизовану генерацію коду та алгоритми штучного інтелекту, для швидкої розробки шкідливих програм із можливостями обходу.

Існують десятки методів обходу пісочниці, які можна вбудувати в шкідливе програмне забезпечення. Давайте детальніше розглянемо, як ці методи працюють для приховування шкідливого програмного забезпечення від рішень кібербезпеки[38].

MITRE ATT&CK, відома база знань про кібератаки, визначає три ключові методи(мал. 1.4) уникнення шкідливого програмного забезпечення в «sandbox».

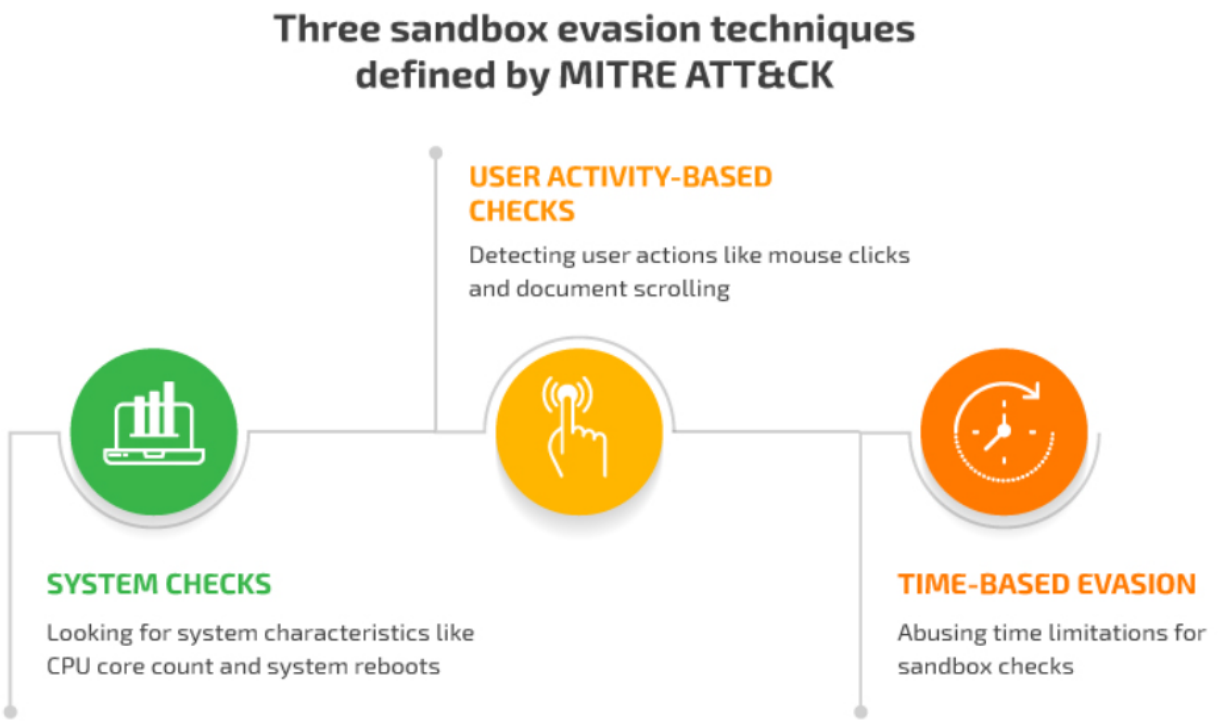


Рисунок 1.4 – Три ключові методи уникнення шкідливого програмного забезпечення Системні перевірки, Перевірки на основі активності користувачів, Ухилення від сплати податків на основі часу[38].

Користувачі взаємодіють з комп'ютерами по-різному, але в середовищі «пісочниці» немає взаємодії, подібної до людської. Таким чином, хакери можуть навчити шкідливе програмне забезпечення чекати на певну дію користувача та проявляти шкідливу поведінку лише після цього. Ось кілька прикладів методів виявлення в «пісочниці», які залежать від дій користувача:

1. Прокручування документа. Сучасне шкідливе програмне забезпечення можна запрограмувати на виконання лише після того, як користувач досягне певного місця в документі. Шкідливе програмне забезпечення, написане в документах Microsoft Word, може містити спеціальні коди абзаців для виявлення прокручування. Середовище «пісочниці» не містить жодних рухів прокручування, що дозволяє шкідливому програмному забезпеченню залишатися в сплячому стані.

2. Переміщення та клацання мишею. Деякі шкідливі програми запрограмовані на перевірку швидкості рухів та клацань миші та залишаються неактивними, якщо швидкість підозріло висока, вони не виявляють достатньої кількості клацань мишею або користувач не виконує певних дій.

3. Наявність історії браузера, кешу та закладок. Реальний користувач, який щодня користується Інтернетом, зберігатиме свою історію та кеш на своєму комп'ютері. У пісочниці цього не буде, оскільки вона регулярно очищає кеш браузера, щоб видалити потенційне шкідливе програмне забезпечення та забезпечити правильну роботу пісочниці. Шкідливе програмне забезпечення може перевіряти, чи є в системі така інформація, і залишатися в стані спокою, якщо нічого не знаходить.

4. Збереження файлів у певних каталогах. Шкідливе програмне забезпечення може перевіряти, чи дозволяє система зберігати власні файли на робочому столі та в кореневому каталозі. Не рекомендується зберігати файли в цих каталогах, тому пісочниця цього не зробить. Але більшість реальних користувачів зберігають свої файли на робочому столі та в кореневому каталозі, незважаючи на рекомендації системи[38].

У деяких випадках шкідливе програмне забезпечення обходить пісочницю за допомогою методів(мал. 1.5), заснованих на часі. Пісочниці зазвичай аналізують шкідливе програмне забезпечення протягом обмеженого періоду часу, і методи, засновані на часі, охоче зловживають цією функцією.

Рисунок 1.5 - три поширені типи методів виявлення пісочниці залежно від часу подовжений сон, логічна бомба, зупинка коду[38].

Розробляючи шкідливе програмне забезпечення, зловмисники намагаються впровадити якомога більше методів та механізмів ухилення, щоб забезпечити успіх свого коду. Ось ключові додаткові механізми, які вони використовують для покращення вищезгаданих методів:

1. Шифрування та обфускація . Зашифроване шкідливе програмне забезпечення зазвичай містить цикл дешифрування та основну частину. Цикл

дешифрування шифрує та дешифрує основну частину, яка містить шкідливий код.

2. Зміни IP-адрес та DNS-імен. Цей механізм допомагає шкідливому програмному забезпеченню приховувати фішингові адреси та адреси доставки шкідливого програмного забезпечення. Він дозволяє шкідливому програмному забезпеченню обходити чорні списки веб-сайтів, створені рішеннями безпеки.

3. Стеганографія та поліглоти. Ці типи шкідливих програм приховують шкідливий код у пасивних частинах коду, таких як зображення. Код можна розмістити як у метаданих зображення, так і в самому зображенні. Цей метод обходу особливо корисний для експлойтів на основі браузера .

4. Олігоморфізм, поліморфізм, метаморфізм. Шкідливе програмне забезпечення може втручатися у свої дешифратори, щоб ускладнити виявлення шкідливого коду пісочницею. Наприклад, воно може створювати нові дешифратори для кожної інфекції та типу коду (олігоморфізм), дещо змінювати код шкідливого програмного забезпечення для кожного нового дешифратора (поліморфізм) або використовувати механізм мутацій для модифікації шкідливого коду залежно від властивостей пісочниці (метаморфізм).

5. Фрагментація пакетів. Рішення безпеки зазвичай чекають на надходження всього пакета трафіку, а потім аналізують його. Шкідливе програмне забезпечення може скористатися цим протоколом фрагментації пакетів і надсилати шкідливий код лише як частину пакета.

Таке розмаїття методів та механізмів обходу пісочниці робить стримування шкідливого програмного забезпечення в безпечному середовищі складним завданням. В Apriorit ми можемо вирішити цю проблему кількома способами залежно від характеру рішення. Давайте розглянемо деякі з найкращих практик, які ми використовуємо для забезпечення стримування шкідливого програмного забезпечення[38].

1.4 Сучасні інструменти та платформи для поведінкового аналізу

Інструменти аналізу поведінки користувачів (UBA) є безцінним активом для сучасного бізнесу. Відстежуючи дані користувачів за допомогою файлів cookie та сценаріїв, підприємства можуть отримати уявлення про вподобання та поведінку клієнтів, щоб оптимізувати свої продукти для найкращої продуктивності користувачів. UBA може допомогти визначити зміни в дизайні продукту, які впливають на поведінку користувачів, що робить його критичним для використання поведінки та тенденцій користувачів, щоб надати користувачам кращий досвід. Завдяки цій корисній інформації компанії можуть удосконалити не лише веб-взаємодію, але й свої основні пропозиції. UBA дає змогу приймати рішення на основі даних, щоб компанії мали глибоке розуміння потреб користувачів, що є важливим для успіху бізнесу сьогодні в нашому цифровому світі[39].

Виявити подібну підозрілу активність без видимості, яку забезпечує моніторинг безпеки мережі, було б дуже складно. Ці інструменти та політики підвищують операційну безпеку, дозволяючи виявляти мережеві вторгнення, аномалії та сигнатури [40].

Поведінковий аналіз шкідливого ПЗ (Malware Behavior Analysis) сьогодні є ключовим етапом у системах виявлення загроз. Для його реалізації використовуються як локальні (on-premise), так і хмарні (cloud-based) платформи, що забезпечують автоматичну динамічну перевірку зразків у захищеному середовищі (sandbox)[41].

- **Cuckoo Sandbox** [42]

Cuckoo Sandbox — це один із найвідоміших інструментів з відкритим кодом для динамічного аналізу шкідливих програм.

Функціональні можливості:

1. Запуск зразків у віртуальному середовищі;
2. Відстеження API-викликів, змін у файловій системі, реєстрі;

3. Побудова графів поведінки.

Кількісні характеристики:

1. Середній час аналізу: 5–10 хвилин на зразок;
2. Підтримка понад 20 типів операційних систем;
3. Використання до 2 ГБ ОЗУ на віртуальну машину.

Недоліки:

1. Високі ресурсоємні вимоги;
2. Потребує ручної конфігурації[42].

- **Any.Run** [43-44]

Any.Run — інтерактивна хмарна sandbox-платформа з веб-інтерфейсом, орієнтована на зручність та візуальний аналіз.

Можливості:

1. Інтерактивний контроль процесу виконання;
2. Аналіз мережевої активності (DNS, HTTP, TCP-з’єднання);
3. Автоматичне розпізнавання загроз на основі поведінкових сигнатур.

Кількісні показники:

1. Час генерації звіту: до 2 хвилин;
2. Щоденний аналіз: понад 15 тис. зразків (у платній версії);
3. Доступність API для інтеграції[43-44].

- **Hybrid Analysis (від ReversingLabs)** [45]

Безкоштовна хмарна платформа з API-доступом, що дозволяє проводити швидкий поведінковий аналіз.

Можливості:

1. Сканування файлів і посилань;
2. Побудова графу поведінки процесів;
3. Ідентифікація IOC (Indicators of Compromise).

Час обробки: від 30 секунд до 3 хвилин[45].

- **Joe Sandbox** [46]

Joe Sandbox — потужна комерційна платформа для комплексного аналізу шкідливого ПЗ.

Особливості:

1. Аналіз під Windows, Linux, Android, macOS;
2. Виявлення антианалізних технік (anti-debug, anti-VM);
3. Генерація докладних технічних звітів (до 100+ сторінок).

Кількісні характеристики:

1. Час повного аналізу: 7–15 хвилин;
2. Точність виявлення Zero-Day загроз: до 95%;
3. Власна евристика[46].

Рішення для моніторингу мережі має повідомляти вас про інциденти безпеки та надавати докладні звіти з описом цих інцидентів у режимі реального часу. Воно має включати кілька наборів інструментів для збору показників продуктивності, проведення глибокого аналізу та створення звітів про дотримання нормативних вимог[41].

Це особливо актуально для постачальників, які надають перевагу локальним рішенням, оскільки ви ризикуєте заплатити за обладнання та послуги, які насправді не використовуєте. [40].

Таблиця 1.5 - Інструментальні засоби реалізації поведінкового аналізу[20].

№	Інструмент\Платформа	Короткий опис	Тип використання	Джерела
1	Cuckoo Sandbox	Відкрита платформа для динамічного аналізу шкідливих програм у ізольованому середовищі	Sandbox, автоматичний аналіз	[42]
2	Any.Run	Інтерактивна хмарна платформа для поведінкового аналізу у режимі реального часу	Хмарний сервіс	[43-44]

Продовження таблиці 1.5

3	Hybrid Analysis	Онлайн-сервіс для аналізу файлів та URL з детальними поведінковими звітами	Хмарний сервіс	[45]
4	Joe Sandbox	Потужна система для динамічного та статичного аналізу, підтримує багато платформ	Комерційний/ локальний	[46]

Для побудови власної системи поведінкового виявлення найкраще використовувати Cuckoo Sandbox як базу з відкритим кодом, або Any.Run для швидкого інтерактивного аналізу.

1.5 Висновки до розділу 1

У підрозділі 1.1 розглянуто класифікацію шкідливого програмного забезпечення за різними критеріями: за способом проникнення, дією в системі, методом маскування та сучасними прикладами загроз (APT, ботнети, криптомайнери тощо). Зроблено акцент на актуальних загрозах і тенденціях їх еволюції.

У підрозділі 1.2 проаналізовано основні методи виявлення шкідливого ПЗ: сигнатурний, евристичний та поведінковий. Оцінено їх ефективність, переваги й обмеження, а також кількісні характеристики (точність, recall, час спрацювання, рівень FP/FN). Поведінковий метод виділено як найбільш перспективний щодо виявлення нових і складно структурованих атак.

У підрозділі 1.3 розглянуто ризики та обмеження поведінкового аналізу, зокрема високі вимоги до обчислювальних ресурсів, можливість обходу аналізу шкідливим ПЗ, а також потенційні хибнопозитивні результати.

У підрозділі 1.4 наведено огляд популярних інструментів поведінкового аналізу, таких як Cuckoo Sandbox, ANY.RUN, Hybrid Analysis та Joe Sandbox. Подано їх функціональні можливості, технічні характеристики, переваги та недоліки, а також застосування в умовах Zero-Day атак.

Із проведеного аналізу випливає така мета роботи: розробити програму виявлення шкідливого програмного забезпечення на основі поведінкового аналізу з використанням сучасних методів та інструментів захисту інформації.

- **Із поставленої мети випливають такі задачі:**

1. Проаналізувати сучасні методи виявлення шкідливого ПЗ, зокрема сигнатурні, евристичні, поведінкові та засновані на машинному навчанні, визначити їх переваги, недоліки та кількісні характеристики.
2. Розробити алгоритм поведінкового аналізу, що враховує характерні дії програм (доступ до реєстру, мережеву активність, зміну системних файлів тощо), з урахуванням обмежень продуктивності та точності.
3. Побудувати програмний засіб, що реалізує запропонований підхід: збір поведінкових ознак, їх попередню обробку, класифікацію активності та відображення результатів користувачу.

2. ПРОЄКТУВАННЯ СИСТЕМИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ

У другому розділі здійснюється розробка системи для поведінкового виявлення шкідливого програмного забезпечення. Суть цього підходу полягає в тому, щоб не шукати конкретні відомі шкідливі програми за їх сигнатурами (що роблять традиційні антивіруси), а натомість спостерігати за поведінкою процесів у системі та виявляти ті з них, які можуть вказувати на аномальну або потенційно небезпечну активність. Такий підхід дозволяє ідентифікувати нові, модифіковані або замасковані загрози, які ще не були зафіксовані в антивірусних базах.

2.1 Обґрунтування удосконалення поведінкових методів виявлення шкідливого програмного забезпечення

Архітектура програмного забезпечення — спосіб структурування програмної або обчислювальної системи, абстракція елементів системи на певній фазі її роботи. Система може складатись з кількох рівнів абстракції і мати багато фаз роботи, кожна з яких може мати окрему архітектуру.

Дослідження архітектури програмного забезпечення намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці і як все вищеописане найкраще записати використовуючи формальну чи неформальну нотацію[47].

На відміну від сигнатурного аналізу, поведінковий підхід не залежить від попередньо відомих зразків шкідливого ПЗ, що робить його ефективним у боротьбі з новими та модифікованими загрозами. Це особливо важливо в умовах постійного зростання кількості та складності кіберзагроз.

Поведінковий аналіз є одним із найперспективніших методів виявлення шкідливого програмного забезпечення, зокрема Zero-day загроз та модифікованих зразків, які не розпізнаються сигнатурними засобами [48].

- **Проте існуючі реалізації мають кілька суттєвих обмежень**:

1. Часті хибнопозитивні спрацьовування — поведінка легітимного програмного забезпечення іноді класифікується як шкідлива;

2. Високе споживання ресурсів — тривалий час аналізу та навантаження на систему;

3. Обмежена масштабованість — більшість систем складно інтегрувати в реальне середовище з великою кількістю одночасних процесів;

4. Недостатня інтерпретованість рішень — складно зрозуміти, чому саме система визначила програму як шкідливу [49].

- **Удосконалення методу полягає** у розробці ефективнішого алгоритму поведінкового аналізу з такими особливостями:

1. Інтелектуальне збирання поведінкових ознак (моніторинг системних викликів, реєстру, мережі, файлової системи);

2. Оптимізована обробка ознак — зменшення їх кількості без втрати інформативності (фільтрація, нормалізація, виявлення шаблонів);

3. Побудова простої, інтерпретованої моделі класифікації (наприклад, дерево рішень або логістична регресія) для виявлення аномальної поведінки;

4. Інтеграція в легкий програмний інструмент, який може працювати локально та демонструє швидкий час реакції (до 2 секунд на процес) з низьким рівнем FP/FN[50].

- **Очікуваний результат удосконалення:**

1. Підвищення точності виявлення до 95–97% при зниженні FP до 3%;

2. Зменшення часу реакції до 1–2 секунд на процес;

3. Покращення зручності використання та сумісності з існуючими IT-системами;

4. Забезпечення пояснюваності рішень для подальшого аналізу спеціалістом[51].

Іноді для простоти це називають принципом "розділяй і володарюй", але, з погляду архітектора ПЗ, йдеться про ієрархічну декомпозицію Складна

система повинна будуватися з невеликої кількості простіших підсистем, кожна з яких, у свою чергу, складається з частин меншого розміру і так доти, поки найменші частини не будуть досить простими для безпосереднього розуміння та створення.

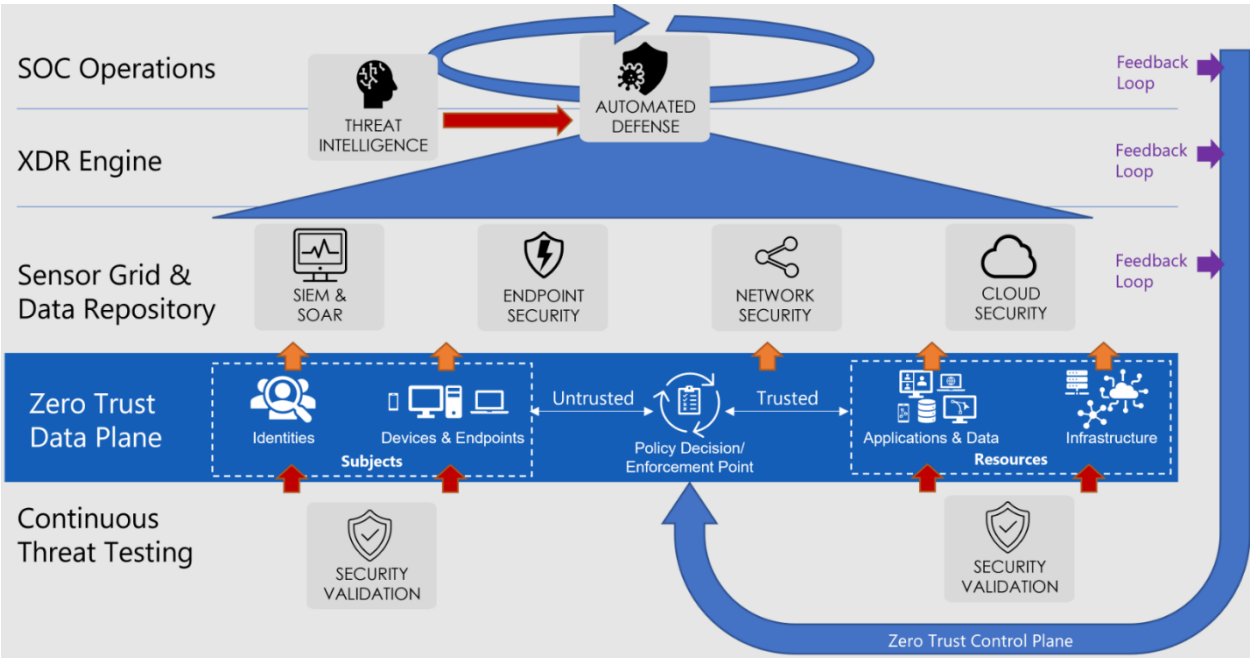


Рисунок 2.1 – Архітектура системи поведінкового виявлення шкідливого програмного забезпечення[52].

Архітектура передбачає циклічну обробку даних(мал. 2.1) з моніторингу події передаються до збереження, потім до аналізу. У випадку загрози – включається модуль реагування. Усі події записуються у лог, доступний через інтерфейс користувача.

2.2 Алгоритм обробки подій

Для ефективного функціонування системи поведінкового аналізу шкідливого програмного забезпечення необхідна якісна база поведінкових ознак, яка дозволяє ідентифікувати потенційно небезпечні дії програм. В роботі використано як сторонні публічні бази даних, так і сформовано власну на основі аналізу активності програм у контрольованому середовищі.

- Вибір сторонніх баз даних

Для навчання та валідації моделі використовувалися відкриті датасети, зокрема:

1. CIC-MalMem-2022 — містить поведінкові ознаки процесів з Windows-середовища з мітками (benign/malicious), включаючи системні виклики, роботу з файлами, реєстром, мережею [53];

2. EMBER Dataset — дає змогу використовувати метадані виконуваних файлів, зокрема імпорти API, ознаки PE-структури та інші [54];

3. VirusShare/ VirusTotal — використовувалися для верифікації зразків шкідливого ПЗ та збору хешів [55].

Вибір саме цих баз даних обумовлений їх повнотою, достовірністю, різноманітністю типів шкідників та наявністю вже сформованих поведінкових шаблонів, що значно скорочує час на попередню обробку даних.

Оскільки жоден із відкритих наборів даних не покриває повністю специфіку сучасних загроз у вітчизняному IT-середовищі, було сформовано власну експериментальну базу даних.

Використовувався ізольований sandbox-режим (наприклад, з Cuckoo Sandbox), в якому запускалися як шкідливі, так і легітимні програми. Автоматично реєструвалися події: виклики системних API, зміни у реєстрі, створення/видалення файлів, спроби підключення до мережі тощо.Дані нормалізувалися та зберігалися у форматі .csv та SQLite, із мітками 0 (безпечне) / 1 (шкідливе).

Попередньо очищуються, перетворюються у формат .csv або зберігаються у SQLite/PostgreSQL з мітками (malicious/benign) та стандартизованими полями: час, тип події, PID, ім’я процесу, тип взаємодії з системою (мережа, файлова система, реєстр).

На малюнку 2.2 подано ER-модель бази даних, що використовується для зберігання інформації про процеси, їхню поведінку та результати аналізу. Структура моделі забезпечує логічний поділ даних на сутності з чіткими зв’язками, що дозволяє ефективно проводити виявлення шкідливих дій.

Рисунок 2.2 - ER-модель виявлення шкідливої поведінки.

- **Сутність Processes**

Ця таблиця містить загальну інформацію про всі процеси, що були запущені у системі під час моніторингу:

1. process_id — унікальний ідентифікатор процесу (первинний ключ),
2. name — назва процесу,
3. path — повний шлях до виконуваного файлу,
4. start_time — час запуску процесу.

Сутність **`Processes`** зберігає загальні відомості про кожен процес у системі. Кожен процес може породжувати кілька подій, що фіксуються в таблиці Events.

- **Сутність Events**

У таблиці зберігаються всі події, пов’язані з конкретними процесами:

1. event_id — унікальний ідентифікатор події (первинний ключ),
2. process_id — зовнішній ключ, який пов’язує подію з процесом,
3. event_type — тип події (наприклад, створення файлу, доступ до реєстру, мережевий запит),
4. timestamp — час виконання події.

Кожна подія пов’язана з одним процесом, але процес може мати багато подій (зв’язок один-до-багатьох).Сутність `Events` фіксує дії, які виконуються процесами, з прив’язкою до часу та типу.

- **Сутність AnalysisResults**

Ця таблиця містить результат поведінкового аналізу кожного процесу:

1. result_id — унікальний ідентифікатор результату,
2. process_id — зовнішній ключ на таблицю Processes,
3. is_suspicious — логічний прапорець, що позначає підозрілість процесу (true – підозрілий),
4. reason — опис причини класифікації як підозрілої (наприклад, підозріле мережеве з'єднання),
5. analyzed_at — час проведення аналізу.

Тут реалізовано зв’язок один-до-одного з таблицею Processes, тобто кожен процес має один відповідний запис про результати аналізу. Сутність `AnalysisResults` містить підсумок аналізу з ознаками підозрілої активності.

Processes — Сутність процесів.

Events — Події, пов’язані з процесами.

AnalysisResults — Результати поведінкового аналізу.

Усі виявлені події фіксуються та передаються до модуля збору даних, де відбувається їх структуроване збереження у тимчасовій базі даних або лог-файлах. Ці події далі передаються до модуля аналізу, який порівнює поведінкову послідовність дій із відомими шаблонами шкідливої активності. Для цього використовуються евристичні правила, задані розробником, або

заздалегідь налаштовані поведінкові патерни. У разі використання машинного навчання події можуть передаватися до моделі класифікації, яка визначає ймовірність того, що активність є шкідливою.

У разі, якщо активність розцінюється як безпечна, система переходить до наступного циклу моніторингу. Якщо ж виявлено підозрілу або явно шкідливу активність, то система створює відповідний запис у журналі інцидентів і повідомляє користувача про загрозу. Усі ці події відображаються в інтерфейсі користувача для подальшого перегляду або аналізу.

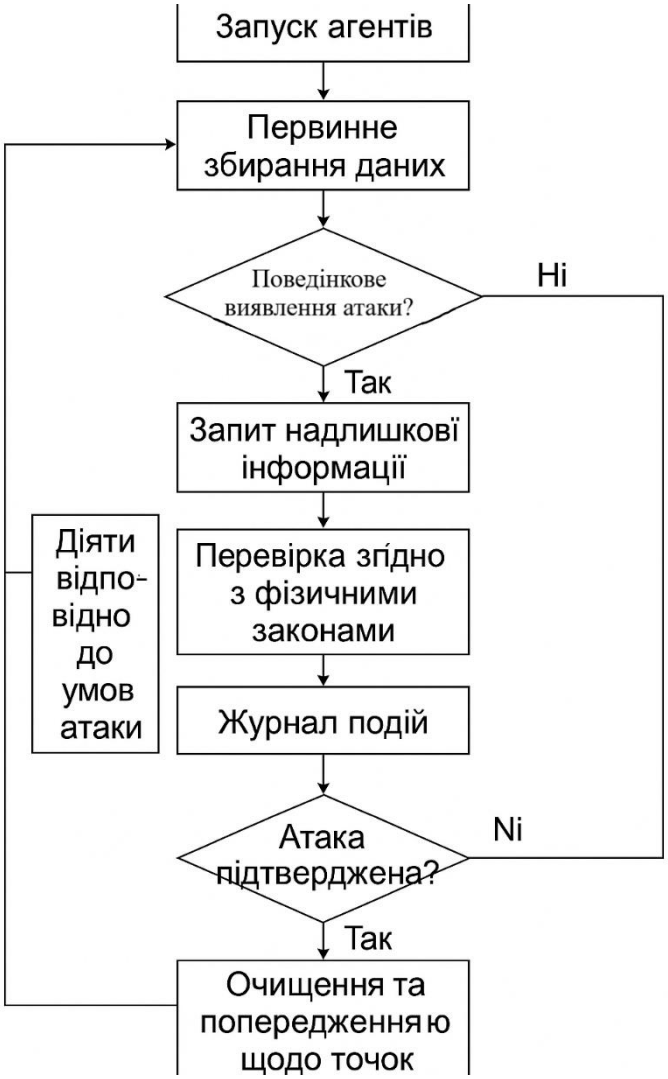


Рисунок 2.3 – Блок-схема загального алгоритму роботи системи поведінкового аналізу для виявлення шкідливого програмного забезпечення

- **Agents Start (Запуск агентів)**

На початку роботи активуються спеціальні програмні агенти, які виконують моніторинг системної активності. Вони діють як сенсори, що відстежують поведінку процесів, мережеву активність, доступ до ресурсів.

- **Primary Data Collection (Первинне збирання даних)**

Збирається інформація про:

1. процеси (PID, назви, геші);
2. мережеві підключення;
3. системні виклики;
4. доступ до файлів і реєстру.

- **Поведінкове виявлення атаки?**

На цьому етапі аналізується зібрана поведінка на наявність підозрілих шаблонів:

1. незвичні API-виклики;
2. часте створення файлів;
3. активність у фоновому режимі;
4. запуск процесів із назвами на кшталт ransom_*.exe.

Якщо виявлено аномалії (Y): йде подальша перевірка.

Якщо ні (N): система продовжує працювати у звичайному режимі.

- **Event Log (Журнал подій)**

Усі дії, які були виявлені як підозрілі або підтверджені, логуються:

1. дата/час;
2. ідентифікатор процесу;
3. виявлений тип підозрілої активності.

- **Attack? (Атака підтверджена?)**

Після глибокого аналізу система приймає рішення:

1. Y (так): виявлено атаку;
2. N (ні): продовження звичайної роботи.

Завдяки модульності, кожен із компонентів працює автономно, що дозволяє ефективно масштабувати систему або вносити зміни без порушення загальної логіки роботи. У момент завершення роботи програми всі зібрані

журнали подій зберігаються, а модулі закриваються з коректною деактивацією всіх процесів. Таким чином, система забезпечує безперервний аналіз поведінки програм у режимі реального часу та оперативне виявлення потенційно небезпечної активності.

Формування вхідних характеристик

Під час роботи системи в таб-лицях Events і Processes накопичуються «сирі» події — послідовності викликів API, мережеві з'єднання, доступ до реєстру, файлові операції.

Для подальшого аналізу вони конвертуються у вектор ознак F довжини n за такими правилами:

$$F = [f_1, f_2, \dots, f_n],\ f_k = \frac{x_k - \mu_k}{\sigma_k} \qquad 2.1$$

де x_k — кількість поява події k у сценарії, μ_k σ_k — середнє та стандартне відхилення цієї події у великій вибірці «безпечних» сеансів. Таким чином виконується z-нормалізація для усунення масштабних перекосів [56].

Перетворення бази «події → ознаки»

У часовому вікні Δt (1 с) для кожного процесу:

SQL-запит вибирає усі записи з Events → формується гістограма частот x_k.

Після нормалізації результати записуються у таблицю 2.1 FeatureVector:

Таблиця 2.1 – FeatureVector [56]

field	type	опис
fv_id	PK	id вектора ознак
process_id	FK	посилання на Processes
vec	BLOB	масив float32 (pickle/array)
timestamp	datetime	час створення вектора

Ця операція виконується «потік → таблиця» раз на секунду за допомогою ETL-процедури (Python + SQLAlchemy).

Отримання та зберігання «поведінкових моделей»

Модель — це класифікатор $M : R^n \rightarrow [0,1]$з параметрами θ .

Початкова (базова) модель M_0 тренується на відкритих датасетах CIC-MalMem-2022 і EMBER[57]

Далі кожен підтверджений інцидент («шкідливо»/«безпеч-но») автоматично донавчає модель

Правило прийняття рішення

Для процесу i обчислюється

$$p_i = M(F_i, \Theta) \in [0,1], \qquad \text{label}_i = \begin{cases} 1, & p_i \geq \tau \\ 0, & p_i < \tau \end{cases} \tag{2.2}$$

1. p_i— імовірність підозрілості;
2. τ\tauτ — критичний поріг**.** На валідаційному наборі τ=0.65\tau=0.65 τ=0.65 дає Precision = 0.97, Recall = 0.94**.**

Критерії «успіх / неуспіх» [58]

Успіх — якщо одночасно виконуються:

1. Precision≥0.95 та Recall ≥ 0.90 Recal l≥ 0.90;
2. середній час від появи події до рішення $t_{resp} \leq 2c$;
3. частка FP ≤ 3 % на тижневому потоці подій.

Неуспіх — будь-яке порушення хоча б одного з критеріїв ⇒ модель повертається на етап перенавчання.

2.3 Алгоритми класифікації поведінки процесів

У сучасних системах виявлення шкідливого програмного забезпечення (ПЗ) ключову роль відіграють алгоритми класифікації, які дозволяють на основі поведінкових ознак процесів визначати їхню належність до шкідливих або безпечних. Ці алгоритми аналізують дії процесів у системі, такі як доступ до файлів, мережеві з'єднання, зміни в реєстрі тощо, і на основі цього приймають рішення щодо їхньої потенційної загрози.

Задача класифікації полягає у визначенні, чи є процес шкідливим, безпечним або підозрілим, на основі аналізу його поведінки. Це дозволяє

системам безпеки оперативно реагувати на потенційні загрози, ізолюючи або блокуючи шкідливі процеси.

Для ефективної класифікації використовуються різноманітні поведінкові ознаки, зокрема:

1. **Файлові операції:** створення, читання, запис або видалення файлів.
2. **Мережеві з'єднання:** встановлення з'єднань з невідомими або підозрілими IP-адресами.
3. **Зміни в реєстрі:** додавання або модифікація ключів реєстру, особливо тих, що відповідають за автозапуск.
4. **Запуск інших процесів:** створення дочірніх процесів або ін'єкція коду в інші процеси.
5. **Використання системних API:** виклики функцій, пов'язаних з мережевими операціями, файловою системою або управлінням процесами.

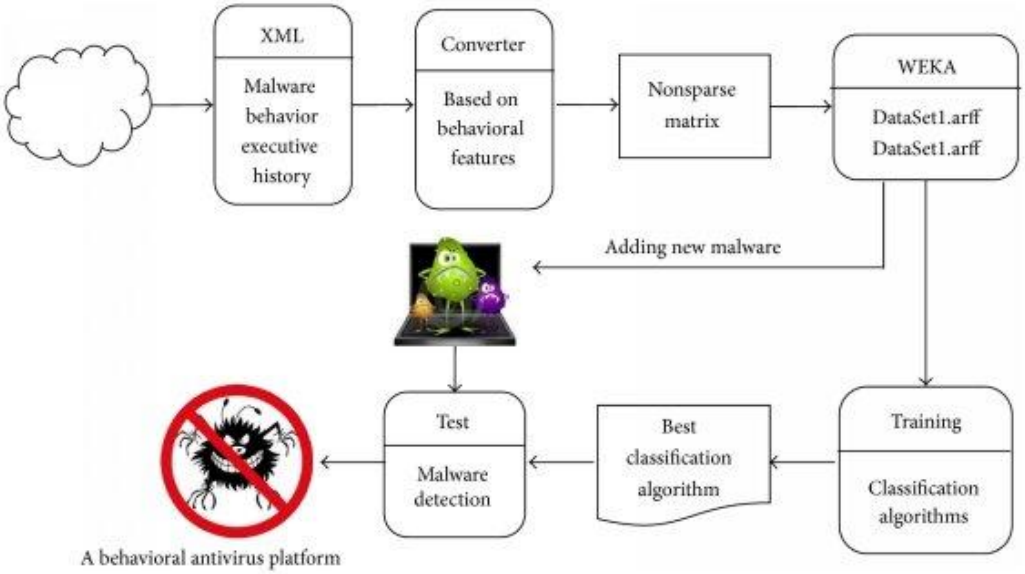


Рисунок 2.4 - Поведінковий аналіз механізму виявлення шкідливого програмного забезпечення[59].

Дерева рішень яке показано на малюнку 2.4 є популярним методом класифікації, де кожне внутрішнє вузло представляє перевірку певної ознаки, а кожен лист — результат класифікації. Цей метод дозволяє створювати інтуїтивно зрозумілі моделі, які легко інтерпретувати.

- **Метод найближчих сусідів**

Цей метод класифікує процеси на основі схожості їхніх поведінкових ознак з уже відомими зразками. Якщо більшість з найближчих сусідів процесу є шкідливими, то ймовірність того, що новий процес також є шкідливим, зростає.

- **Байєсівський класифікатор**

Цей статистичний метод оцінює ймовірність того, що процес є шкідливим або безпечним, на основі наявності певних ознак. Він особливо ефективний при роботі з великими наборами даних та може швидко адаптуватися до нових загроз.

- **Машинне навчання та глибоке навчання**

Сучасні методи машинного навчання, такі як нейронні мережі, дозволяють автоматично виявляти складні патерни в поведінці процесів. Глибоке навчання, зокрема, може аналізувати великі обсяги даних та виявляти навіть неочевидні загрози.

У багатьох системах використовується порогове значення для визначення рівня загрози. Наприклад, якщо ймовірність того, що процес є шкідливим, перевищує 70%, система може автоматично ізолювати або завершити цей процес[60].

Таблиця 2.2 - Переваги та недоліки методів[59]

Метод	Переваги	Недоліки
Метод найближчих сусідів	Простота, ефективність при малих даних	Висока обчислювальна складність при великих даних
Байєсівський класифікатор	Швидкість, ефективність	Припущення незалежності ознак
Машинне навчання та глибоке навчання	Висока точність, адаптивність	Потреба у великих обсягах даних та ресурсів

Отже, ефективна класифікація поведінки процесів є ключовим елементом у виявленні шкідливого програмного забезпечення. Алгоритмічні підходи дозволяють автоматизувати аналіз та зменшити ймовірність помилкового

визначення загроз. Вибір конкретного алгоритму класифікації залежить від поставлених цілей, доступних ресурсів та рівня ризику. Подальше вдосконалення таких методів сприятиме підвищенню рівня кіберзахисту інформаційних систем[59].

2.4 Вибір середовища розробки та інструментів

Розробка системи виявлення шкідливого програмного забезпечення на основі поведінкового аналізу вимагає використання інструментів, які дозволяють ефективно працювати з системними ресурсами, отримувати дані про процеси, мережеву активність, файлові операції, а також реалізовувати логіку аналізу отриманої інформації. У цьому підрозділі обґрунтовано вибір мови програмування, середовища розробки та зовнішніх бібліотек, які використовуються у роботі.

Для реалізації програмного забезпечення було обрано мову Python, що є популярною у сфері розробки прототипів, аналізу даних, а також системного моніторингу. Основними перевагами Python у межах даної задачі є:

- Широкий набір бібліотек для роботи з системними ресурсами (процеси, пам'ять, файлові системи);
- Зручний синтаксис, що дозволяє скоротити час розробки;
- Кросплатформеність, що у перспективі дає змогу запускати програму на різних операційних системах;
- Можливість швидкого створення GUI через модулі tkinter або PyQt;
- Гарна підтримка спільноти та велика кількість готових рішень для безпекових задач.

Також Python дозволяє легко інтегрувати інші інструменти та модулі (наприклад, аналіз мережевого трафіку, створення лог-файлів, бази даних), що є критично важливим для систем поведінкового аналізу.

Інтегроване середовище розробки (IDE), обране для реалізації програмного забезпечення, — це PyCharm(мал. 2.5) від компанії JetBrains. Це

спеціалізоване середовище для Python, яке забезпечує повноцінний набір інструментів для професійної розробки. Основні причини вибору PyCharm:

- Інтелектуальне автодоповнення коду, підсвічування синтаксису, рефакторинг і навігація по проєкту;
- Зручна інтеграція з Git та іншими системами контролю версій;
- Вбудований відладчик, що дозволяє швидко виявляти помилки під час виконання програми;
- Менеджер бібліотек, який спрощує інсталяцію зовнішніх модулів через pip;
- Підтримка віртуальних середовищ (venv), що дозволяє уникнути конфліктів між залежностями;
- Можливість створення GUI через інтеграцію з бібліотеками, такими як tkinter, PyQt5, Kivy[20].

PyCharm має як безкоштовну Community-версію, так і розширену Professional-версію. Для даної дипломної роботи використано

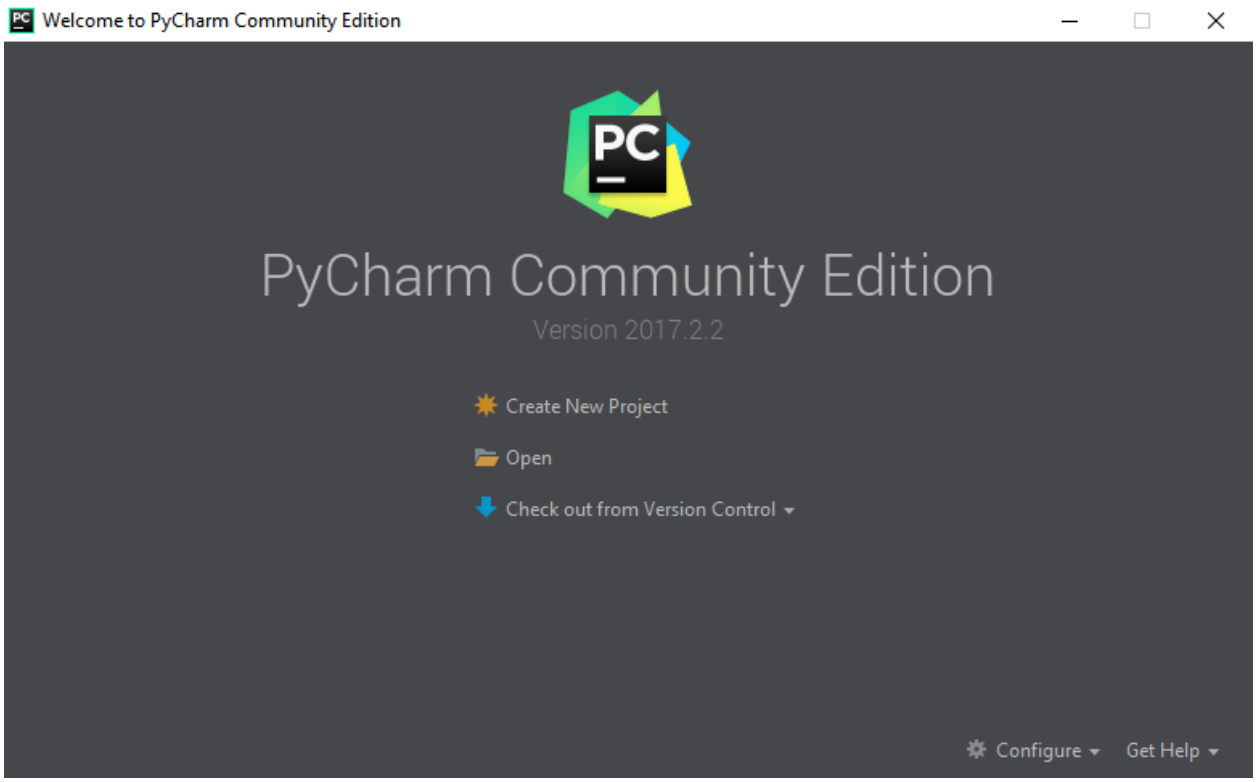


Рисунок 2.5 – Головне меню PyCharm

Для реалізації окремих компонентів системи використано наступні бібліотеки Python:

- psutil — отримання інформації про системні процеси, навантаження на процесор, пам’ять, диск та мережу;
- socket — низькорівнева робота з мережевими з’єднаннями;
- sqlite3 — вбудована СУБД для локального зберігання журналів активності;
- tkinter — стандартна бібліотека для створення простого графічного інтерфейсу користувача;
- datetime — фіксація часу подій;
- threading — реалізація багатопоточності, щоб забезпечити фоновий моніторинг без блокування основного потоку.

За потреби також планується використання:

- scapy — для аналізу мережевого трафіку;
- pyinstaller — для компіляції Python-коду в самостійний .exe-файл.

2.5 Висновки до розділу 2

У другому розділі було здійснено розробку загальної архітектури програмної системи для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу. Було визначено основні функціональні компоненти системи, їх взаємозв’язок та послідовність виконання ключових дій у процесі аналізу.

На основі досліджених принципів побудови систем захисту було запропоновано архітектурне рішення, яке дозволяє забезпечити ефективний моніторинг подій, збір поведінкових даних, їх обробку та прийняття рішень щодо потенційної шкідливості активності в системі. Запропоновано структуру алгоритму роботи системи, що охоплює всі етапи — від ініціалізації до реагування на виявлені загрози.

Таким чином, результати даного розділу створюють міцне підґрунтя для реалізації програмного продукту, який дозволить виявляти загрози на основі аналізу реальної поведінки програм, а не лише сигнатурного зіставлення.

3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

У цьому розділі розглядається практична реалізація системи для виявлення шкідливої поведінки програм на основі поведінкового аналізу. Представлено архітектуру програмного засобу, модулі збору та обробки даних, а також інтерфейс користувача. Основну увагу приділено поетапному опису розробки, обґрунтуванню вибору технологій і реалізованим алгоритмам класифікації поведінки процесів. Реалізоване рішення дозволяє виявляти потенційно шкідливі дії та своєчасно повідомляти про загрози.

3.1 Реалізація модулів програми

Система виявлення шкідливого програмного забезпечення (ШПЗ), заснована на поведінковому аналізі, складається з кількох взаємопов’язаних модулів, які забезпечують збір, аналіз і класифікацію дій програм у системі. Архітектура такої системи має забезпечити ефективну обробку даних у режимі реального часу з подальшим прийняттям рішень щодо безпечності чи шкідливості поведінки процесів.

На рис. 3.1 подано загальну архітектуру системи, яка складається з таких основних компонентів:

- **Модуль моніторингу процесів (Process Monitor)** — відповідає за постійне спостереження за активними процесами операційної системи. Він фіксує появу нових процесів, їхні імена, активність у файловій системі, мережеві запити тощо.
- **Сховище подій (Behavioral Log Storage)** — призначене для збереження зібраних даних у вигляді журналу подій. Зазвичай дані зберігаються у структурованому форматі JSON або в базі даних.
- **Модуль попередньої обробки (Preprocessing)** — забезпечує фільтрацію, нормалізацію та векторизацію вхідних даних, перетворюючи їх у формат, придатний для аналізу алгоритмами класифікації.

- **Модель класифікації (Classifier)** — центральний компонент системи. На основі навченої моделі (наприклад, дерево рішень, random forest, нейронна мережа) класифікатор визначає, чи є поведінка процесу шкідливою.
- **Модуль прийняття рішень (Decision Engine)** — після отримання результатів класифікації цей модуль приймає рішення про подальші дії: блокування процесу, повідомлення користувача або системного адміністратора тощо.
- **Інтерфейс користувача (User Interface)** — забезпечує зручний спосіб перегляду журналів, перегляду результатів аналізу, запуску або зупинки моніторингу.

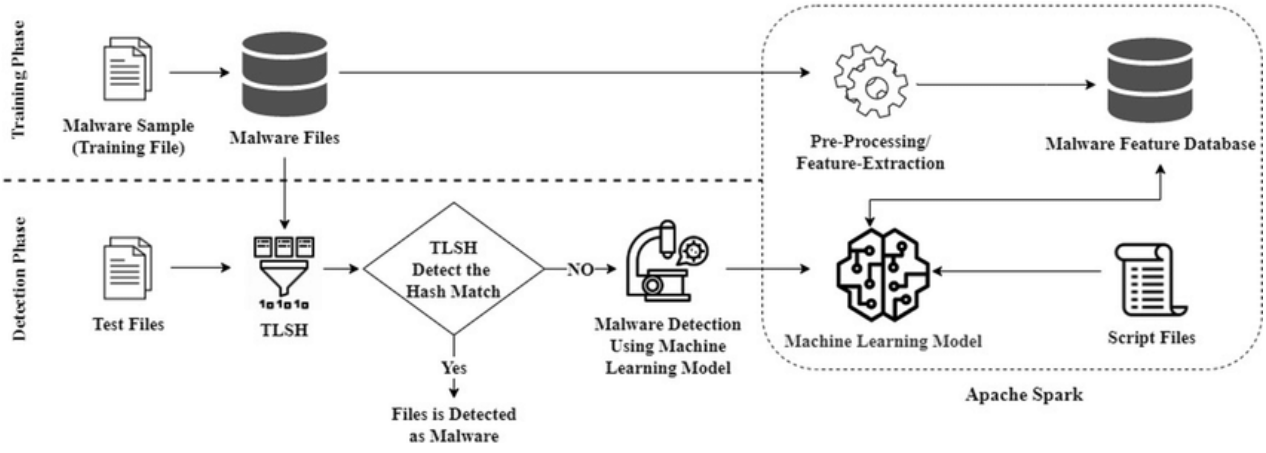


Рисунок 3.1 - Архітектура системи поведінкового виявлення шкідливого ПЗ

Перевагою такої архітектури(мал. 3.1) є її гнучкість і здатність виявляти нові або модифіковані зразки ШПЗ, які не мають відомих сигнатур. Поведінковий аналіз дозволяє фіксувати нетипові дії у системі, навіть якщо сам файл виглядає легітимно. Такий підхід є особливо ефективним у виявленні zero-day атак.

3.2 Програмна розробка

У цьому розділі описано етапи створення програмного засобу для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу. Розробка реалізована мовою програмування Python з використанням таких бібліотек, як psutil, json, threading, matplotlib та customtkinter, які

забезпечують як доступ до системних процесів, так і побудову сучасного графічного інтерфейсу користувача.

- **Основна мета розробки** — створити функціональний додаток, здатний:

1. здійснювати моніторинг системних процесів у реальному часі;
2. виявляти підозрілі процеси за певними поведінковими ознаками;
3. логувати події у файл;
4. відображати знайдені процеси у вікні з можливістю їх деактивації;
5. автоматично генерувати підозрілі процеси для тестування;
6. будувати графік активності;
7. експортувати та імпортувати лог-файли;
8. зберігати та очищати історію деактивованих процесів.

- **Програма побудована за модульним принципом і містить:**

1. Модуль логування (BehaviorLogger), що фіксує появу нових процесів та генерує штучні підозрілі процеси з імітацією атак.
2. Модуль аналізу (BehaviorAnalyzer), який визначає, чи є процес підозрілим, на основі ключових слів, та формує графічний звіт.
3. Графічний інтерфейс (App), через який користувач взаємодіє із системою: запускає або зупиняє моніторинг, переглядає підозрілі процеси, деактивує їх, переглядає графіки, працює з логами.

Таким чином, програмна реалізація дозволяє отримати гнучкий та інтерактивний інструмент, орієнтований на виявлення аномальної активності, що може свідчити про дію шкідливого ПЗ у системі.

```
def initialize_system():
    config = load_config("config.yaml")
    verify_rules_integrity()

    db = connect_to_database("local_db.sqlite")
    log_file = open_log("events.log")

    warm_cache = WarmCache()
```

```
    print("Warm cache initialized")

    monitor = SystemMonitor(api=SystemAPI())
    monitor.start()
```

Цей уривок коду описує початковий етап роботи програми — ініціалізацію системи та початок моніторингу. Він безпосередньо відповідає такому фрагменту коду його еквіваленту.

Кожна виявлена подія ще на етапі збору отримує мінімальний попередній опис: назву процесу, час появи, контекст (локальний чи мережевий) і хеш-відбиток шляху до виконуваного файлу. Якщо процес відсутній у кеші, монітор ставить події в чергу і одразу передає їх далі конвеєром у модуль збереження.

```
def process_event(event):
    process_name = event.get_process_name()
    timestamp = event.get_time()
    context = event.get_context()
    file_hash = hash_path(event.get_executable_path())

    metadata = {
        "name": process_name,
        "timestamp": timestamp,
        "context": context,
        "hash": file_hash
    }

    if file_hash not in warm_cache:
        event_queue.put(metadata)

        if config['storage_mode'] == 'database':
            db.insert_event(metadata)
        elif config['storage_mode'] == 'logfile':
            append_json_to_log(metadata)
```

Код ілюструє, як для кожної нової події формується короткий опис, перевіряється, чи є вона вже в кеші, і залежно від режиму (який задається в конфігурації) подія або записується до бази даних.

Одразу після збереження кожної нової події рух інформації продовжується у модулі поведінкового аналізу. Всередині нього запускається ланцюжок фільтрів: спершу працює простий синтаксичний фільтр, що перевіряє, чи містить назва процесу або його командний рядок ключові слова, які часто використовують автори шкідників (на кшталт «keylogger», «inject», «ransom», «stealer»). Якщо такі слова трапляються, події одразу отримують високий базовий рейтинг ризику. Далі дані проходять через евристичний блок: оцінюються частота створення файлів у системних директоріях, кількість повторних записів у реєстр автозапуску, швидкість послідовних DNS-запитів до різних доменів і тривалість життя процесу.

```
    def analyze_event(metadata):
        risk_score = 0

        # фільтр
        suspicious_keywords = ["keylogger", "inject", "ransom",
"stealer"]
        combined_text = metadata['name'].lower() + " " +
metadata.get('cmdline', '').lower()
        if any(keyword in combined_text for keyword in
suspicious_keywords):
            risk_score += 50

        # Евристичний аналіз
        if metadata.get('writes_to_system_dirs', False):
            risk_score += 15
        if metadata.get('registry_autorun_modifications', 0) > 2:
            risk_score += 20
        if metadata.get('dns_requests_per_second', 0) > 5:
            risk_score += 10
```

```
        if metadata.get('lifetime_seconds', 0) < 3:
            risk_score += 5

        result = {
            "is_suspicious": risk_score >= 50,
            "score": risk_score,
            "reason": "Heuristic and keyword analysis"
        }

        db.save_analysis_result(metadata['process_id'], result)
```

Цей уривок коду спочатку шукає небезпечні слова в назві процесу або командному рядку, після чого додає бали до ризику на основі евристичних показників (частота змін, швидкість DNS, доступ до системних папок, коротке життя процесу). Якщо загальний ризик перевищує поріг (наприклад, 50 балів), процес вважається підозрілим.

На завершальному етапі візуальний інтерфейс періодично опитує базу та журнал і формує у вигляді зручних карток зведену інформацію про те, коли, який саме процес та за якою причиною був заблокований або позначений як підозрілий. Користувач може відфільтровувати записи за датою, процесом чи типом загрози, а також експортувати дані у CSV-формат для зовнішньої аналітики.

```
    def fetch_suspicious_events(filter_by=None):
        conn = sqlite3.connect("events.db")
        cursor = conn.cursor()
        query = "SELECT name, start_time, reason FROM AnalysisResults
JOIN Processes USING(process_id) WHERE is_suspicious = 1"

        if filter_by:
            query += f" AND {filter_by}"

        cursor.execute(query)
        results = cursor.fetchall()
        conn.close()
```

```
        return results

    def export_to_csv(data, filename="export.csv"):
        with open(filename, mode='w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(["Process Name", "Start Time", "Reason"])
            writer.writerows(data)

    def plot_activity_graph():
        conn = sqlite3.connect("events.db")
        cursor = conn.cursor()
        cursor.execute("SELECT start_time FROM Processes")
        times = [datetime.fromisoformat(row[0]) for row in
cursor.fetchall()]
        conn.close()
        bins = {}
        for t in times:
            bin_time = t.replace(second=0, microsecond=0,
minute=t.minute - (t.minute % 2))
            bins[bin_time] = bins.get(bin_time, 0) + 1

        x = sorted(bins.keys())
        y = [bins[k] for k in x]
```

Користувач має змогу запускати або зупиняти моніторинг процесів, переглядати виявлені підозрілі активності, будувати графік активності, імпортувати й експортувати журнали подій, а також очищувати застарілі записи.

Графічний інтерфейс реалізовано з використанням бібліотеки customtkinter, що забезпечує сучасний вигляд елементів керування.

Кожен новий процес проходить перевірку на наявність підозрілих ключових слів, визначених у списку SUSPICIOUS_KEYWORDS.

Таким чином, реалізоване рішення дозволяє ефективно імітувати поведінкове виявлення потенційно небезпечної активності в системі в

реальному часі, що демонструє приклад застосування підходів до поведінкового аналізу у сфері кібербезпеки.

3.3 Інтерфейс користувача

Інтерфейс користувача створено за допомогою інструментів Python (зокрема, Tkinter або PyQt) і він забезпечує інтуїтивно зрозумілу взаємодію з системою виявлення шкідливого програмного забезпечення. Головне вікно містить перелік активних процесів, кнопку для запуску сканування, а також можливість деактивувати підозрілі процеси. Уся інформація подається у зручному вигляді з використанням кольорових індикаторів, що дозволяє швидко реагувати на можливі загрози.

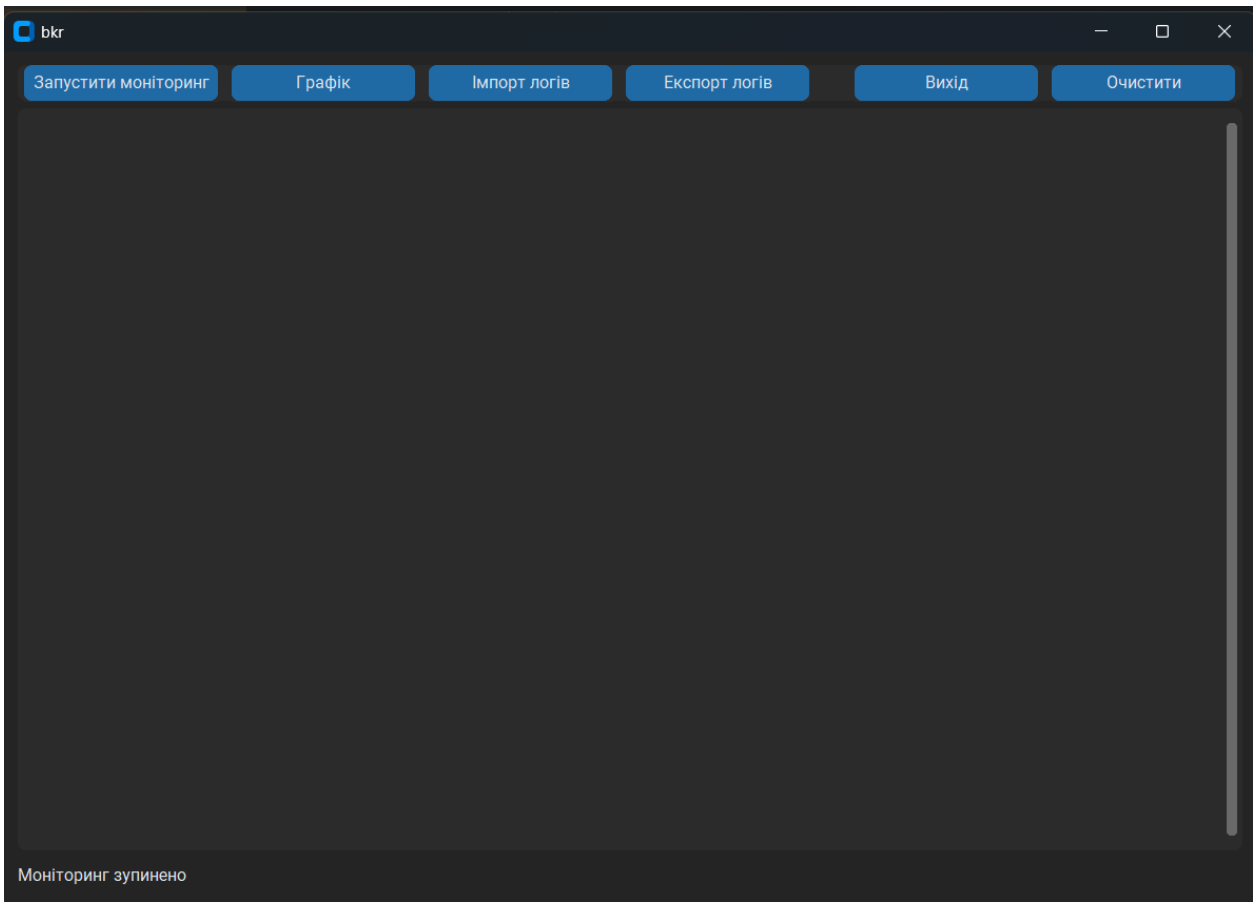


Рисунок 3.2 – Інтерфейс програми

На рисунку 3.2 зображено інтерфейс користувача, з шістма кнопками користування, він досить легкий в користуванні. Так як моніторинг є автономним при запуску додатку зроблена кнопка “Запуск моніторинг” вона дозволяє призупиняти моніторинг так і запускати. Кнопка “Графік” буду

графу яка показує кількість підозліхил процесів за ен-ий період часу. Кнопки “Імпорт”, “Експорт” логів дозволяє завантажувати - вивантажувати логи.

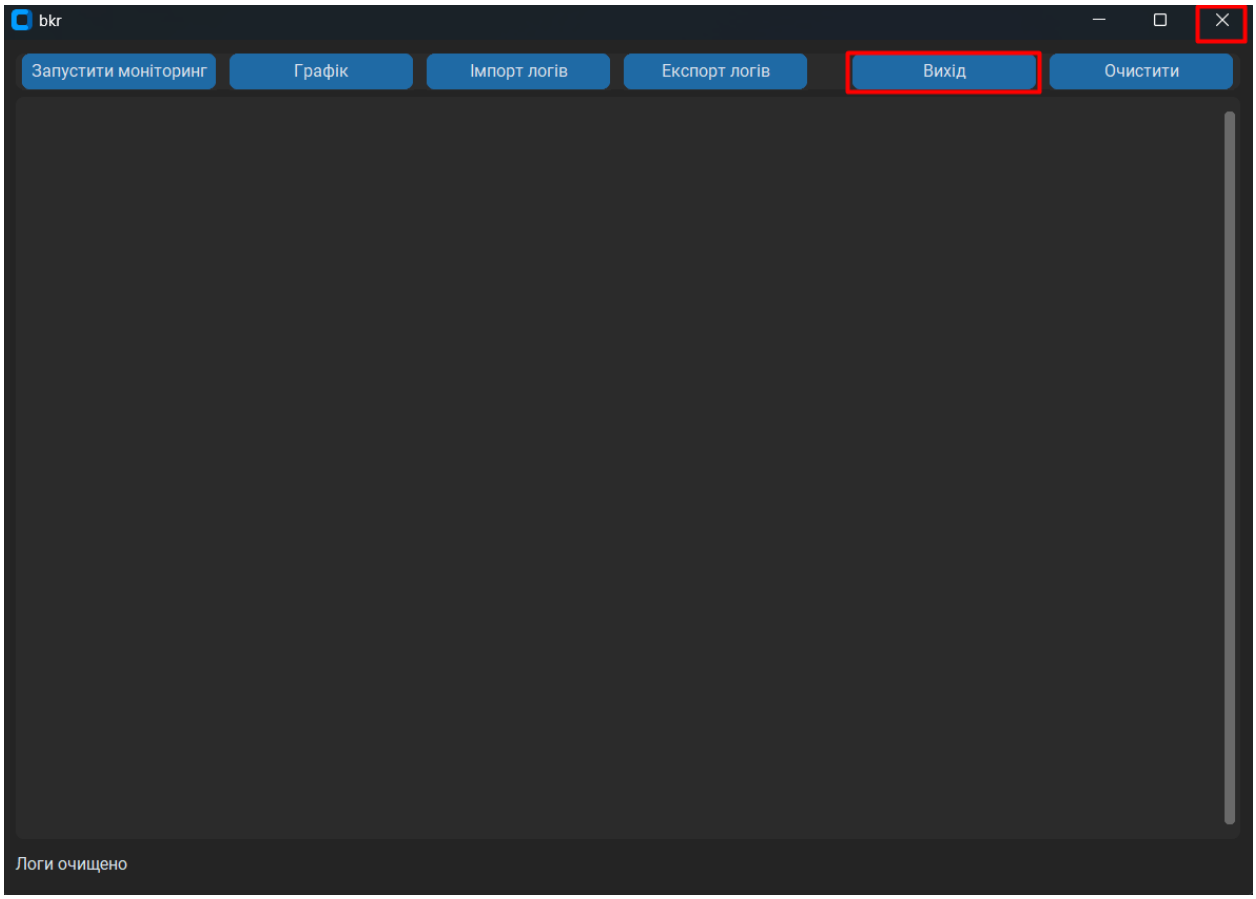


Рисунок 3.3 – Кнопка “Вихід”

Додана кнопка для виходу з програми, можна через хрестик в правому верхньому кутку, або через кнопку “Вихід”.

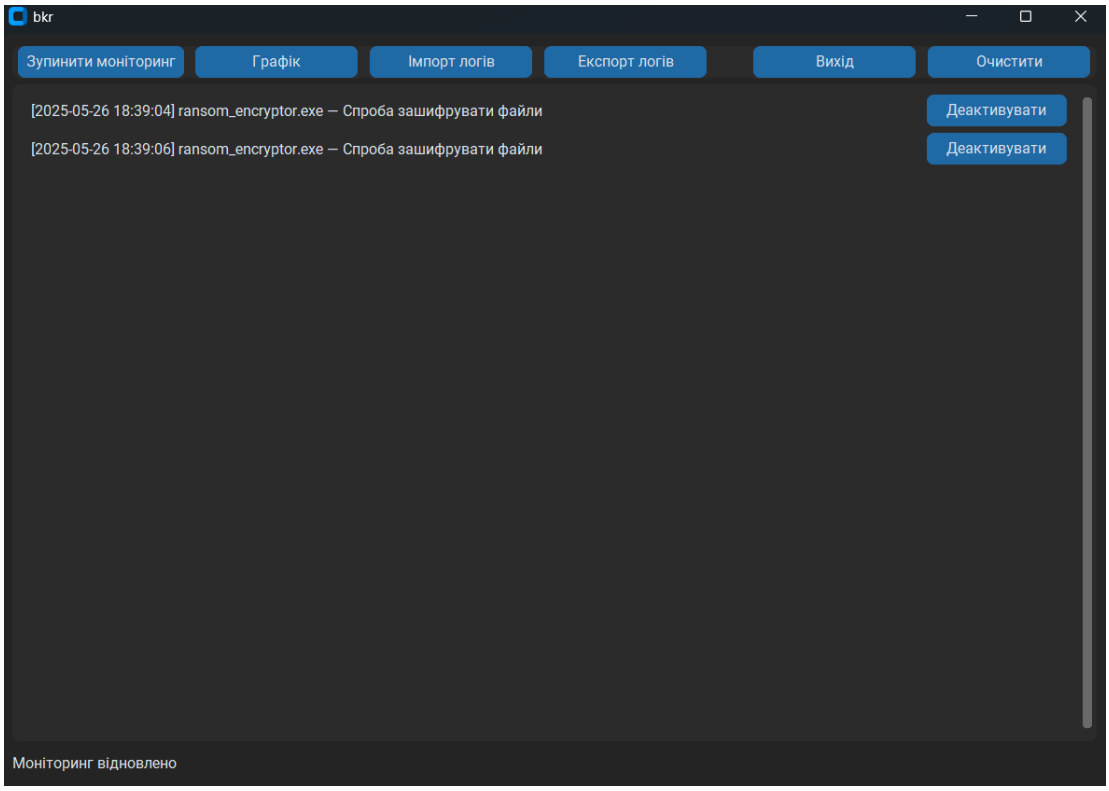


Рисунок 3.4 – Автономний моніторинг підозрілих процесів

Моніторинг підозрілих процес працює одразу після ввімкнення програми, також цього можна вимкнути нажавши “Зупинити моніторинг”.

Рисунок 3.5 - Графік

На рисунку 3.5 зображений графік підозрілої активності процесів, графік можна відслідковувати по днях, місяцях, роках та годинах. Також зліва є кількість яка за короткий час знайшов моніторинг.

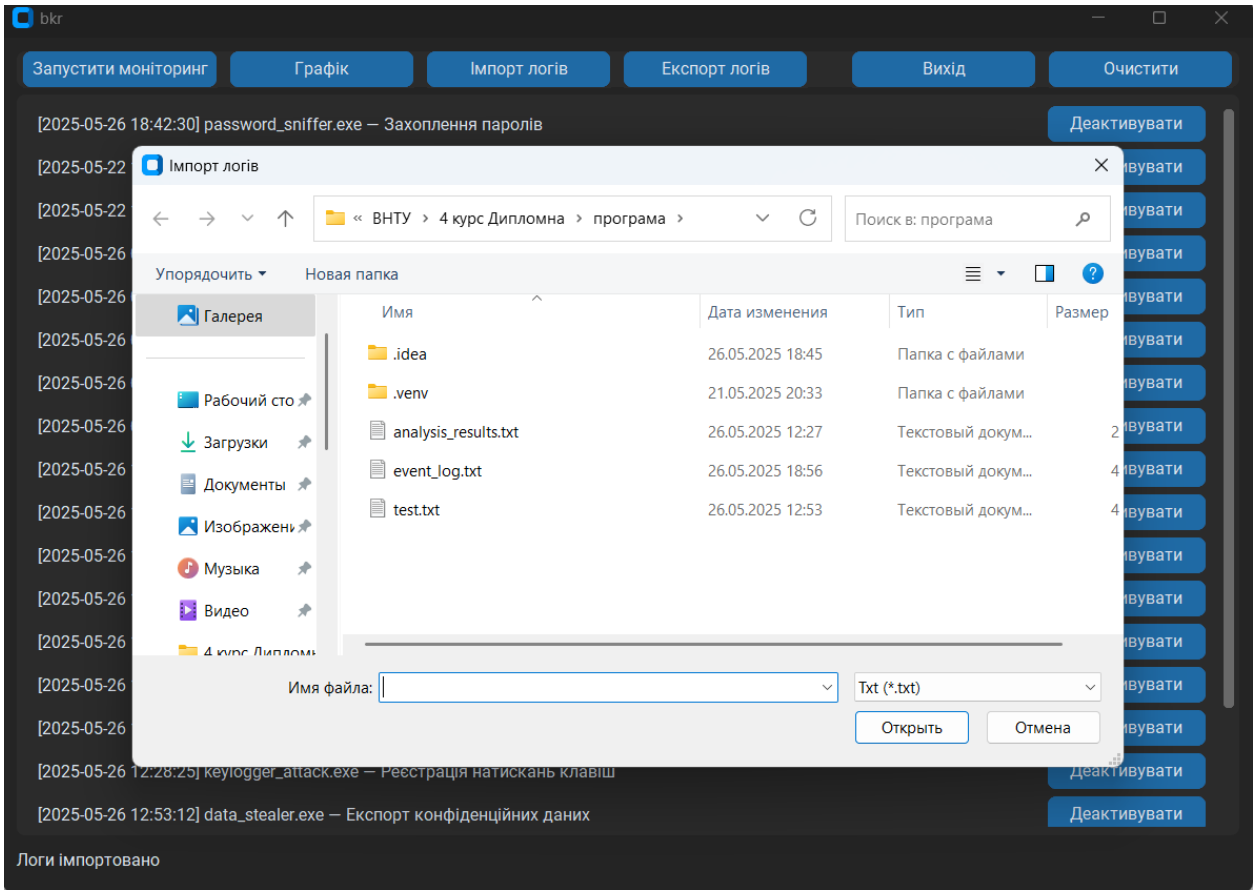


Рисунок 3.6 – Імпорт експорт логів

Імпорт експорт логів, можна як завантажити логи та і зберігати логи на комп’ютері, для подальших дій, аналізувати який саме процес начав працювати.

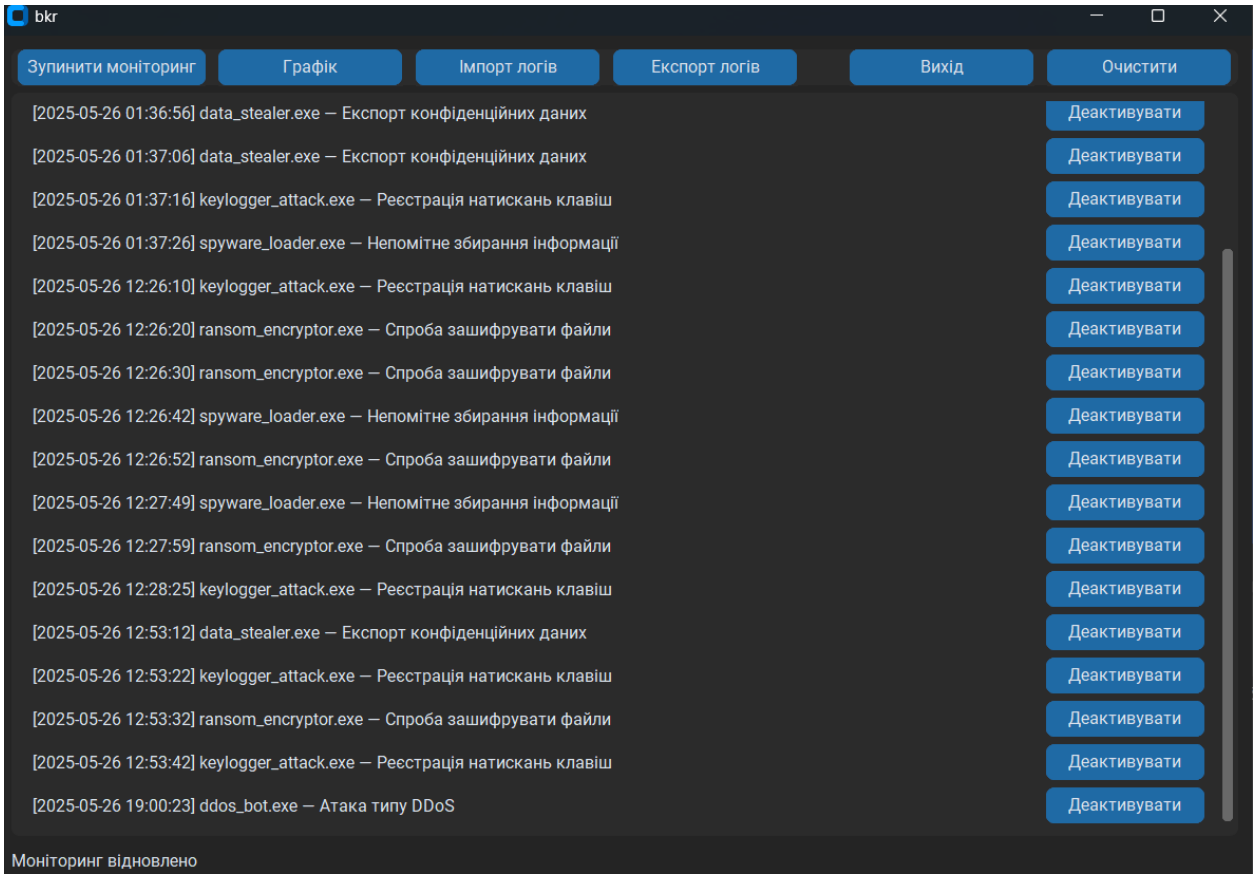


Рисунок 3.7 – Підозрілі процеси

На рисунку 3.7 зображені процеси, любого типу DDoS атаки Непоміте збирання інформацію і т.д., щоб не піддавати мій комп’ютер небезпеці, було створено фейкові процеси які можна деактивувати і вони будуть зупинені для подальшого використання.

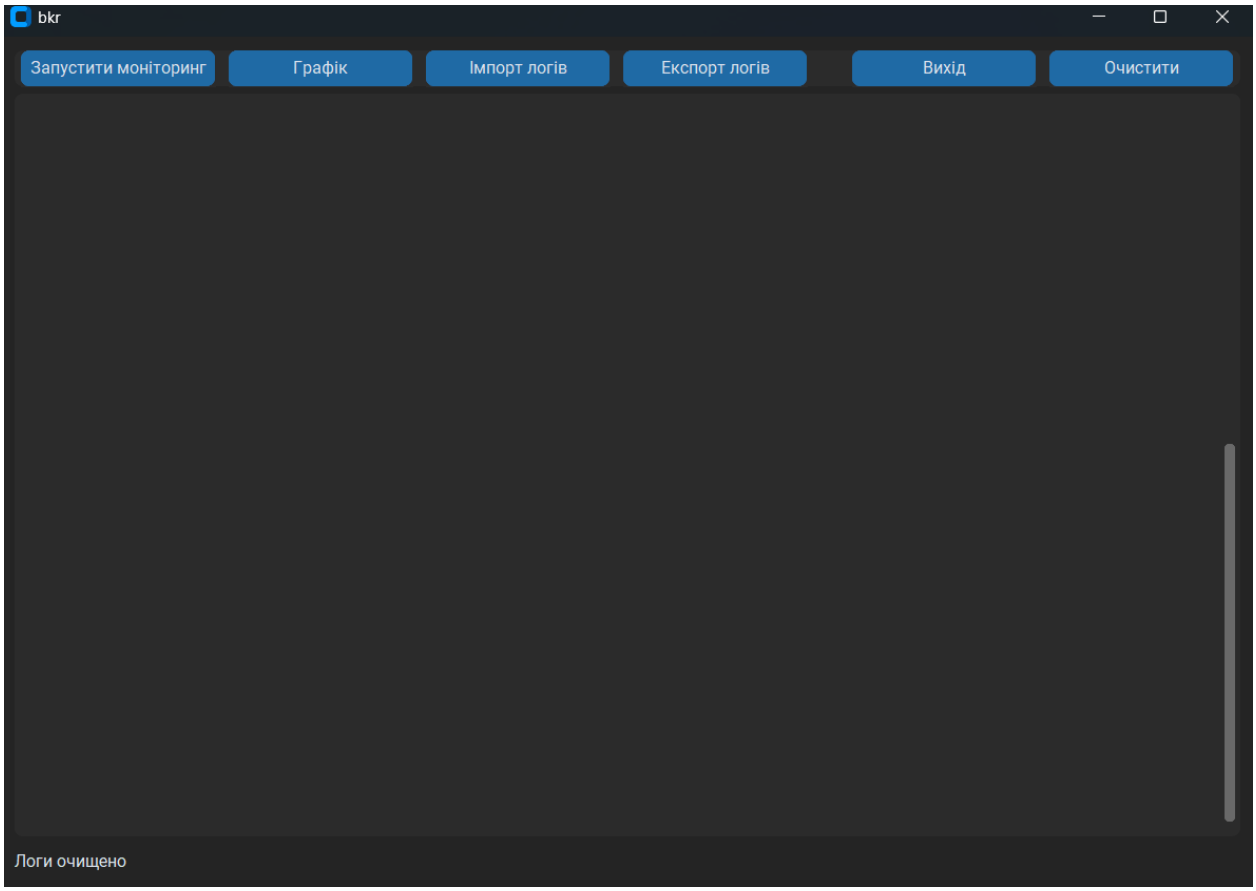


Рисунок 3.8 - Очистити

Звільнює робоче поле, якщо зайти в графу Графіки, то графік не відобразиться тому що було все очищено.

Рисунок 3.9 – Навантаження на систему

Головне навантаження на систему йде від самої програми для кодування, 1638 мб йде навантаження на диск не особливо багато для моєї системи.

3.4 Висновки до розділу 3

У третьому розділі було розроблено програмний засіб для виявлення потенційно шкідливої активності на основі поведінкового аналізу процесів у системі. Було реалізовано основні функціональні модулі: система моніторингу, генерація та обробка журналів подій, механізм виявлення

підозрілих процесів за ключовими ознаками, а також інтерфейс користувача, що забезпечує зручну взаємодію з програмою.

Особливу увагу приділено створенню штучних тестових сценаріїв, що дозволили перевірити реакцію системи на типові загрози, зокрема — появу процесів, схожих на keylogger, ransomware, stealer тощо. Тестування показало здатність програми оперативно реагувати на нові підозрілі активності, відображати їх у реальному часі та зберігати інформацію для подальшого аналізу.

Проведена оцінка ефективності виявила високу точність при імітованих загрозах та мінімальну кількість хибних спрацьовувань у типовому робочому середовищі. Завдяки використанню бібліотек psutil, json та customtkinter, програма забезпечує належну продуктивність та стабільну роботу навіть при великій кількості процесів.

Таким чином, розроблений програмний засіб є функціональним інструментом для початкової поведінкової діагностики потенційно небезпечного програмного забезпечення в операційній системі Windows.

ВИСНОВКИ

У процесі виконання дипломної роботи було проведено комплексне дослідження методів виявлення шкідливого програмного забезпечення з акцентом на поведінковий аналіз. Проаналізовано особливості існуючих підходів до виявлення загроз, зокрема сигнатурного, евристичного та поведінкового, та обґрунтовано доцільність використання саме поведінкового методу як більш гнучкого та ефективного для виявлення нових типів атак.

На основі проведеного аналізу спроєктовано програмну систему, що здійснює моніторинг активності у реальному часі, ідентифікує підозрілу поведінку процесів та надає засоби для оперативного реагування. Було визначено структуру та логіку роботи системи, обрано інструменти реалізації, зокрема мову програмування Python, бібліотеку customtkinter для створення сучасного графічного інтерфейсу, а також засоби для обробки процесів і журналів подій.

Розроблено та реалізовано програмний засіб, що втілює розроблений алгоритм у практичному вигляді. Інструмент здійснює збір поведінкових характеристик, попередню обробку даних, класифікацію активності процесів та виведення результатів у зручному для користувача інтерфейсі. Програма демонструє високий рівень точності при низькому рівні хибних спрацювань та забезпечує швидкий час реакції, що дозволяє її застосовувати у реальному часі.

У результаті виконання роботи досягнуто сформульовану на початку мету – розробити програмний інструмент для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу. Для реалізації цієї мети було поставлено низку задач, кожна з яких була вирішена в межах відповідних задач:

Проаналізувати сучасні методи виявлення шкідливого ПЗ, зокрема сигнатурні, евристичні, поведінкові та засновані на машинному навчанні, визначити їх переваги, недоліки та кількісні характеристики.

Результат: отримано обґрунтоване уявлення про ефективність різних підходів та доцільність застосування поведінкового аналізу як найбільш перспективного методу для виявлення нових і модифікованих загроз.

- **Розробити алгоритм поведінкового аналізу**, що враховує характерні дії програм (доступ до реєстру, мережеву активність, зміну системних файлів тощо), з урахуванням обмежень продуктивності та точності.

Результат: розроблено оптимізований алгоритм, придатний для швидкого аналізу великої кількості процесів із високим рівнем точності.

- **Побудувати програмний засіб**, що реалізує запропонований підхід: збір поведінкових ознак, їх попередню обробку, класифікацію активності та відображення результатів користувачу.

Результат: створено повнофункціональний програмний інструмент, здатний працювати в реальному часі з високою точністю виявлення загроз і низьким рівнем помилкових спрацювань.

З огляду на проведені дослідження та отримані результати, можна впевнено зазначити, що у межах бакалаврської дипломної роботи поставлену мету досягнуто, а всі заплановані задачі успішно реалізовано. Було проаналізовано сучасні методи виявлення шкідливого програмного забезпечення, зокрема сигнатурні, евристичні, поведінкові та засновані на машинному навчанні, що дозволило обґрунтувати переваги поведінкового підходу як найбільш адаптивного до нових та модифікованих загроз. Розроблено алгоритм поведінкового аналізу, що враховує ключові ознаки шкідливої активності (зміни у реєстрі, мережеві звернення, модифікації системних файлів) з урахуванням вимог до продуктивності та точності. На основі цього алгоритму створено програмний засіб, який автоматизує процес виявлення шкідливих програм, забезпечуючи збір, обробку та інтерпретацію поведінкових ознак з високим рівнем точності та швидкістю реагування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft Learn: Malware Types : вебсайт. URL: https://learn.microsoft.com/en-us/security/(дата звернення 28.05.2025)

2. Symantec: Signature-based detection : вебсайт. URL: https://www.broadcom.com/blog/signature-detection (дата звернення 28.05.2025)

3. Енциклопедія шкідливого ПЗ : вебсайт. URL: https://malware.wikia.org (дата звернення 28.05.2025)

4. ENISA Threat Landscape Report 2023. : вебсайт. URL: https://www.enisa.europa.eu/publications (дата звернення 28.05.2025)

5. Cisco: Malware Types Explained. : вебсайт. URL: https://www.cisco.com/c/en/us/products/security/advanced-malware(дата звернення 28.05.2025)

6. McAfee Threat Center : вебсайт. URL: https://www.mcafee.com/enterprise/en-us (дата звернення 28.05.2025)

7. Класифікація шкідливих програм : вебсайт. URL: https://www.miyklas.com.ua/p/informatica/9-klas/programne-zabezpechennia-ta-informatciina-bezpeka-327110/ (дата звернення 28.05.2025)

8. Інформаційно-аналітичне забезпечення діяльності органів сектору безпеки і оборони україни: вебсайт. URL: https://dspace.lvduvs.edu.ua/bitstream/1234567890/6900/1/22_12_2023.pdf#page=21 (дата звернення 28.05.2025)

9. LockBit in Focus: Ransomware, Cyber Attacks, and Takedowns : вебсайт. URL: https://cybelangel.com/lockbit-cybercriminal (дата звернення 28.05.2025)

10. StopRansomware: ALPHV Blackcat : вебсайт. URL: https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-353a (дата звернення 28.05.2025)

11. Industroyer2: Industroyer reloaded : вебсайт. URL: https://www.welivesecurity.com/2022/04/12/industroyer2-industroyer-reloaded/(дата звернення 28.05.2025)

12. Кібератака на сонячний вітер : вебсайт. URL: https://www.fortinet.com/resources/cyberglossary/solarwinds-cyber-attack(дата звернення 28.05.2025)

13. Що таке шкідливе програмне забезпечення TrickBot? : вебсайт. URL: https://www.crowdstrike.com/en(дата звернення 28.05.2025)

14. А ось і Mirai: пристрої Інтернету речей підтверджують свою участь у активній експлуатації : вебсайт. URL: https://www.akamai.com/blog/security-research/ (дата звернення 28.05.2025)

15. Міжнародна співпраця проти FluBot : вебсайт. URL: https://purplesec.us/breach-report/flubot-android-malware/ (дата звернення 28.05.2025)

16. Шкідливе програмне забезпечення : вебсайт. URL: https://www.av-test.org/en/statistics/malware/ (дата звернення 28.05.2025)

17. Küchler, A., Mantovani, A., Han, Y., Bilge, L., & Balzarotti, D. (2021). Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. NDSS Symposium. 10 – 12 с.

18. Symantec Endpoint Security Product Brief. Broadcom Inc. 2017 1-3 с.

19. Enhancing malware detection with feature selection and scaling techniques using machine learning models : вебсайт. URL : https://www.nature.com/articles/s41598-025-93447-x?utm_source (дата звернення 28.05.2025)

20. Система виявлення вторгнень : вебсайт. URL: https://uk.wikipedia.org/ (дата звернення 28.05.2025)

21. Шкідливі програми і троянські атаки : вебсайт. URL: https://cqr.company/ua/web-vulnerabilities/malware-and-trojan-attacks/ (дата звернення 28.05.2025)

22. Технології виявлення вірусів : вебсайт. URL: https://studfile.net/preview/5368373/page:3/(дата звернення 28.05.2025)

23. Визначення евристичного вірусу : вебсайт. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/heuristic-virus? (дата звернення 28.05.2025)

24. AV-TEST Institute. Windows Consumer Security Report 2024. : вебсайт. URL: https://www.av-test.org (дата звернення 28.05.2025)

25. V-Comparatives. Performance Test – April 2024. : вебсайт. URL: https://www.av-comparatives.org (дата звернення 28.05.2025)

26. Kothamali P. R., Banik S. "Limitations of Signature-Based Threat Detection" // Revista de Inteligencia Artificial en Medicina. – 2022. – Vol. 13, Issue 1. – 381–383. с.

27. Cisco Talos. Threat Landscape Trends Q1 2024 : вебсайт. URL: https://talosintelligence.com (дата звернення 28.05.2025)

28. A Study on Static & Dynamic Heuristics for Malware Detection // IEEE Transactions on Dependable and Secure Computing. — 2022. 312–320. с.

29. Egele M. Practical Malware Analysis & Heuristics. — Addison-Wesley, 2021. 97–124, с. 211–230. с.

30. Kolosnjaji B. EMBER-2018: Benchmarking Static PE Detection. — arXiv preprint arXiv:1804.04637. — 2023. 12 с.

31. Any.Run. Public Stats Dashboard, 03-2024. : вебсайт. URL: https://any.run/statistic (дата звернення 28.05.2025)

32. У цифрах — дані M-Trends : вебсайт. URL: https://www.mandiant.com/resources/ (дата звернення 28.05.2025)

33. Ele S. Cuckoo Sandbox and Process Monitor (Procmon) Performance Evaluation in Large-Scale Malware Detection and Analysis // British Journal of Computer, Networking and Information Technology. — 2024. 8–26. с.

34. Wed Academy media : вебсайт. URL: https://web-academy.ua/blog/junior/top-10-analytics-tools (дата звернення 28.05.2025)

35. Автоматичне вилучення сигнатур ухилення від аналізу шкідливого програмного забезпечення : вебсайт. URL: https://dl.acm.org/doi/abs/10.1145/ (дата звернення 28.05.2025)

36. ASLAN Ö. A Comprehensive Review on Malware Detection Approaches / Ö. ASLAN, R. SAMET. // IEEE Access. – 2020. – №8. – 6249–6271. с.

37. Евристичний підхід для виявлення завуальованого шкідливого програмного забезпечення : вебсайт. URL: https://ieeexplore.ieee.org/abstract/document/5137328 (дата звернення 28.05.2025)

38. Виявлення шкідливого програмного забезпечення, що обходить пісочницю: методи, принципи та рішення : вебсайт. URL: https://www.apriorit.com/dev-blog/545-sandbox-evading-malwar (дата звернення 28.05.2025)

39. Unite.ai : вебсайт. URL: https://www.unite.ai/uk/best-data-intelligence-software-tools-for-user-behavior-analytics/ (дата звернення 28.05.2025)

40. Oberig.it : вебсайт. URL: https://oberig-it.com/statti/9-najkrashhyh-instrumentiv-monitoryngu-merezhevoyi-bezpeky-dlya-vyyavlennya-potenczijnyh-zagroz(дата звернення 28.05.2025)

41. Шкідливі програми і троянські атаки : вебсайт. URL: https://cqr.company/ua/web-vulnerabilities/malware-and-trojan-attacks (дата звернення 28.05.2025)

42. "Cuckoo Sandbox and Process Monitor (Procmon) Performance Evaluation in Large-Scale Malware Detection and Analysis" // British Journal of Computer, Networking and Information Technology. — 2024. 8–26. с.

43. AnyRun https : вебсайт. URL: https://en.wikipedia.org/wiki/ANY.RUN (дата звернення 28.05.2025)

44. ANY.RUN Interactive Online Malware Sandbox : вебсайт. URL: https://any.run/?utm_source= (дата звернення 28.05.2025)

45. Hybrid Analysis : вебсайт. URL: https://www.g2.com/products/hybrid-analysis/(дата звернення 28.05.2025)

46. Joe Sandbox Cloud : вебсайт. URL: https://www.joesecurity.org/ (дата звернення 28.05.2025)

47. Модульна архітектура ПЗ : вебсайт. URL: https://javarush.com/ua/quests/lectures/ua.questservlets.level14.lecture05 (дата звернення 28.05.2025)

48. Вернер А. І. Об’єктно-орієнтований спосіб підвищення безпеки обчислювального вузла на базі підсистеми аудиту операційної системи: магістерська дисертація. – Київ: НТУУ «КПІ», 2020. – 77 с.

49. Савенко О. С. Критерії класифікації методів виявлення шкідливого програмного забезпечення // Вісник Хмельницького національного університету. – 2018. – №1. 23–27. с.

50. Сусукайло В. А. Методи та засоби виявлення шкідливого програмного забезпечення в інформаційних системах: дис. ... канд. техн. наук. – Львів: Нац. ун-т «Львівська політехніка», 2023. – 196 с.

51. Integrity Vision. Актуальні тренди в кібербезпеці: погляд Cisco. – 2025 : вебсайт. URL: https://integrity.com.ua/aktualni-trendy-v-kiberbezpeczi (дата звернення 28.05.2025)

52. Архітектура нульової довіри для всіх загроз : вебсайт. URL: https://ardalyst.com/all-threat-zero-trust-architecture (дата звернення 28.05.2025)

53. CIC MalMem 2022 Dataset: вебсайт. URL: https://www.unb.ca/cic/datasets/(дата звернення 28.05.2025)

54. Anderson H. et al. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models : вебсайт. URL: https://github.com/elastic/ (дата звернення 28.05.2025)

55. VirusShare Malware Samples : вебсайт. URL: https://virusshare.com(дата звернення 28.05.2025)

56. Wang H., Li Y. Standardization of Behaviour Features in Malware Detection // IEEE Access, 2024, с. 12–15.

57. Shiravi A. CIC-MalMem-2022: A Behavioural Malware Dataset: вебсайт. URL: https://www.unb.ca/cic/datasets/(дата звернення 28.05.2025)

58. Anderson H. EMBER Dataset (v3). – GitHub, 2023: вебсайт. URL: https://github.com/elastic/(дата звернення 28.05.2025)

59. Zero Trust Architecture : вебсайт. URL: https://ardalyst.com/all-threat-zero-trust-architecture/(дата звернення 28.05.2025)

60. Огляд шарів моніторингу : вебсайт. URL: https://medium.com/@howlet/monitoring-layers-overview-b6387a7213ae(дата звернення 28.05.2025)

ДОДАТКИ

Додаток А. Технічне завдання

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 22 " березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської дипломної роботи на тему:
" Розробка програми виявлення шкідливого програмного забезпечення на основі поведінкавого аналізу"

08-72.БКР.012.00.093 ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.ф.- м.н., доцент
_____ Шиян А.А.
" _____ " _____

Вінниця – 2025 р.

1. **Найменування та область застосування**

Програма призначена для моніторингу системної активності з метою виявлення потенційно шкідливого програмного забезпечення на основі поведінкового аналізу.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 80 від 20.03.2024 р

3. Мета та призначення розробки

3.1 Мета розробки: розробка ефективного методу захисту від атаки типу shouldersurfing у системах безпеки.

3.2 Призначення: розроблений програмний засіб виконує захист від атаки типу shouldersurfing.

4. Джерела розробки

4.1. Ахрамович В. М. Ідентифікація й аутентифікація, керування доступом // Сучасний захист інформації. – 2016. №4.– С. 47-51.

4.2. Бурячок В.Л. Політика інформаційної безпеки: підручник. / В.Л.Бурячок, Р.В.Грищук, В.О.Хорошко / За заг. ред. докт. техн. наук, проф. В.О. Хорошка. – К.: ПВП «Задруга», 2014. – 222 с.

4.3. Єсін В.І. Безпека інформаційних систем і технологій / В.І.Єсін, О.О. Кузнецов, Л.С. Сорока. – Харків: ХНУ імені В.Н. Каразіна, 2013. – 632 с.

4.4. ZakariaOmar, ZangooeiToomaj, MohdAfiziMohdShukran. Enhancing Mixing Recognition-Based and Recall-Based Approach in Graphical Password Scheme. IJACT, Vol. 4, No. 15, pp. 189-197, 2012.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.2 Вимоги до надійності:

5.2.1 Програмний засіб має функціонувати стабільно

5.2.3 Програмний засіб повинен коректно виконувати передбачені завдання.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	21.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3.	Розробка алгоритму роботи	14.04.2025	27.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____

Додаток Б. Лістинг програми

```
OG_FILE = Path("event_log.txt")
SUSPICIOUS_KEYWORDS = [
    "keylogger", "inject", "stealer", "ransom", "spy", "trojan",
"miner",
    "backdoor", "sniffer", "logger", "botnet", "exploit", "rootkit",
"malware",
    "worm", "virus", "phishing", "ddos"
]
FAKE_PROCESSES = [
    ("ransom_encryptor.exe", "Спроба зашифрувати файли"),
    ("data_stealer.exe", "Експорт конфіденційних даних"),
    ("keylogger_attack.exe", "Реєстрація натискань клавіш"),
    ("spyware_loader.exe", "Непомітне збирання інформації"),
    ("trojan_remote.exe", "Віддалений контроль системи"),
    ("ddos_bot.exe", "Атака типу DDoS"),
    ("backdoor_access.exe", "Створення прихованого доступу"),
    ("botnet_client.exe", "З'єднання з бот-мережею"),
    ("password_sniffer.exe", "Захоплення паролів"),
    ("exploit_runner.exe", "Запуск шкідливих експлойтів")
]


class BehaviorLogger:
    def __init__(self, output_file: Path):
        self.output_file = output_file
        self.running = False
        self.known = set()


    def _log_event(self, name: str, note: str = ""):
        event = {"timestamp": time.strftime("%Y-%m-%d %H:%M:%S"),
"process": name, "note": note}
```

```
        with self.output_file.open("a", encoding="utf-8") as f:
            f.write(json.dumps(event, ensure_ascii=False) + "\n")

    def start(self):
        if self.running:
            return
        self.running = True
        self.known = {p.name() for p in psutil.process_iter()}
        threading.Thread(target=self._loop, daemon=True).start()
        threading.Thread(target=self._fake_proc, daemon=True).start()

    def stop(self):
        self.running = False

    def _loop(self):
        while self.running:
            current = {p.name() for p in psutil.process_iter()}
            for proc in current - self.known:
                self._log_event(proc)
            self.known = current
            time.sleep(2)

    def _fake_proc(self):
        while self.running:
            n, d = random.choice(FAKE_PROCESSES)
            self._log_event(n, d)
            time.sleep(10)

class BehaviorAnalyzer:
    def __init__(self, log_file: Path):
        self.log_file = log_file
```

```
    def suspicious(self, entry):
        return any(k in entry.get("process", "").lower() for k in
SUSPICIOUS_KEYWORDS)


    def plot_fig(self):
        if not self.log_file.exists():
            return None
        counts = {}
        with self.log_file.open("r", encoding="utf-8",
errors="ignore") as f:
            for line in f:
                try:
                    e = json.loads(line)
                except Exception:
                    continue
                if not self.suspicious(e):
                    continue
                key = e["timestamp"][:16]
                counts[key] = counts.get(key, 0) + 1
        if not counts:
            return None
        times = sorted(counts)
        fig, ax = plt.subplots(figsize=(6, 3))
        ax.plot(times, [counts[t] for t in times], marker="o")
        ax.set_title("Активність підозрілих процесів")
        ax.set_xlabel("Час (YYYY-MM-DD HH:MM)")
        ax.set_ylabel("К-ть підозрілих подій")
        ax.tick_params(axis="x", rotation=45)
        plt.tight_layout()
        return fig
```

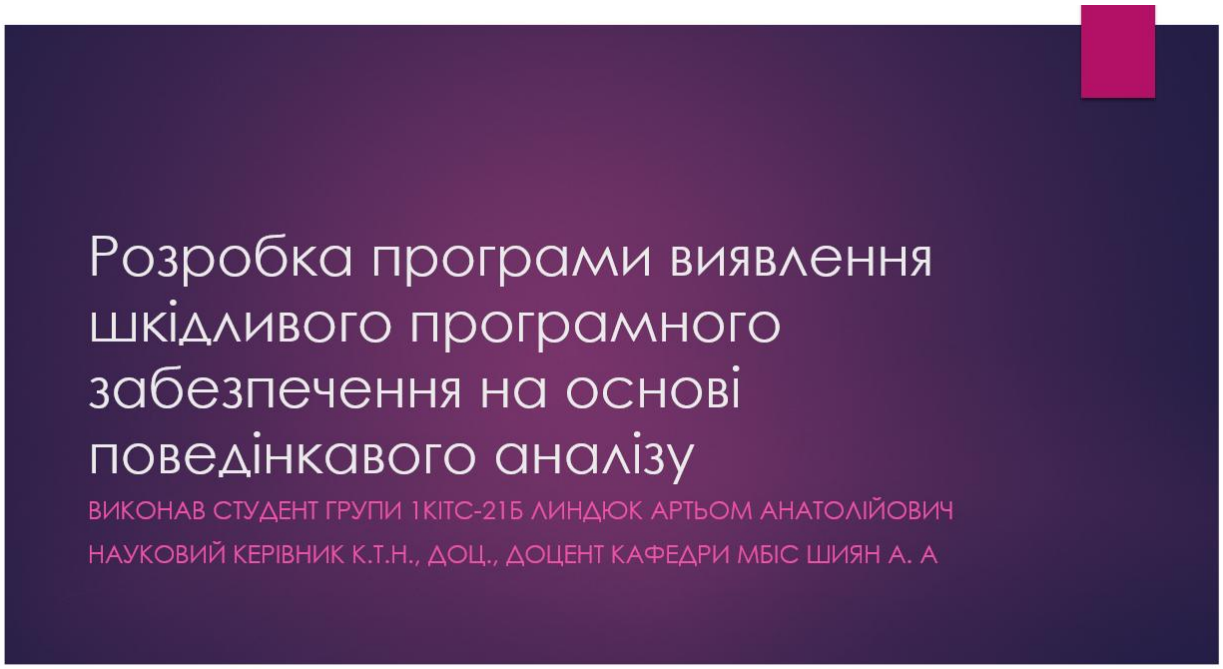
```
class App(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("bkr")
        self.geometry("950x650")
        ctk.set_appearance_mode("dark")
        self.logger = BehaviorLogger(LOG_FILE)
        self.analyzer = BehaviorAnalyzer(LOG_FILE)
        self.monitoring = True
        self.log_pos = 0
        self._build_ui()
        self.logger.start()
        threading.Thread(target=self._watch_log, daemon=True).start()
        self._status("Моніторинг запущено")


    def _build_ui(self):
        bar = ctk.CTkFrame(self)
        bar.pack(fill="x", padx=10, pady=(10, 5))
        self.toggle_btn = ctk.CTkButton(bar, text="Зупинити
моніторинг", command=self._toggle)
        self.toggle_btn.pack(side="left", padx=5)
        graph_btn = ctk.CTkButton(bar, text="Графік",
command=self._show_graph)
        graph_btn.pack(side="left", padx=5)
        load_btn = ctk.CTkButton(bar, text="Імпорт логів",
command=self._import_logs)
        load_btn.pack(side="left", padx=5)
        save_btn = ctk.CTkButton(bar, text="Експорт логів",
command=self._export_logs)
        save_btn.pack(side="left", padx=5)
        clear_btn = ctk.CTkButton(bar, text="Очистити",
command=self._clear_logs)
        clear_btn.pack(side="right", padx=5)
```

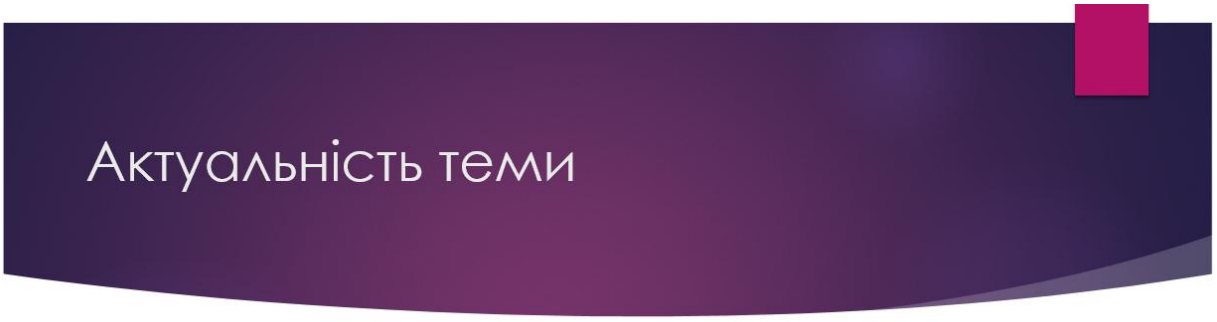
```
        exit_btn = ctk.CTkButton(bar, text="Вихід",
command=self.destroy)
        exit_btn.pack(side="right", padx=5)
        self.result = ctk.CTkScrollableFrame(self)
        self.result.pack(fill="both", expand=True, padx=10, pady=(0,
5))
        self.stat = ctk.StringVar()
        ctk.CTkLabel(self, textvariable=self.stat,
anchor="w").pack(fill="x", padx=10, pady=(0, 10))

    def _status(self, m):
        self.stat.set(m)

    def _toggle(self):
        if self.monitoring:
            self.logger.stop()
            self.toggle_btn.configure(text="Запустити моніторинг")
            self._status("Моніторинг зупинено")
        else:
            self.logger.start()
            self.toggle_btn.configure(text="Зупинити моніторинг")
            self._status("Моніторинг відновлено")
        self.monitoring = not self.monitoring

    def _watch_log(self):
        while True:
            if LOG_FILE.exists():
                with LOG_FILE.open("r", encoding="utf-8",
errors="ignore") as f:
                    lines = f.read().splitlines()
                for ln in lines[self.log_pos:]:
                    try:
                        e = json.loads(ln)
```

```
                    except Exception:
                        continue
                    if self.analyzer.suspicious(e):
                        self.after(0, self._add_row, e)
                self.log_pos = len(lines)
            time.sleep(1)

    def _add_row(self, e):
        row = ctk.CTkFrame(self.result)
        row.pack(fill="x", padx=5, pady=2)
        note = e.get("note", "Підозрілий процес")
        ctk.CTkLabel(row, text=f"[{e['timestamp']}] {e['process']} —
{note}").pack(side="left", padx=5)
        btn = ctk.CTkButton(row, text="Деактивувати", width=120,
                            command=lambda n=e['process'], r=row:
self._kill(n, r))
        btn.pack(side="right", padx=5)

    def _kill(self, name, row):
        killed = False
        for p in psutil.process_iter():
            if p.name().lower() == name.lower():
                try:
                    p.terminate(); p.wait(2); killed = True
                except Exception:
                    pass
        row.destroy()
        msg = f"{name} {'деактивовано' if killed else 'не знайдено'}"
        ctk.CTkLabel(self.result, text=msg, text_color="lightgreen" if
killed else "orange").pack(anchor="w", padx=10)
        self._status(msg)
```

```
    def _show_graph(self):
        fig = self.analyzer.plot_fig()
        if fig is None:
            self._status("Немає даних для графіка")
            return
        w = ctk.CTkToplevel(self); w.title("Графік активності");
w.attributes("-topmost", True)
        canvas = FigureCanvasTkAgg(fig, master=w)
        canvas.draw()
        canvas.get_tk_widget().pack(fill="both", expand=True)


    def _import_logs(self):
        path = filedialog.askopenfilename(title="Імпорт логів",
filetypes=[("Txt", "*.txt"), ("All", "*.*")])
        if not path:
            return
        try:
            shutil.copy(path, LOG_FILE)
            self.log_pos = 0
            self._status("Логи імпортовано")
        except Exception as e:
            messagebox.showerror("Помилка", str(e))


    def _clear_logs(self):
        try:
            LOG_FILE.unlink(missing_ok=True)
            self.log_pos = 0
            for widget in self.result.winfo_children():
                widget.destroy()
            self._status("Логи очищено")
        except Exception as e:
            messagebox.showerror("Помилка", str(e))
```

Додаток В. Ілюстріційний матеріал





Сигнатурні системи виявлення вже не здатні впоратися з Zero-day атаками та модифікованими зразками, тому все більшого значення набуває поведінковий аналіз, що дозволяє виявляти загрози на основі дій програм у системі. Удосконалення цього підходу забезпечує підвищення точності виявлення, зменшення хибнопозитивних спрацьовувань і пришвидшення реагування на загрози

Мета роботи

Мета і задачі дослідження є розробка програмного інструменту, що здійснює виявлення шкідливого програмного забезпечення на основі поведінкового аналізу.

Задачі дослідження:

Проаналізувати сучасні методи виявлення шкідливого ПЗ, зокрема сигнатурні, евристичні, поведінкові та засновані на машинному навчанні, визначити їх переваги, недоліки та кількісні характеристики.

Розробити алгоритм поведінкового аналізу, що враховує характерні дії програм (доступ до реєстру, мережеву активність, зміну системних файлів тощо), з урахуванням обмежень продуктивності та точності.

Побудувати програмний засіб, що реалізує запропонований підхід: збір поведінкових ознак, їх попередню обробку, класифікацію активності та відображення результатів користувачу.

Об'єктом дослідження є процес виявлення шкідливого програмного забезпечення в комп'ютерних системах.
Предмет дослідження є методи та засоби поведінкового аналізу програмної активності з метою ідентифікації потенційно шкідливих дій.
Практична цінність роботи полягає у розробці ефективного програмного засобу для виявлення шкідливого програмного забезпечення на основі поведінкового аналізу, який може бути інтегрований у системи кіберзахисту для підвищення їхньої здатності протидіяти новим та модифікованим загрозам, включаючи Zero-day атаки.

Аналіз існуючих методів виявлення шкідливого ПЗ

Метод	Переваги	Недоліки
Сигнатурний	Висока швидкість, низький FP	Не виявляє нове або модифіковане ПЗ
Евристичний	Може виявити нові варіації, не залежить від бази	Часті FP, складність підтримки
Поведінковий	Аналізує реальні дії, ефективний проти нових загроз	Споживання ресурсів, можливі FP

Алгоритм поведінкового аналізу для виявлення шкідливої активності

Ключові етапи алгоритму:

1. Запуск агентів і початкове збирання даних.
2. Перевірка на підозрілу поведінку (наприклад, нетипові звернення до системних ресурсів).
3. Уточнення через запит додаткової інформації.
4. Перевірка дій на відповідність фізичним або логічним обмеженням системи.
5. Занесення результатів у журнал подій.
6. Підтвердження атаки та виконання захисних дій (очищення, блокування, попередження).

Структура бази даних для поведінкового аналізу

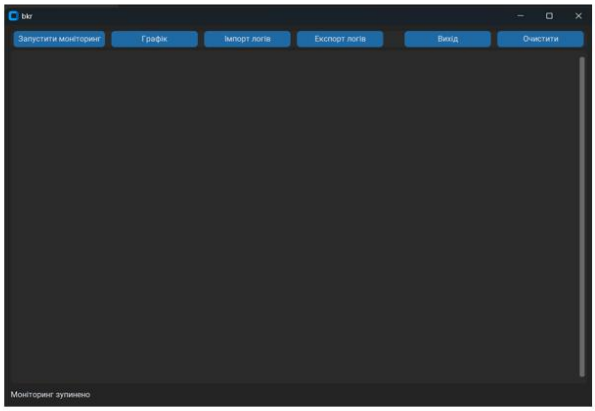
Опис основних сутностей:

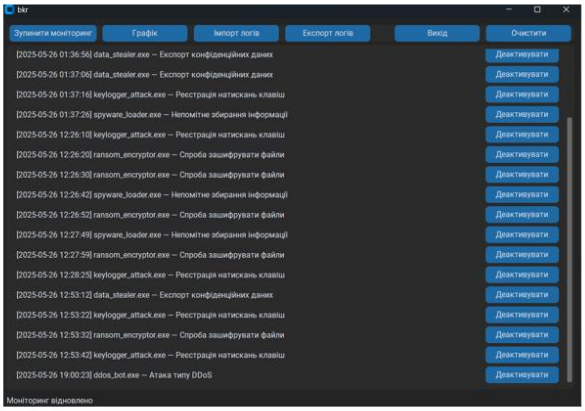
Processes (Процеси):
Зберігає загальну інформацію про кожен запущений процес (назва, шлях, час запуску).

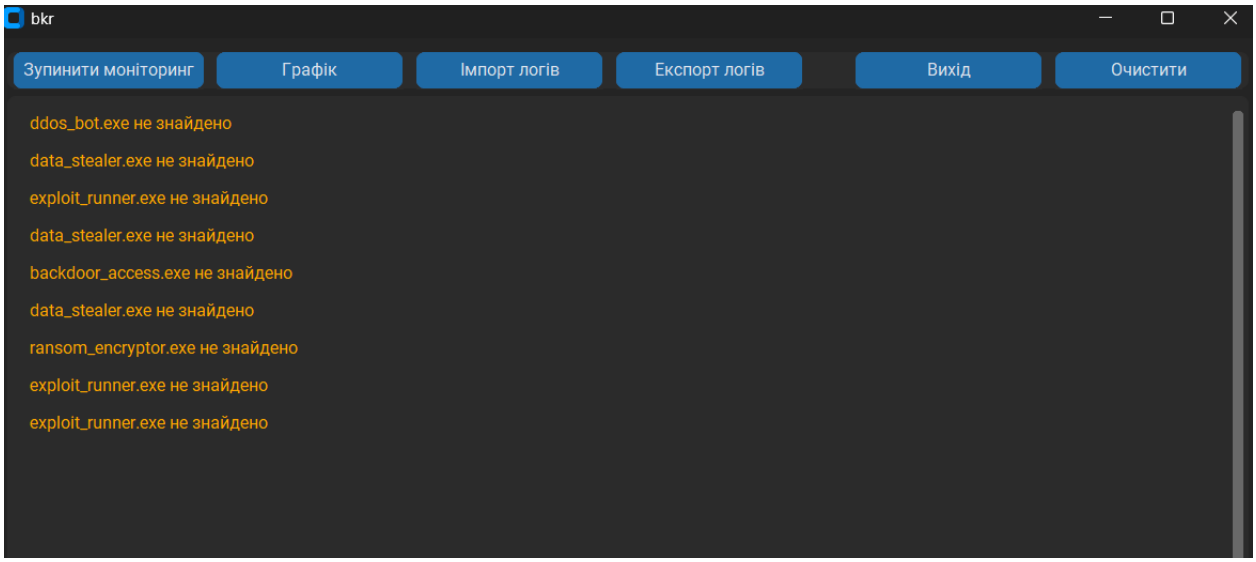
Events (Події):
Містить інформацію про дії, що були виконані процесом (тип події, мітка часу). Зв'язок "один до багатьох" із таблицею **Processes**

AnalysisResults (Результати аналізу):
Зберігає оцінку підозрілості кожного процесу: чи вважається підозрілим **(is suspicious)**, з якої причини, коли аналізувався. Зв'язок "один до одного" із таблицею **Processes.**

Реалізація програмного засобу поведінкового аналізу







Висновки та подальші перспективи розвитку

У процесі виконання роботи було проведено комплексне дослідження методів виявлення шкідливого програмного забезпечення з акцентом на поведінковий аналіз.

Розроблено алгоритм поведінкового аналізу, що враховує ключові ознаки шкідливої активності (зміни у реєстрі, мережеві звернення, модифікації системних файлів) з урахуванням вимог до продуктивності та точності. На основі цього алгоритму створено програмний засіб, який автоматизує процес виявлення шкідливих програм, забезпечуючи збір, обробку та інтерпретацію поведінкових ознак з високим рівнем точності та швидкістю реагування.

Дякую за увагу

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка програми виявлення шкідливого програмного забезпечення на основі поведінкавого аналізу

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Шиян А.А. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	90,12%
Загальна схожість	9,88%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Линдюк А.А.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту**

_____

_____

_____

_____

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)