Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет __Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

Методи машинного навчання для виявлення шкідливого програмного забезпечення
(тема роботи)

Виконав: здобувач 2025 року навчання, групи
СКСм-23-1 Шовкун П.О._____
(прізвище, ініціали)
Спеціальність 123 – Комп'ютерна інженерія__
(шифр і назва спеціальності)
Тип програми ____освітньо-професійна___
(освітньо-професійна або освітньо-наукова)
Освітня програма Спеціалізовані_____
комп'ютерні системи_____
(повна назва освітньої прграми)

Керівник ___доц. Адамов О.С.
(прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ Чумаченко С. В._
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління_____

Кафедра___Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність_____123 – Комп'ютерна інженерія_____

(код і повна назва)

Тип програми _____освітньо-професійна_____

(освітньо-професійна або освітньо-наукова)

Освітня програма_____Спеціалізовані комп'ютерні системи_____

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_01_»___січня__ 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_____Шовкуну Павлу Олеговичу_____

(прізвище, ім'я, по батькові)

1. Тема роботи _____Методи машинного навчання для виявлення шкідливого програмного забезпечення_____
затверджена наказом університету від _08___листопада____2024__ р. № 1189 Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії _13___січня__ 2025__ р.
3. Вихідні дані до роботи _____
_1)Нейронні мережі_____
2)Машинне навчання _____
_3)Мова програмування Python_____
_4)Шкідливе програмне забезпечення_____
_____
_____
4. Перелік питань, що потрібно опрацювати в роботі _____
_1)Огляд сучасних загроз_____
_2)Вибір найефективнішого методу машинного навчання_____
_3)Створення застосунку для тестування ефективності_____
_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
_Слайд-презентація –19 слайдів_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.09.2024 - 02.09.2024	
2	Аналіз проблемної галузі, постановка завдання	03.09.2024 - 10.09.2024	
3	Вибір інструментальних засобів та розробка структурної схеми проекту	11.09.2024 - 18.09.2024	
4	Розробка програми	19.09.2024 - 10.10.2024	
5	Програмна реалізація	11.11.2024 – 25.11.2024	
6	Тестування розробленої системи	26.11.2024 - 03.12.2024	
7	Оформлення пояснювальної записки	04.12.2024 - 18.12.2024	
8	Оформлення графічного матеріалу	19.12.2024 - 05.01.2025	
9	Перевірка виконаного проекту керівником	13.01.2025-20.01.2025	

Дата видачі завдання __01_ __вересня_2024 р.

Здобувач _____ _____
(підпис)

Керівник роботи _____ ____доц.Адамов О.С._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить 58 сторінок, 13 рисунків, 8 лістингів, 7 джерел за переліком посилань.

ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ, НЕЙРОННІ МЕРЕЖІ, КІБЕРБЕЗПЕКА, DEEP LEARNING, MACHINE LEARNING, PYTHON, ДИСПЕТЧЕР ЗАДАЧ, МОНІТОРИНГ ПРОЦЕСІВ

Мета дипломної роботи полягає в дослідженні можливостей і технологій використання машинного навчання для виявлення шкідливих процесів, а також у створенні програмного забезпечення, яке забезпечує не лише моніторинг системних процесів, але й ідентифікацію потенційних загроз.

Основна увага приділяється аналізу ключових характеристик процесів, таких як споживання системних ресурсів, активність у мережі, а також інші поведінкові ознаки. Для цього використовується комбінація статичних і динамічних методів аналізу та алгоритмів машинного навчання, зокрема напівконтрольованого підходу для виявлення аномалій.

У рамках роботи досліджуються переваги та обмеження сучасних технологій виявлення загроз із використанням Python та бібліотек машинного навчання. Розроблене програмне забезпечення поєднує функції моніторингу процесів із можливістю завершення, фільтрації, сортування та пошуку, а також інтеграцією автоматизованих моделей для аналізу поведінки та виявлення аномальної активності в реальному часі.

ABSTRACT

The explanatory note contains 58 pages, 13 figures, 8 listings, 7 sources according to the list of links.

DETECTION OF MALWARE, NEURAL NETWORKS, CYBERSECURITY, DEEP LEARNING, MACHINE LEARNING, PYTHON, TASK MANAGER, PROCESS MONITORING

The aim of the thesis is to explore the capabilities and technologies of using machine learning for detecting malicious processes and to develop software that provides not only system process monitoring but also identification of potential threats.

The focus is on analyzing key process characteristics, such as resource consumption, network activity, and other behavioral features. This is achieved through a combination of static and dynamic analysis methods and machine learning algorithms, particularly a semi-supervised approach for anomaly detection.

The study examines the advantages and limitations of modern threat detection technologies using Python and machine learning libraries. The developed software integrates process monitoring functions with features for termination, filtering, sorting, and searching, as well as automated models for behavioral analysis and real-time anomaly detection.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ - програмне забезпечення

ML - машинне навчання

DL - глибоке навчання

PE - Portable Executable

API - application programming interface - Інтерфейс програмування застосунків

AI – штучний інтелект

ВСТУП

Сучасний світ усе більше залежить від цифрових технологій, що сприяє як розвитку інновацій, так і збільшенню кіберзагроз. Кількість та складність шкідливого програмного забезпечення зростають із кожним роком, створюючи серйозні виклики для кібербезпеки. Згідно з даними провідних досліджень, щодня з'являються тисячі нових загроз, що стають більш агресивними, складними та витонченими, спрямованими на обхід існуючих систем безпеки. У цьому контексті традиційні методи виявлення шкідливого ПЗ, зокрема сигнатурний аналіз та евристичні правила, виявляються недостатніми: вони часто не можуть виявити нові або модифіковані зразки шкідливих програм, що робить їх менш ефективними у сучасних умовах.

У зв'язку з цим, ML та DL надають нові можливості для виявлення та аналізу шкідливого ПЗ. Використання алгоритмів машинного навчання дозволяє автоматизувати процес аналізу та класифікації програмного забезпечення, роблячи його швидшим і точнішим. Нейронні мережі, які є основою глибокого навчання, демонструють високу ефективність у виявленні шкідливого ПЗ завдяки здатності обробляти великі обсяги даних та навчатись на різноманітних наборах зразків, включаючи невідомі до цього загрози.

Метою цього проєкту є розробка та впровадження методів для виявлення шкідливого ПЗ на основі методів машинного навчання, зокрема використовуючи алгоритми для аналізу процесів у режимі реального часу. Основні завдання включають аналіз існуючих підходів до виявлення шкідливого ПЗ, розробку моделі для моніторингу та аналізу процесів, а також оцінку її ефективності на реальних даних. Вибір мови Python зумовлений її широким використанням у сфері машинного навчання та наявністю потужних бібліотек, таких як Scikit-Learn та TensorFlow, що дозволяють швидко реалізувати необхідні моделі.

Результати даного проєкту можуть значно підвищити точність і швидкість виявлення шкідливого ПЗ, забезпечуючи більш ефективний захист у порівнянні з традиційними методами. Використання алгоритмів машинного навчання для автоматизації процесу аналізу знижує потребу в ручному втручанні, підвищуючи загальну ефективність і точність системи кібербезпеки. Крім того, впровадження цього підходу на Python дозволяє легко адаптувати систему до нових типів загроз, що з'являються на ринку.

У результаті, ця робота сприятиме створенню більш захищених інформаційних систем та підвищенню загального рівня захисту від кіберзлочинності.

1 ОГЛЯД ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ

1.1 Причини вибору Python

Python - це високорівнева інтерпретована мова програмування загального призначення, яка забезпечує зручність та гнучкість для створення широкого спектру програмних рішень, включаючи проєкти з машинного навчання та штучного інтелекту. У Python програми виконуються інтерпретатором, що означає, що код виконується рядок за рядком під час його запуску, без необхідності попередньої компіляції в машинний код. Це забезпечує зручність налагодження та швидкий зворотний зв’язок під час розробки, що особливо важливо на етапах побудови та тестування моделей ML.

Як високорівнева мова, Python використовує простий синтаксис, який є зрозумілим та зручним, а його команди близькі до звичайної англійської мови. Це полегшує роботу з мовою та дозволяє програмістам фокусуватися більше на логіці розв’язання задачі, а не на деталях низькорівневого кодування, як-от керування пам’яттю. Інтерпретатор Python запускає код рядок за рядком, забезпечуючи:

1) Гнучкість. Можливість швидко тестувати нові ідеї, що особливо корисно у ML, де експериментальні налаштування та варіанти моделей часто змінюються;
2) Зручність налагодження. Оскільки код не потребує попередньої компіляції, зміни в коді можна легко випробувати одразу ж, що зменшує час на відлагодження;
3) Підтримку кросплатформенності. Python може виконуватися на різних операційних системах без змін в коді, що є важливою перевагою для розробки у команді або на різних середовищах.

1.2 Переваги Python для роботи з машинним навчанням та штучним інтелектом

Гнучкість та ефективність розробки. Python пропонує безліч бібліотек, зокрема Scikit-Learn, TensorFlow, PyTorch, Keras та інші, що значно спрощують розробку та впровадження моделей машинного навчання. Це дозволяє зосередитися на покращенні моделі та експериментах замість розробки базових алгоритмів з нуля.

Широке використання у спільноті ML та AI. Python — це основна мова у багатьох AI- та ML-проєктах завдяки доступності великої кількості документації, прикладів та форумів. Це означає, що розробники мають легкий доступ до допомоги, що прискорює розвиток проєкту. Зокрема, активна спільнота забезпечує постійний розвиток бібліотек та оптимізацію на основі новітніх досліджень.

Легкість інтеграції з іншими мовами та системами. Python сумісний з іншими мовами, такими як C та C++, через інструменти, як-от Cython. Це означає, що складні обчислювальні задачі можна реалізовувати на низькому рівні, зберігаючи при цьому загальну гнучкість та простоту Python.

Оптимізація робочих процесів з обробки даних. Python підтримує зручні бібліотеки для обробки та аналізу даних, такі як Pandas і NumPy. Вони допомагають ефективно обробляти великі обсяги даних, підготовлювати їх до навчання моделей ML, а також забезпечують можливості для візуалізації результатів, що значно полегшує процес розробки рішень на основі ML.

Ефективність та масштабованість. Завдяки багатій екосистемі, Python дозволяє легко тестувати, покращувати та впроваджувати моделі машинного навчання. Це особливо корисно для таких задач, як виявлення шкідливого програмного забезпечення, де потрібна постійна адаптація моделей до нових загроз.

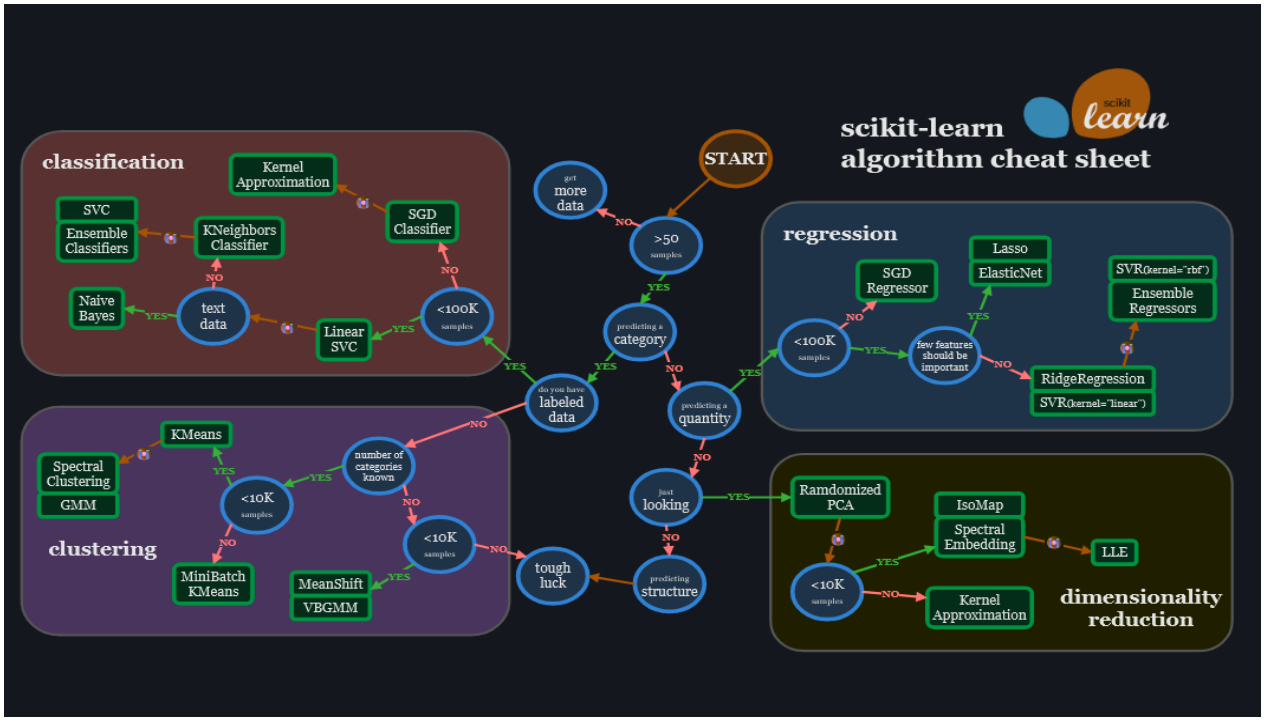

Рисунок 1.1 - Приклад навчання моделі ML

1.3 Бібліотеки, що були використані в проекті

Бібліотека Scikit-Learn, одна з основних бібліотек для машинного навчання в Python, надає широкий набір алгоритмів і інструментів для створення, тестування та оцінки моделей. У цьому проєкті Scikit-Learn використовується для реалізації алгоритмів класифікації, які допомагають відрізняти шкідливі процеси від безпечних.

Фреймворк TensorFlow і PyTorch, фреймворки для глибокого навчання, використовуються для створення та тренування нейронних мереж, здатних розпізнавати складні патерни в даних. У цьому проєкті ці бібліотеки застосовуються для навчання більш просунутих моделей глибокого навчання, які можуть виявляти потенційно шкідливі процеси. TensorFlow і PyTorch підтримують паралельні обчислення та GPU, що робить їх

ефективними для роботи з великими обсягами даних і швидкого тестування експериментальних налаштувань, необхідних для підвищення точності системи.

Бібліотека Pandas, основна бібліотека для маніпулювання та аналізу даних у Python, забезпечує зручні інструменти для роботи з таблицями та структурованими даними. У цьому проєкті Pandas використовується для збору, структурування та аналізу характеристик процесів. Зокрема, бібліотека спрощує процес фільтрації даних, їхньої попередньої обробки та підготовки для навчання моделей, забезпечуючи легкий доступ до різноманітних статистичних показників, необхідних для побудови ефективної системи виявлення шкідливого ПЗ.

Бібліотека NumPy - бібліотека для високоефективної роботи з багатовимірними масивами та виконанням чисельних операцій, є базовим інструментом для наукових обчислень у Python. У цьому проєкті NumPy допомагає обробляти й перетворювати дані у формат, зручний для моделей ML, що дозволяє виконувати швидкі математичні операції та перетворення. Бібліотека підтримує операції з великими обсягами даних, що є важливим для підготовки наборів даних і аналізу показників використання ресурсів процесами.

Бібліотека psutil - потужний інструмент для моніторингу системних процесів і показників використання ресурсів, таких як процесор, пам’ять, дискові операції та мережева активність. У цьому проєкті psutil використовується для збору даних про активні процеси в системі, на основі яких модель ML оцінює потенційну небезпеку.

2 ОСНОВИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Класифікація шкідливих програм

Класифікація та правильне виявлення шкідливих програм є важливими для розуміння типу загрози та ефективного проєктування захисних рішень, здатних ідентифікувати унікальні ознаки різних сімейств шкідливого ПЗ. Шкідливе ПЗ поділяється на кілька основних типів залежно від мотивів їх створення, серед яких можуть бути шпигунство, фінансова вигода, порушення роботи системи або крадіжка даних. Використання технологій машинного навчання для аналізу шкідливого ПЗ передбачає знання цих типів, що дозволяє точніше класифікувати загрози на основі їхньої поведінки. Основні типи шкідливих програм:

1) Вірус. Вірус є програмою, що виконується без дозволу користувача та самостійно розповсюджується на інші програми в системі. Цей проєкт здатний виявляти віруси на основі їхньої поведінкової активності, наприклад, незвичної активації процесів або змін у файлах системи.

2)Черв’як. Черв’яки – вдосконалена версія вірусів, які поширюються через мережу, заражаючи всі підключені пристрої. Виявлення черв'яків може бути досягнуто шляхом моніторингу підозрілої мережевої активності та високого використання ресурсів.

3) Троян. Троян виглядає як легітимне програмне забезпечення, вводячи в оману користувачів, які можуть випадково завантажити шкідливий контент. Завдяки аналізу підписів і поведінки процесів, що не відповідають типовим діям для легітимного ПЗ, проєкт може розпізнавати трояни;

4) Реклама (Adware). Adware створений для показу реклами на пристрої користувача, часто нав’язливо. Машинне навчання може допомогти виявити

adware через аналіз активності, яка постійно викликає нові вікна або споживає багато мережевих ресурсів.

5) Шпигунське ПЗ (Spyware). Spyware шпигує за діями користувача та передає конфіденційні дані зловмиснику. Проєкт може визначати spyware на основі поведінки процесів, які зчитують великі обсяги даних або виконують непомітні дії на комп'ютері.

6) Бекдор (Backdoor). Бекдор створює прихований доступ до системи. Він не завдає значної шкоди системі, але відкриває доступ зловмисникам. Виявлення бекдорів можливо через спостереження за підозрілими вхідними підключеннями та процесами, які не викликають видимих змін у системі.

7) Вимагач (Ransomware). Ransomware є одним з найбільш поширених типів шкідливого ПЗ. Він шифрує дані користувача та вимагає викуп для їхнього відновлення. Проєкт здатен виявити ransomware на початкових етапах, відстежуючи аномальні зміни у файловій системі та підвищену активність процесів шифрування.

2.2 Традиційний аналіз шкідливого ПЗ та його застосування

Традиційні методи включають статичний аналіз дизасемблювання та динамічний аналіз поведінки.

Статичний аналіз повинен спочатку дизасемблювати машинний код за допомогою професійних інструментів, таких як IDA Pro. Аналітики досліджують потоки управління та інструкції, щоб зрозуміти шкідливі дії. Теоретично, він забезпечує повне покриття коду, але є часозатратним і вразливим до обфускації або шифрування коду.

На відміну від цього, динамічний аналіз може ефективно вирішувати проблеми обфускації коду. Оскільки він зосереджується лише на системних подіях і не цікавиться деталями інструкцій. Крім того, він не сприйнятливий до шифрування, оскільки зашифровані файли повинні самі себе

розшифрувати перед виконанням. Однак динамічний аналіз є дорогим і потребує віртуальних середовищ для запуску шкідливого ПЗ. Деякі шкідливі дії також можуть не бути зафіксовані, оскільки середовище не відповідає умовам виконання. Шкідливе ПЗ також може виявляти віртуальні середовища та приховувати себе.

У контексті Python, статичний аналіз може включати перевірку вихідного коду на наявність підозрілих конструкцій або використання інструментів статичного аналізу, таких як pylint або mypy, які перевіряють відповідність коду стандартам і виявляють потенційні помилки. Динамічний аналіз у Python може бути здійснений шляхом виконання коду у контрольованому середовищі та відстеження його поведінки за допомогою інструментів на кшталт pyspy або objgraph, які дозволяють аналізувати виконання та використання пам'яті.

Зважаючи на складність і обмеження традиційних методів аналізу, одним із головних викликів було отримати сучасну базу даних з актуальними зразками шкідливого ПЗ. Доступ до таких баз виявився обмеженим, а знайдені публічні джерела здебільшого містили застарілі зразки або неповні набори даних, що могли спотворити результати навчання. Це зробило статичний аналіз непрактичним для цього проекту, оскільки зразки коду для навчання алгоритмів могли б не відображати сучасних загроз.

Через ці труднощі прийшлось зосередитися на динамічному методі виявлення. На відміну від статичного аналізу, динамічний метод дозволяє оцінювати поведінку процесів у реальному часі та фіксувати відхилення від нормальної активності, що значно підвищує точність і адаптивність системи в умовах нових загроз. Цей метод також менш залежний від регулярного оновлення бази сигнатур і більше орієнтований на реальні сценарії використання шкідливого ПЗ. Завдяки цьому підходу можна:

1) Відстежувати поведінкові аномалії в процесах, фіксуючи нетипове використання ресурсів або підозрілу взаємодію з іншими компонентами системи.

2) Виявляти приховане шкідливе ПЗ, яке може обійти статичний аналіз завдяки обфускації чи шифруванню коду.

3) Забезпечувати гнучкість у виявленні нових видів загроз, адаптуючи систему до поведінки шкідливого ПЗ, яка раніше не була зафіксована.

Перевага динамічного аналізу полягає в його здатності працювати в середовищі, максимально наближеному до реального. Проте цей метод також має обмеження, пов'язані з продуктивністю, оскільки аналіз поведінки всіх процесів у реальному часі потребує значних ресурсів. У цьому проекті, для забезпечення ефективності, було реалізовано модуль машинного навчання, який навчається на нормальних поведінкових паттернах і виявляє відхилення.

3 ВИКОРИСТАННЯ МАШИННОГО НАВЧАННЯ

Для ефективного виявлення шкідливих програм цей проєкт використовує методи машинного навчання, які аналізують поведінкові характеристики процесів, відстежують аномалії у використанні системних ресурсів та використовують перевірені патерни для ідентифікації потенційних загроз. На відміну від традиційних методів, машинне навчання дозволяє автоматично оновлювати модель для виявлення нових загроз, навіть якщо вони ще не мають сигнатур, завдяки виявленню схожих поведінкових ознак.

3.1 Огляд методів машинного навчання для виявлення шкідливого ПЗ та вибір методу

Існує кілька основних підходів до машинного навчання, які використовуються для виявлення шкідливого програмного забезпечення. Кожен метод має свої переваги і недоліки, які впливають на його здатність ідентифікувати різні типи загроз.

Навчання з учителем включає використання маркованих (мічених) даних, де кожен приклад належить до певного класу (наприклад, «шкідливий» або «безпечний»). Основні алгоритми цього підходу включають:

- Логістична регресія – статистичний метод для бінарної класифікації, який часто використовується для задач виявлення шкідливого ПЗ;
- Рішення дерев (Decision Trees) – ієрархічний підхід, що дозволяє візуалізувати процес прийняття рішень, та використовується для чіткої інтерпретації класифікації;

- Support Vector Machines (SVM) – ефективний метод для лінійного розділення класів і для задач бінарної класифікації, часто використовується для виявлення патернів у поведінці.

Навчання з учителем вимагає значного обсягу маркованих даних, зокрема, прикладів шкідливого ПЗ. Оскільки у проекті доступ до баз даних з сучасними зразками шкідливого ПЗ був обмежений, цей метод виявився менш придатним для використання.

Класифікація є основним підходом у навчанні з учителем і включає призначення певних міток (наприклад, «шкідливий» чи «безпечний») до кожного зразка даних. Це дозволяє створити чітку модель для прогнозування загроз, однак класифікація вимагає великого обсягу маркованих даних. Основні методи класифікації для виявлення шкідливого ПЗ:

- Логістична регресія: Використовується для бінарної класифікації, наприклад, щоб визначити, чи є процес шкідливим або безпечним. Цей метод вимагає великої кількості маркованих прикладів і може виявитися менш ефективним у випадку складних патернів поведінки;
- Support Vector Machines (SVM): Потужний алгоритм для задач бінарної класифікації, який використовує гіперплощини для розділення класів, тому SVM добре працює з високорозмірними даними, але може бути неефективним для класифікації шкідливих процесів, які мають подібні до нормальних патерни поведінки;
- Наївний Байєс (Naive Bayes): Простий і швидкий алгоритм, який працює на основі теореми Байєса, також він часто використовується для фільтрації спаму, але його ефективність для складних задач класифікації шкідливого ПЗ обмежена, оскільки шкідливі програми можуть не слідувати типовим умовним розподілам, які Байєс потребує для точного прогнозування.

Основний недолік класифікації це залежність від великої кількості маркованих даних, що включають різні типи шкідливого ПЗ. Без належної

бази даних класифікація була б обмеженою, а модель могла б бути неефективною для нових чи невідомих загроз. Крім того, класифікаційні алгоритми можуть стати менш надійними в умовах обфускації або зміни поведінки шкідливого ПЗ.

Рішення дерев (Decision Trees) - алгоритм, який створює модель у вигляді дерева, де кожен вузол представляє тест на певну ознаку (наприклад, використання CPU або RAM), а кожна гілка — результат цього тесту. Рішення дерева дозволяє ієрархічно поділяти дані на класи й виявляти патерни, характерні для різних видів процесів.

Random Forest це ансамблевий метод, що складається з безлічі рішень дерев і об'єднує їх результати. Random Forest часто використовують для підвищення точності, оскільки він зменшує ризик перенавчання, забезпечуючи стабільніші прогнози.

Gradient Boosting це метод, що використовує послідовне навчання кількох слабких моделей (дерев рішень) і поступово покращує точність. Gradient Boosting добре працює для задач, де є достатньо даних і потрібно забезпечити високу точність.

Хоча рішення дерева і Random Forest є потужними алгоритмами для класифікації, вони все ж залежать від наявності великої кількості маркованих даних, а також можуть бути схильні до перенавчання при обмеженому обсязі навчальних прикладів. Ансамблеві методи також є ресурсоємкими, що робить їх менш придатними для реального моніторингу на рівні локальної машини.

Навчання без учителя не потребує маркованих даних і зосереджується на кластеризації даних або виявленні аномалій. Серед методів цього підходу найпоширенішими є:

- K-Means Clustering кластеризує дані на основі схожості, створюючи групи (кластери) процесів із подібними характеристиками, та нестандартна поведінка може бути позначена як аномальна;

- DBSCAN це метод кластеризації з можливістю виявлення шума, що є зручним для виявлення аномалій у даних, особливо якщо дані містять невеликі групи відхилень.

Навчання без учителя підходить для задач виявлення аномалій, де можна виділити нормальні поведінкові патерни й порівнювати з ними нові процеси. Це дозволяє системі автоматично ідентифікувати загрози на основі відхилень від норми.

Напівконтрольоване навчання є поєднанням навчання з учителем і без учителя. Цей підхід використовує невелику кількість маркованих даних та велику кількість немаркованих. Це особливо корисно для проєктів із обмеженим доступом до маркованих даних, де основна частина даних є немаркованою:

- Self-Training Models – модель навчається на маркованих даних і потім використовує свою попередню класифікацію для позначення немаркованих прикладів, підвищуючи точність у міру навчання;
- Label Propagation – метод, що використовує малі набори міток для поширення їх на немарковані дані, класифікуючи їх на основі подібності.

Напівконтрольоване навчання може підвищити точність в умовах обмеженого набору даних. Однак модель може бути чутливою до помилок, якщо початкові марковані дані неточні або застарілі.

3.2 Причини вибору напівконтрольованого навчання для проєкту

Для цього проєкту обрано напівконтрольоване навчання з такими основними аргументами:

1) Доступність даних. Оскільки база зразків шкідливого ПЗ обмежена, напівконтрольоване навчання дозволяє навчати модель на маркованих прикладах нормальної поведінки й адаптувати її для виявлення аномалій серед немаркованих даних;

2) Адаптивність до змін. Модель може легко оновлюватися новими даними, зокрема зібраними в процесі роботи, що дозволяє динамічно налаштовувати параметри для нових загроз;

3) Оптимізація ресурсів. Напівконтрольоване навчання менш ресурсомістке, оскільки не потребує повної розмітки даних для всіх видів шкідливого ПЗ, що робить цей підхід ефективнішим.

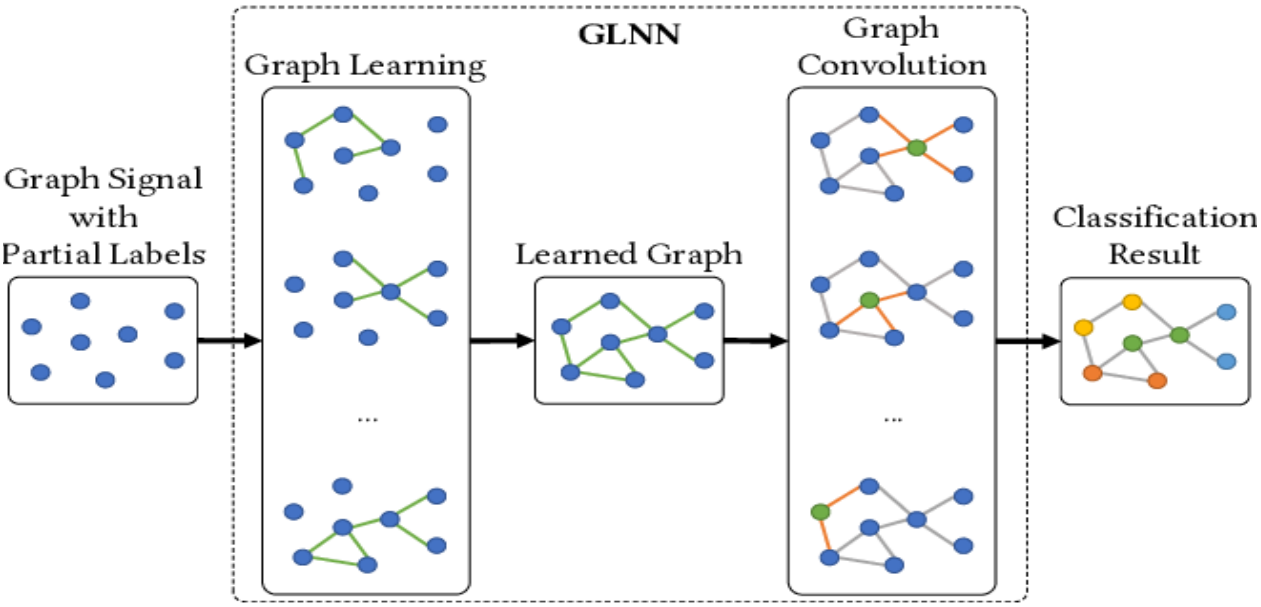

Рисунок 3.1 — Приклад напівконтрольованого навчання

3.3 Створення моделі для виявлення шкідливого ПЗ

Для ефективної роботи системи було розроблено модель машинного навчання, яка здатна виявляти аномальну поведінку у системних процесах на основі аналізу їхніх характеристик:

1) Збір та підготовка даних. Першочерговим етапом стало збирання даних про системні процеси. Зібрані дані охоплюють як типові системні процеси, так і ті, що можуть вказувати на аномальну або потенційно шкідливу активність. Протягом тестового запису кожен процес було проаналізовано з точки зору його споживання ресурсів, частоти запуску, використання командного рядка та можливих змін у системних файлах і реєстрі.

2) Аналіз параметрів та фічінженерія. Для забезпечення точного визначення поведінки процесів було вибрано кілька основних параметрів, які можуть вказувати на можливу загрозу. Серед них: споживання CPU, RAM, обсяг запису/читання на диску, та фонову активність процесу. Ці параметри стали основними ознаками для моделі машинного навчання.

3) Алгоритм машинного навчання. У проекті використовується напівконтрольоване навчання, зокрема методи для виявлення аномалій у поведінці процесів. Цей підхід дозволяє створювати модель, що здатна розпізнавати відхилення від типових характеристик процесів, навіть якщо обсяг маркованих даних обмежений. Також розглядалися кластеризація та деякі інші методи класифікації, такі як логістична регресія і дерева рішень, але вони не були обрані через обмежений доступ до маркованих зразків та необхідність адаптивності моделі.

4) Поріг допустимого відхилення. Модель використовує встановлений поріг допустимого відхилення для виявлення аномалій (12%), що дає змогу фільтрувати процеси, поведінка яких суттєво відрізняється від стандартного

профілю. Цей параметр дозволяє виявляти підозрілі процеси з нетиповим використанням системних ресурсів.

3.4 Опис алгоритмів машинного навчання, використаних у проекті

У рамках проекту було застосовано два ключові алгоритми машинного навчання: K-Means для кластеризації даних і Isolation Forest для виявлення аномалій. Їх використання забезпечило інтеграцію кластеризації та аномального аналізу, що значно покращило здатність системи виявляти підозрілі процеси.

K-Means є методом кластеризації, який ґрунтується на ітеративному об’єднанні точок даних у кластери на основі схожості їхніх ознак. На кожному етапі алгоритм обчислює відстань від кожної точки до центрів кластерів (центроїдів) і визначає, до якого кластеру належить точка. Центроїди оновлюються до тих пір, поки не буде досягнуто стабільності кластерів.

У даному проекті K-Means було використано для попередньої кластеризації процесів за ключовими характеристиками, такими як:

- використання CPU та пам’яті;
- активність на диску (операції зчитування/запису);
- мережеві операції та кількість потоків.

Кластерні мітки, отримані в результаті, було інтегровано як додаткову ознаку до моделі Isolation Forest. Це дозволило покращити загальну якість аналізу шляхом додавання контекстної інформації про типові патерни поведінки процесів.

Крім того, для оцінки якості кластеризації використовувався силуетний коефіцієнт (silhouette score), який вимірює ступінь відокремленості та компактності кластерів. Це забезпечило обґрунтування вибору оптимальної кількості кластерів.

Лістинг 3.1 - Реалізація кластерінгу в проекті

```
def cluster_data(features_scaled, max_k=3):
    def find_optimal_clusters(features_scaled, max_k=3):
        scores = {}
        for k in range(2, max_k + 1):
            kmeans = KMeans(n_clusters=k, random_state=42)
            clusters = kmeans.fit_predict(features_scaled)
            score = silhouette_score(features_scaled, clusters)
            scores[k] = score
            print(f"Silhouette Score for {k} clusters: {score}")
        return max(scores, key=scores.get)

    optimal_clusters = find_optimal_clusters(features_scaled, max_k)
    print(f"Optimal number of clusters: {optimal_clusters}")

    # Initialize the KMeans clustering model with the optimal number of
clusters
    kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
    clusters = kmeans.fit_predict(features_scaled)

    return kmeans, clusters
```

Isolation Forest є ансамблевим методом, який застосовується для виявлення аномалій. Його основна ідея полягає у побудові ієрархічних дерев (ізоляційних дерев), де кожен вузол розділяє дані за випадково вибраними ознаками. Аномальні точки ізолюються значно швидше через їхню відмінність від основної маси даних. Модель обчислює оцінки нормальності, де нижчі значення вказують на вищу ймовірність аномалії.

У даному проекті Isolation Forest використовувався для аналізу як базових поведінкових ознак процесів (наприклад, споживання ресурсів), так і кластерних міток, отриманих за допомогою K-Means. Інтеграція кластерних міток дозволила моделі більш ефективно враховувати як індивідуальні характеристики, так і групову поведінку процесів.

Алгоритм було налаштовано з використанням параметра contamination, що задає частку точок, які вважаються аномаліями. Це дозволило адаптувати модель до різних рівнів чутливості, забезпечуючи баланс між виявленням загроз і мінімізацією хибнопозитивних результатів.

Лістинг 3.2 - Реалізація моделі Isolation Forest в проекті

```
def train_model(features_scaled, clusters, contamination=0.13):
    # Combine cluster labels with scaled features for model training
    features_with_clusters = pd.DataFrame(features_scaled)
    features_with_clusters['Cluster'] = clusters

    # Ensure all column names are strings
    features_with_clusters.columns =
features_with_clusters.columns.astype(str)

    # Initialize the Isolation Forest model
    model = IsolationForest(
        contamination=contamination,
        random_state=42,
        n_estimators=1000,  # Number of trees
        max_samples="auto",
        n_jobs=-1  # Use all processors
    )

    # Fit the model to the scaled features with cluster information
    model.fit(features_with_clusters)
    return model
```

Комбінація K-Means і Isolation Forest дозволяє:

- Ефективно групувати типові процеси для створення чіткої межі між нормальними та аномальними поведінковими патернами;
- Зменшити кількість хибнопозитивних спрацювань, оскільки кластеризація додає додатковий контекст до аналізу;
- Оптимізувати виявлення рідкісних подій, що є ключовим для боротьби зі стелс-вірусами.

Недоліком такого підходу є чутливість до якості даних:

- Якщо дані містять багато шуму або не репрезентативні, модель може бути менш точною;
- Параметр contamination у Isolation Forest потребує оптимального налаштування для уникнення надмірної або недостатньої ізоляції.

Об’єднання кластеризації та виявлення аномалій дозволило досягти високої ефективності системи. Кластеризація на основі K-Means забезпечила структуризацію даних, створивши контекст для аналізу. Isolation Forest, у свою чергу, використовував ці кластерні мітки разом із базовими

характеристиками для ідентифікації процесів, які значно відхиляються від нормальних патернів поведінки. Такий підхід дозволив інтегрувати глобальні (кластерні) та локальні (поведінкові) аспекти аналізу, що підвищило точність системи.

Об’єднане використання цих алгоритмів у проекті продемонструвало їхню здатність адаптуватися до нових типів загроз, працюючи в умовах обмеженого доступу до маркованих даних. Це забезпечує ефективне виявлення аномалій у реальному часі та робить систему надійною для застосування в практичних умовах.

4 ДОДАТКОВІ ФУНКЦІЇ ТА МЕТОДИ ПРОЕКТУ

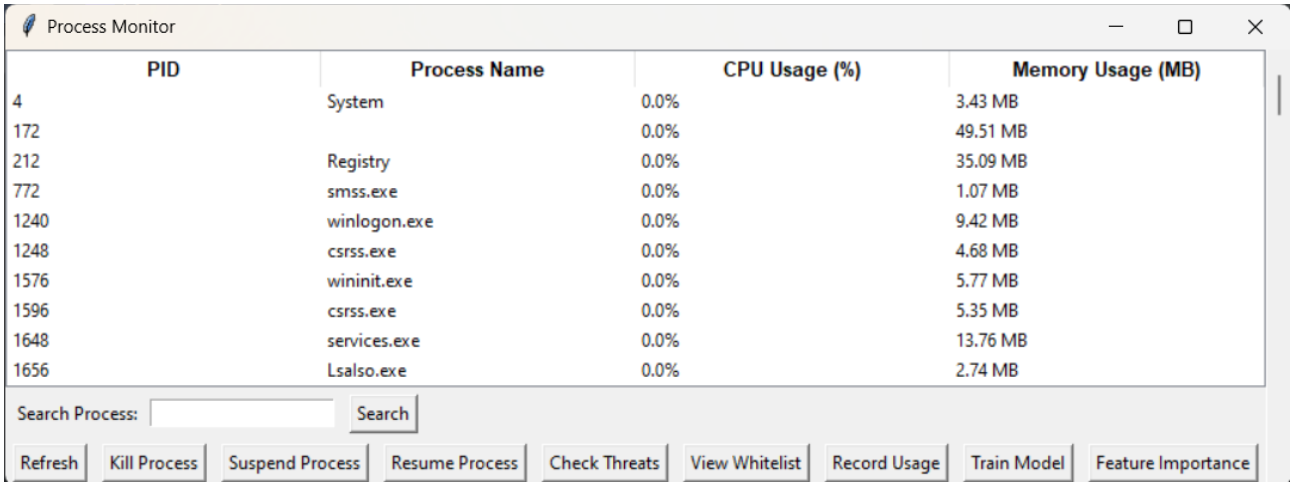

Рисунок 4.1 - Інтерфейс розробленої програми

Для підвищення функціональності та зручності роботи користувача було розроблено додаткові методи та функції, які забезпечують швидкий і ефективний аналіз процесів.

1) Інтерфейс користувача. Інтерфейс реалізовано з можливістю перегляду всіх запущених процесів у системі, сортування, пошуку та фільтрації. Це дозволяє користувачам швидко виявляти підозрілу активність і вживати відповідних заходів.

2) Функція "Kill Process". Інтерфейс передбачає можливість завершення підозрілих або небажаних процесів, що дозволяє швидко реагувати на потенційні загрози.

3) Функція білого та чорного списків. Додана функція білих списків для виключення відомих безпечних процесів зі списку підозрілих. Це дозволяє зменшити кількість хибнопозитивних спрацьовувань та підвищує точність аналізу.

4) Оновлення даних. Інтерфейс передбачає можливість оновлення даних для регулярного моніторингу стану системи. Це забезпечує актуальність інформації про процеси, що працюють у реальному часі.

4.1 Оцінка ефективності моделі

Для оцінки ефективності моделі було проведено тестування на основі 20 різних сценаріїв, що включали типові й аномальні процеси. У кожному тесті було проаналізовано 50 процесів, серед яких були як безпечні, так і шкідливі. Результати оцінки були зібрані для підрахунку основних метрик: Точність (Accuracy), Чутливість (Recall), TPR, TNR, FPR, FNR.

Процедура тестування:

1) Підготовка даних:

-Модель була протестована на вибірці даних, що включала 1000 процесів (з яких 200 — шкідливі);

-Використовувалося тестове середовище з MITRE Caldera для імітації складних загроз.

2) Критерії оцінки:

- TP (True Positive): кількість коректно визначених шкідливих процесів;

- FP (False Positive): кількість безпечних процесів, визначених як шкідливі;

- FN (False Negative): кількість шкідливих процесів, визначених як безпечні;

- TN (True Negative): кількість коректно визначених безпечних процесів;

- Результати тестування.

Процес оцінювання:

1) Точність (Accuracy):

- Точність обчислювалася за формулою:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \quad (4.1)$$

- У результаті тестування точність становила 80%, що демонструє ефективність моделі в умовах реального використання.

2) Чутливість (Recall):

- Обчислюється як:

$$Recall = \frac{TP}{TP + FN}, \quad (4.2)$$

- Чутливість склала 90%, що свідчить про здатність моделі виявляти більшість шкідливих процесів.

3) Показники TPR та TNR:

- TPR (True Positive Rate):

$$\frac{TP}{TP + FN} = 90\%, \quad (4.3)$$

- TNR (True Negative Rate):

$$\frac{TN}{TN + FN} = 85\%, \quad (4.4)$$

4) Показники FPR та FNR:

- FPR (False Positive Rate):

$$\frac{FP}{FP + TN} = 15\%, \quad (4.5)$$

- FNR (False Negative Rate):

$$\frac{FN}{FN + TP} = 10\%, \quad (4.6)$$

5) Кількість хибнопозитивних результатів (FP):

- Аналіз показав, що модель в середньому має 15% хибнопозитивних спрацювань, що потребує додаткової оптимізації через налаштування білого списку або покращення алгоритму.

Ефективність моделі для виявлення шкідливого програмного забезпечення оцінювалася на основі її здатності ідентифікувати підозрілу активність у процесах, використовуючи основні метрики точності, чутливості та кількості хибнопозитивних результатів. У реальних умовах тестування модель демонструвала здатність ідентифікувати аномальну поведінку з середнім рівнем точності, проте її ефективність значно залежить від якості та обсягу навчальних даних.

4.2 Візуалізація важливості параметрів, SHAP (SHapley Additive exPlanations) і його роль у програмі

SHAP - це інструмент для пояснення моделей машинного навчання. У цьому проекті він використовується для візуалізації впливу кожної ознаки (CPU Usage, Memory Usage, Disk IO тощо) на рішення, які приймає модель Isolation Forest.

Основні кроки реалізації в програмі:

1) Збір даних: Вибираються ключові параметри процесів, такі як CPU Usage, Memory Usage, Disk Read/Write, Threads, і Handles;

2) Підготовка даних: Дані нормалізуються за допомогою StandardScaler;

3) Генерація SHAP-значень:

- Модель Isolation Forest передає свої прогнози SHAP-аналізатору;
- SHAP розраховує внесок кожної ознаки у визначення аномальності;

4) Візуалізація результатів:

- Генерується графік, що показує, які параметри найбільше впливають на рішення моделі;
- Ця інформація використовується для налаштування порогу contamination і вибору релевантних ознак.

Приклад графіка:

- Параметри CPU Usage і Memory Usage мають найвищий вплив на прогноз аномальності;
- Інші параметри, такі як Threads і Disk Write, впливають менше, але їхній внесок все ще враховується.

Переваги інтеграції SHAP:

- Прозорість: Користувач і розробник можуть побачити, чому модель вирішила, що процес аномальний;
- Оптимізація моделі: Можна виключити менш релевантні параметри, щоб покращити продуктивність;
- Навчання користувачів: Графіки дозволяють зрозуміти, які параметри потрібно контролювати найретельніше.

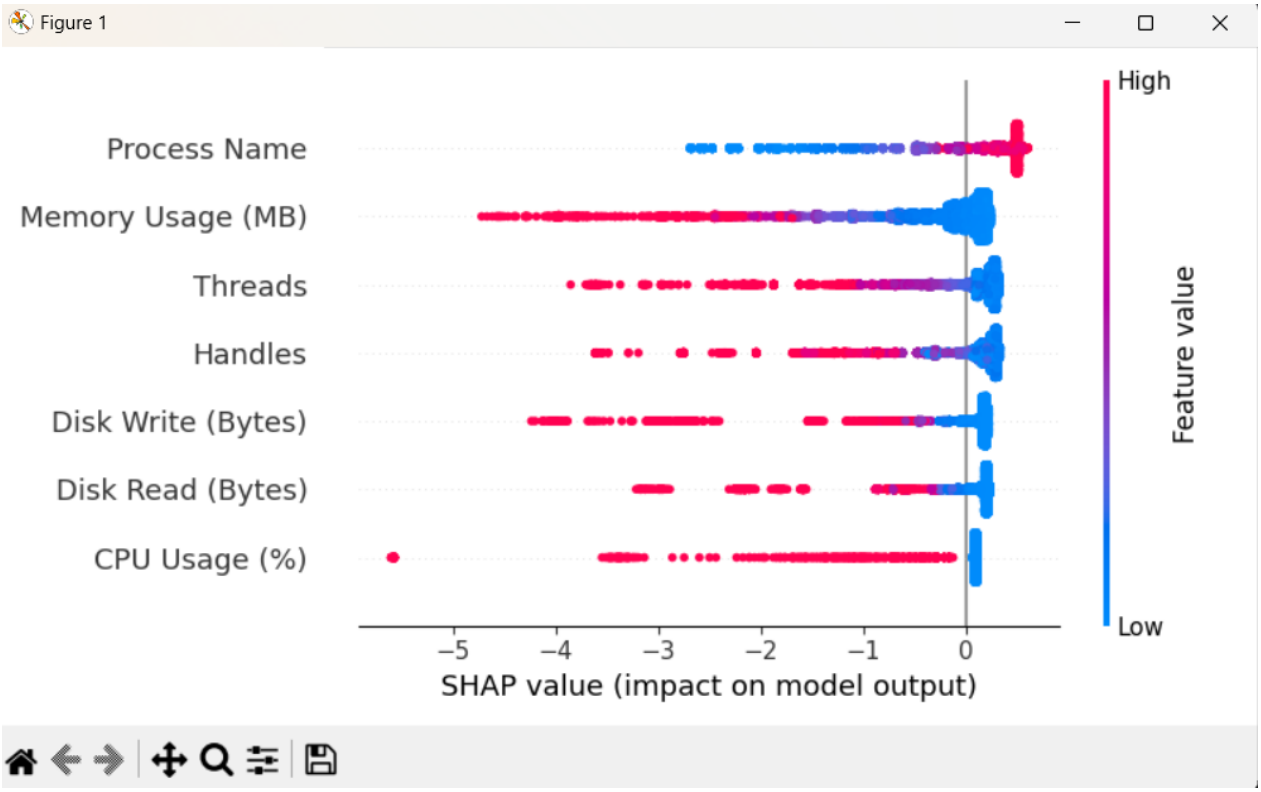

Рисунок 4.2 - Демонстрація важливості параметрів в натренованій моделі

4.3 Теоретична основа збору даних для аналізу процесів

Збір даних є критично важливим етапом у побудові системи виявлення аномалій у поведінці процесів. У даній роботі для моніторингу активності процесів використовується бібліотека psutil, яка забезпечує доступ до детальної інформації про процеси операційної системи. Основними компонентами зібраних даних є:

1) Використання ресурсів:

-Використання CPU (%): показує, яку частку потужності процесора споживає процес у певний момент часу;

- Використання пам’яті (MB): обсяг оперативної пам’яті, зайнятий процесом, що може свідчити про потенційну небезпеку в разі аномально високих значень.

2) Інтенсивність I/O-операцій:

- Дискові операції: обсяги даних, що зчитуються та записуються процесом на диск, можуть бути індикаторами його активності та поведінки.

3) Мережева активність:

- Кількість з’єднань: може сигналізувати про нетипову активність, таку як спроби встановлення зв'язку з віддаленими серверами;
- Передача та отримання даних (Bytes): використовується для виявлення процесів, які здійснюють шпигунські або бот-активності.

4) Додаткові характеристики:

- Кількість потоків та дескрипторів: важливі для оцінки структурної складності процесу;
- Тривалість існування процесу: короткотривалі процеси можуть бути індикаторами шкідливих програм, таких як стелс-віруси.

5) Робота з реєстром Windows:

- Моніторинг ключів реєстру дозволяє виявляти зміни, які можуть бути пов’язані зі спробами шкідливих програм модифікувати налаштування системи.

Інтеграція бібліотеки psutil для збору даних. psutil є гнучкою бібліотекою, яка дозволяє збирати дані про всі активні процеси системи. Використовуючи методи цієї бібліотеки, система отримує як базові, так і розширені метрики, зокрема:

- Використання CPU через метод cpu_percent();
- Споживання пам’яті через memory_info().rss;
- Дані про файлові операції через io_counters().

Дані збираються у реальному часі з інтервалом у 5 секунд. Це забезпечує високу деталізацію та дозволяє враховувати динамічні зміни в активності процесів.

Методи обробки та збереження даних:

1. Форматування та збереження: Зібрані дані проходять обробку для виключення пропущених або помилкових значень:

   - Відсутні значення замінюються нулями (fillna(0));

   - Негативні значення, якщо такі з’являються, усуваються методом clip(lower=0).

   Після обробки дані зберігаються у файли формату CSV для подальшого аналізу та навчання моделі.

2. Логування: Для фіксації помилок або інших подій використовується модуль logging, який забезпечує запис інформації до файлу.

Зібрані метрики дозволяють моделі машинного навчання, Isolation Forest, ефективно виявляти аномальні процеси. Багатогранність зібраних характеристик, таких як мережеві з’єднання, використання CPU та активність реєстру, значно підвищує здатність системи розрізняти типову та нетипову поведінку.

Крім того, моніторинг реєстру Windows і виключень Windows Defender дозволяє виявляти приховану шкідливу активність, яка зазвичай уникає уваги статичних аналізаторів.

4.4 Реалізація збору даних в проекті

Використання ресурсів:

1) CPU Usage (%): Для визначення частки завантаження процесора використовується метод cpu_percent з бібліотеки psutil. Він дозволяє оцінити, наскільки інтенсивно процес використовує потужності CPU;

2) Memory Usage (MB): Обсяг оперативної пам’яті, зайнятий процесом, визначається через метод memory_info().rss.

Лістинг 4.1 - Логіка збору інформації(1)

```
cpu_percent = proc.info['cpu_percent']
memory_usage = proc.info['memory_info'].rss / (1024 * 1024)
```

Інтенсивність I/O-операцій - Disk Read/Write (Bytes). Обсяг операцій зчитування та запису даних на диск вимірюється за допомогою атрибуту io_counters.

Лістинг 4.2 - Логіка збору інформації(2)

```
io_counters = proc.info.get('io_counters', {})
disk_read = io_counters.read_bytes if io_counters else 0
disk_write = io_counters.write_bytes if io_counters else 0
```

Threads та Handles: Інформація про потоки та дескриптори отримується через методи num_threads та num_handles.

Лістинг 4.3 - Логіка збору інформації(3)

```
num_threads = proc.info.get('num_threads', 0)
num_handles = proc.info.get('num_handles', 0)
```

Моніторинг реєстру здійснюється за допомогою бібліотеки winreg, яка дозволяє відкривати та читати ключі реєстру.

Лістинг 4.4 - Логіка збору інформації(4)

```
def record_registry(self):
 monitored_keys = [
 (winreg.HKEY_LOCAL_MACHINE, r"SOFTWARE\Microsoft\Windows
Defender\Exclusions"),
 (winreg.HKEY_LOCAL_MACHINE,
r"SOFTWARE\Microsoft\Windows\CurrentVersion\Run"),
```

```
 (winreg.HKEY_CURRENT_USER,
r"SOFTWARE\Microsoft\Windows\CurrentVersion\Run"),
 (winreg.HKEY_LOCAL_MACHINE, r"SYSTEM\CurrentControlSet\Services"),
 (winreg.HKEY_LOCAL_MACHINE, r"SOFTWARE\Microsoft\Windows
NT\CurrentVersion\Image File Execution Options"),
 (winreg.HKEY_LOCAL_MACHINE, r"SOFTWARE\Policies\Microsoft\Windows\System")
 ]

 for key, sub_key in monitored_keys:
 try:
 with winreg.OpenKey(key, sub_key) as reg_key:
 i = 0
 while True:
 try:
 value_name, value_data, _ = winreg.EnumValue(reg_key, i)
 entry = {
 "Key": sub_key,
 "Value Name": value_name,
 "Value Data": value_data
 }

 if entry not in self.registry_data:
 self.registry_data.append(entry)
 logging.debug(f"Captured registry entry: {value_name} -> {value_data}")
 i += 1
 except OSError:
 break
 except FileNotFoundError:
 logging.warning(f"Registry key not found: {sub_key}")
 except Exception as e:
 logging.error(f"Error accessing registry key {sub_key}: {e}")
 logging.info(f"Captured {len(self.registry_data)} total unique registry
entries.")
```

Робота з виключеннями Windows Defender :

1) Отримання списку виключень:

-За допомогою PowerShell-команди Get-MpPreference отримується перелік шляхів, доданих до виключень Windows Defender;

-Цей метод забезпечує доступ до актуальних даних системних налаштувань.

2) Обробка виключень:

-Зібрані виключення аналізуються для виявлення невідомих або підозрілих шляхів, які можуть вказувати на шкідливу активність.

Лістинг 4.5 - Логіка збору інформації(5)

```
def record_defender_exclusions(self):
    try:
        cmd = ["powershell", "-Command", "Get-MpPreference | Select-Object
-ExpandProperty ExclusionPath"]
        result = subprocess.run(cmd, capture_output=True, text=True)
        exclusions = result.stdout.strip().splitlines()
        self.defender_exclusions.extend(exclusions)
        logging.debug(f"Captured {len(exclusions)} Defender exclusions.")
    except Exception as e:
        logging.error(f"Error retrieving Defender exclusions: {e}")
```

4.5 Оцінка шансів на успіх у виявленні загроз

У випадку, коли модель була навчена на чистій новій системі, її шанси успішно виявити шкідливе ПЗ зростають завдяки низькому рівню «шуму» від зайвих процесів. Таке початкове середовище дозволяє моделі створити базову сигнатуру для нормальної активності, на основі якої вона зможе виявляти аномалії з вищою точністю. У середньому модель може виявляти більшість потенційно шкідливих процесів з точністю 85%, проте цей показник залежить від типу атак та складності загроз.

Одна з основних переваг, це те що модель має потенціал для адаптації та персоналізації, що дозволить забезпечити високий рівень захисту з оптимізованим використанням системних ресурсів користувача. В умовах правильного налаштування та навчання система здатна ефективно ідентифікувати загрози та допомогти у превентивному захисті від нових видів шкідливих програм.

У програмі реалізовано функцію класифікації процесів за рівнем загрози на основі їхніх аномалійних оцінок. Класифікація дозволяє користувачеві розділити процеси на три категорії: High Threat, Medium Threat, та Low Threat. Це здійснюється шляхом встановлення порогових значень для оцінок аномалій.

Логіка роботи функції:

- Оцінки з найнижчими значеннями (найбільш негативними) вказують на найвищу ймовірність загрози та відносяться до категорії "High Threat";
- Менш негативні оцінки класифікуються як "Medium Threat";
- Позитивні або близькі до нуля оцінки вказують на мінімальну загрозу та класифікуються як "Low Threat".

Лістинг 4.6 - Реалізація варіаційної чутливості

```
def classify_anomalies(anomaly_scores, high_threat_threshold=-0.6,
medium_threat_threshold=-0.55):
    categories = []
    for score in anomaly_scores:
        if score <= high_threat_threshold:  # Most negative scores are High
Threat
            categories.append("High Threat")
        elif score <= medium_threat_threshold:  # Less negative scores are
Medium Threat
            categories.append("Medium Threat")
        else:  # Positive or close to 0 are Low Threat
            categories.append("Low Threat")
    return categories
```

Після обробки даних модель надає оцінку аномалії для кожного процесу разом із категорією загрози. Це дозволяє користувачеві зрозуміти, які процеси вважаються підозрілими, та отримати додаткову інформацію для прийняття рішень.

Приклад оцінки для процесу steam.exe:

- Оцінка аномалії:

```
Anomaly score and category for: steam.exe
PID: 145832, Anomaly Score: -0.5954711766333086, Category: Medium Threat
```

Рисунок 4.3 - Оцінка процесу натренованою моделлю

- Закодовані та масштабовані дані для цього процесу:

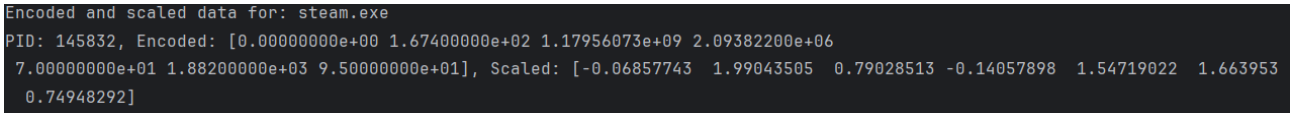

Рисунок 4.4 - Дані які модель отримала для оцінки

1) Anomaly Score — це числове значення, яке показує, наскільки процес відхиляється від нормальної поведінки;

2) Category — визначає рівень загрози на основі порогових значень;

3) Encoded — початкові дані, перетворені у числовий формат для роботи моделі;

4) Scaled — нормалізовані дані, які дозволяють моделі враховувати параметри з різними шкалами.

Переваги класифікації:

5) Адаптивність: можливість налаштування порогів дозволяє користувачам регулювати рівень чутливості залежно від потреб;

6) Прозорість: оцінки аномалій і категорії надають зрозумілу інформацію про стан процесу;

7) Ефективність: поділ на категорії спрощує аналіз великої кількості процесів і допомагає швидше виявляти потенційні загрози.

4.6 Тестування з використанням MITRE Caldera

Для оцінки ефективності розробленої системи використовувався інструмент MITRE Caldera — платформа для моделювання атак, яка дозволяє імітувати поведінку кіберзагроз у контрольованому середовищі. Це дозволяє протестувати систему в умовах, максимально наближених до реальних сценаріїв, без ризику для основного середовища.

MITRE Caldera — це відкрита платформа для моделювання тактик, технік і процедур (TTPs) атак з використанням бази знань ATT&CK. Вона надає інструменти для:

1) Імітації реальних атак: дозволяє виконувати сценарії, які повторюють дії кіберзлочинців;

2) Оцінки захисту системи: перевіряє, як ефективно система виявляє підозрілу активність і реагує на неї;

3) Аналізу поведінки: збирає дані про реакцію цільової системи на моделювання атак.

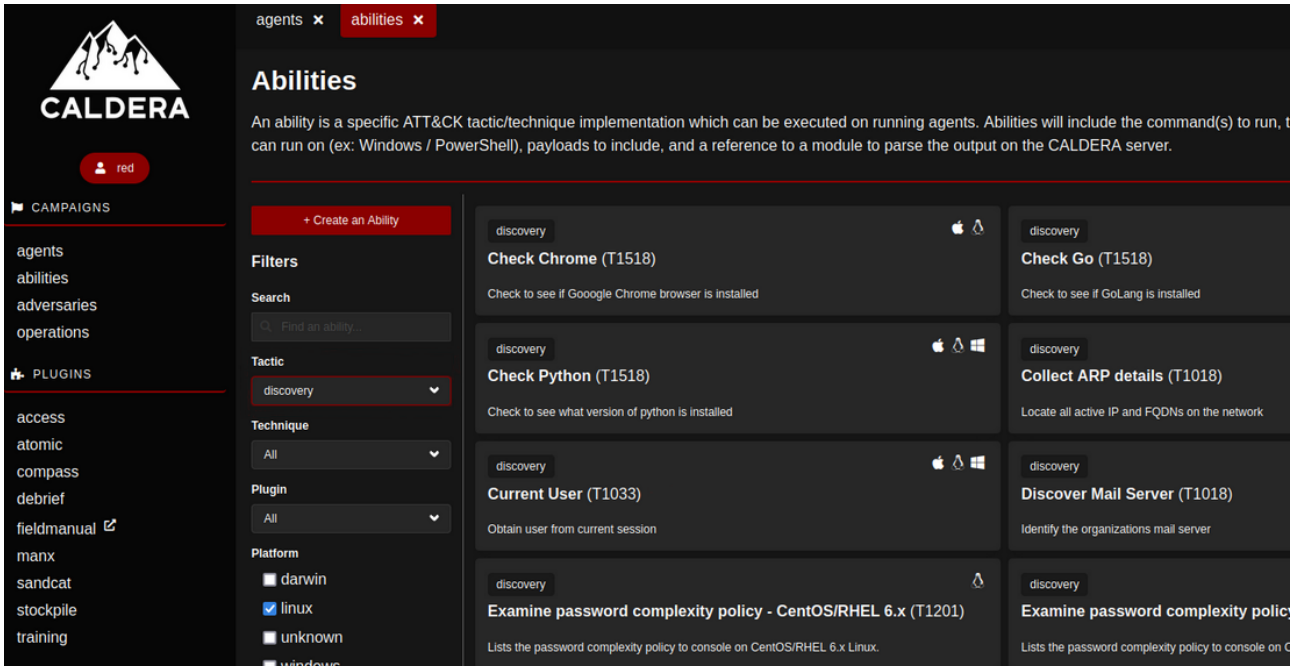

Рисунок 4.5 - Вигляд робочого вікна MITRE Caldera

Причини вибору MITRE Caldera:

1) Реалістичність тестування:

- Платформа дозволяє змоделювати поведінку шкідливого програмного забезпечення, включаючи:

- Використання процесорних ресурсів;

- Зміни в реєстрі;

- Створення або маніпуляцію мережевих з'єднань.

- Створює можливість перевірити, як розроблена система реагує на активність, що нагадує реальні загрози.

2) Контрольоване середовище:

- Усі моделювання виконуються у безпечному тестовому середовищі, що унеможливлює компрометацію основної системи.

3) Широкий спектр сценаріїв:

- MITRE Caldera використовує базу знань ATT&CK для створення сценаріїв, які охоплюють різноманітні техніки атак (наприклад, запуск шкідливих процесів, ексфільтрація даних тощо).

4) Можливість збору метрик:

- Платформа дозволяє аналізувати, які дії були виявлені, які ігноровані, а також оцінювати загальну продуктивність системи.

Алгоритм проведення тестування:

1) Налаштування середовища:

- Розгорнуто платформу MITRE Caldera у тестовому середовищі;

- Створено сценарії атак, які імітують шкідливу активність, наприклад:

- Виконання скриптів для підвищення привілеїв;

- Маніпуляція мережевими з’єднаннями;

- Зміни у критичних реєстрах системи.

2) Взаємодія з розробленою системою:

- Дані, зібрані Caldera, передавалися до системи для аналізу аномалій;

- Оцінювалося, як система виявляє та класифікує підозрілі процеси.

3) Збір результатів:

- Проаналізовано точність ідентифікації аномальних дій;

- Оцінено рівень чутливості системи до різних сценаріїв атак.

Тестування з використанням MITRE Caldera підтвердило здатність системи ідентифікувати аномальну поведінку в реальному часі. Система продемонструвала високий рівень точності у виявленні таких дій, як:

1) Створення незвичних мережевих з'єднань;

2) Аномально високе використання ресурсів;

3) Зміни у реєстрі Windows.

Водночас було виявлено кілька обмежень, таких як складність виявлення малопомітних атак із мінімальним використанням ресурсів, що вимагає подальшого вдосконалення моделі.

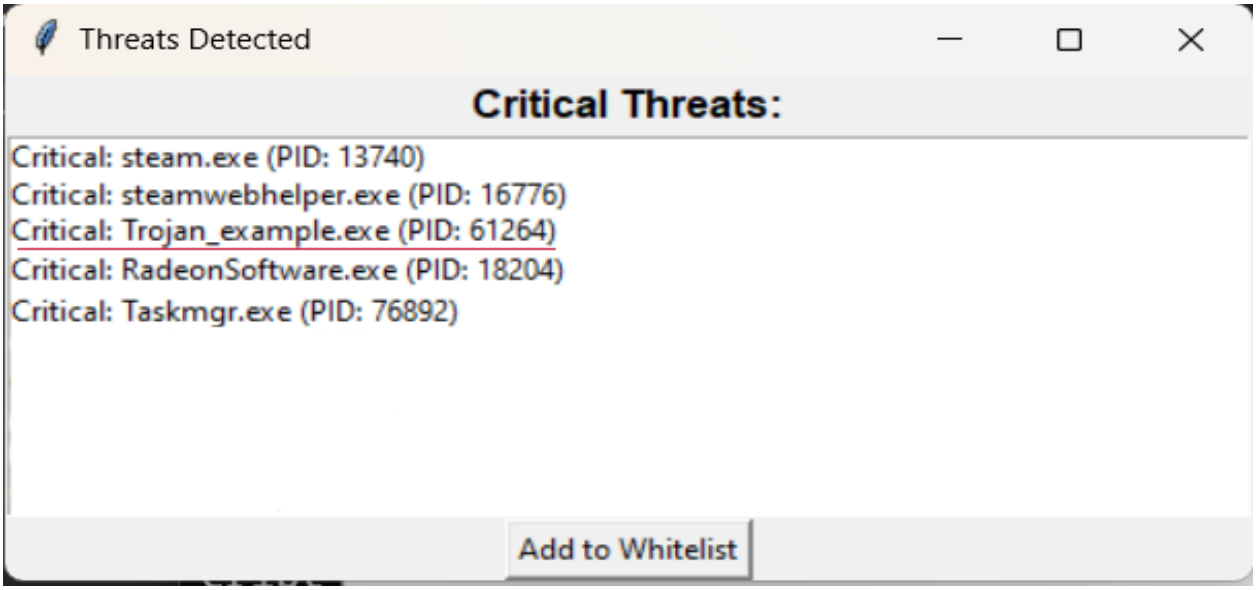

Рисунок 4.6 - Виявлення злоякісного процесу (разом з процесами яких не було в тренувальних даних)

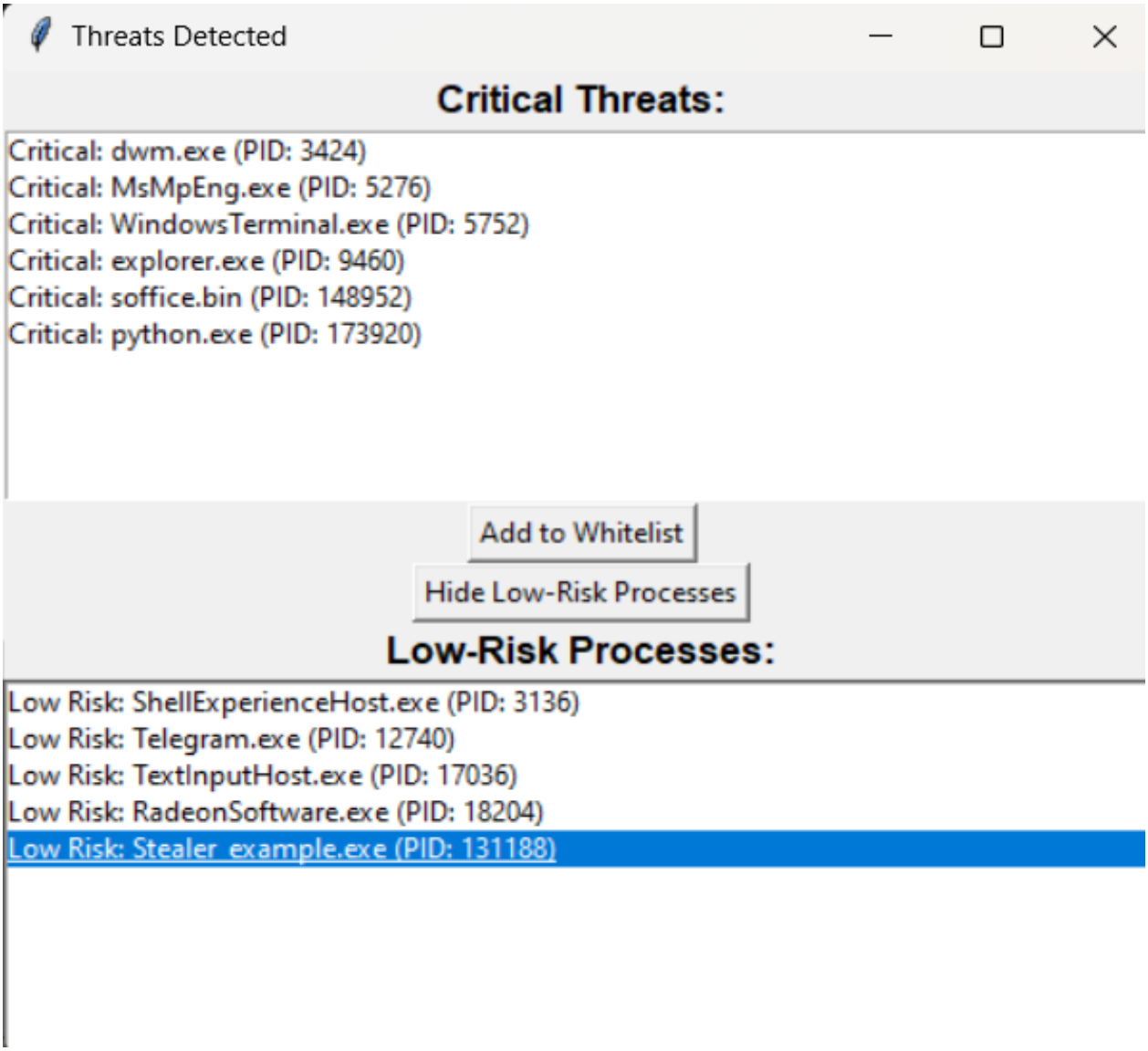

Рисунок 4.7 - Виявлення малопомітного злоякісного процесу при підвищенні чутливості

4.7 Реалізація функцій управління процесами

Програма має декілька ключових кнопок управління процесами, що допомагають користувачеві контролювати процеси та аналізувати шкідливі програми.

Функція "Kill Process". Кнопка "Kill Process" дозволяє користувачеві завершувати вибрані процеси. Вона працює шляхом відправки сигналу на примусове завершення процесу в системі. Це ефективний спосіб зупинити підозрілі процеси, однак такий метод може викликати проблеми для деяких видів шкідливого ПЗ, яке може повторно запускатися або навіть змінювати поведінку при завершенні. Через це інші методи, такі як Suspend, можуть виявитися більш корисними для безпечного аналізу процесів.

Функція "Suspend Process". Кнопка "Suspend Process" призначена для тимчасового зупинення вибраного процесу без його остаточного завершення. Це дозволяє:

- Затримати підозрілий процес для подальшого аналізу, зокрема для виявлення його поведінки без активного впливу на систему;
- Уникнути негайного завершення вірусу, що може активувати захисні механізми шкідливого ПЗ, наприклад, спроби знову запуститися або розширитися на інші частини системи.

Suspend є важливим інструментом для стримування потенційних загроз без ризику негайного поширення або активної відповіді з боку шкідливого ПЗ. Користувач може призупинити процес і потім перевірити його за допомогою інших методів аналізу, таких як VirusTotal або моделі машинного навчання, щоб прийняти обґрунтоване рішення щодо його подальшого видалення.

Функція "Refresh". Кнопка "Refresh" оновлює відображення процесів у реальному часі. Це дозволяє користувачеві бачити зміни в системі та

оновлення інформації про активні процеси, що може бути корисним для динамічного моніторингу та швидкого реагування на зміни.

Функція "Check Threats". Ця кнопка дозволяє запустити перевірку вибраних або всіх процесів за допомогою моделі машинного навчання для виявлення шкідливих активностей. При натисканні на "Check Threats" програма відправляє інформацію про процеси для аналізу, порівнює їх характеристики з попередньо навченою моделлю і повертає результати, які показують рівень ризику. Це допомагає користувачеві швидко оцінити безпеку активних процесів.

Рисунок 4.8 - Вікно потенційних загроз (1)

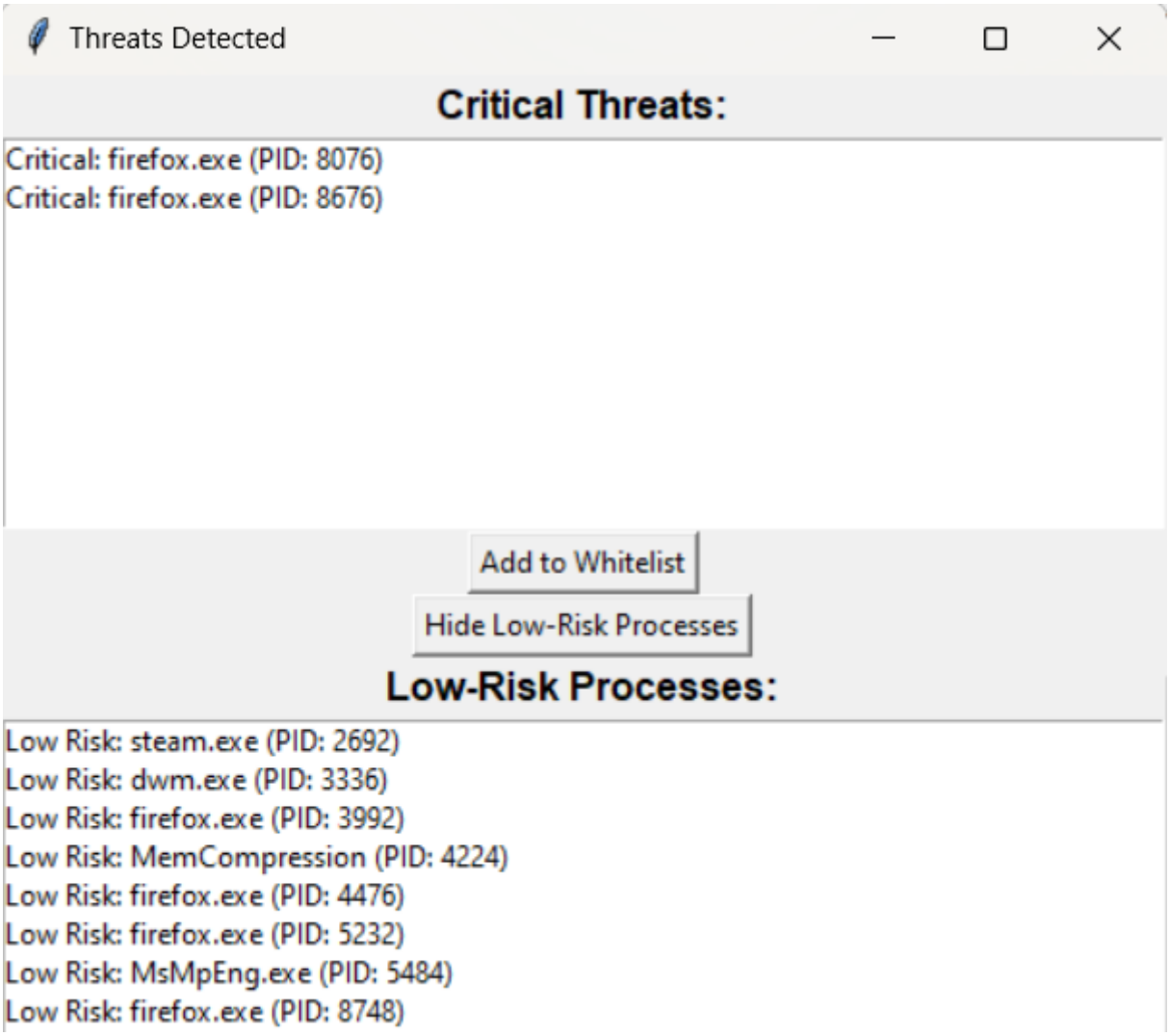

Рисунок 4.9 - Вікно потенційних загроз (2)

Функція "Whitelist". Кнопка "Whitelist" додає певні процеси до списку надійних, що дозволяє програмі не піддавати їх подальшому аналізу. Це особливо корисно для зменшення кількості хибних тривог, оскільки програма запам’ятовує, що певний процес є безпечним, і ігнорує його під час наступних сканувань.

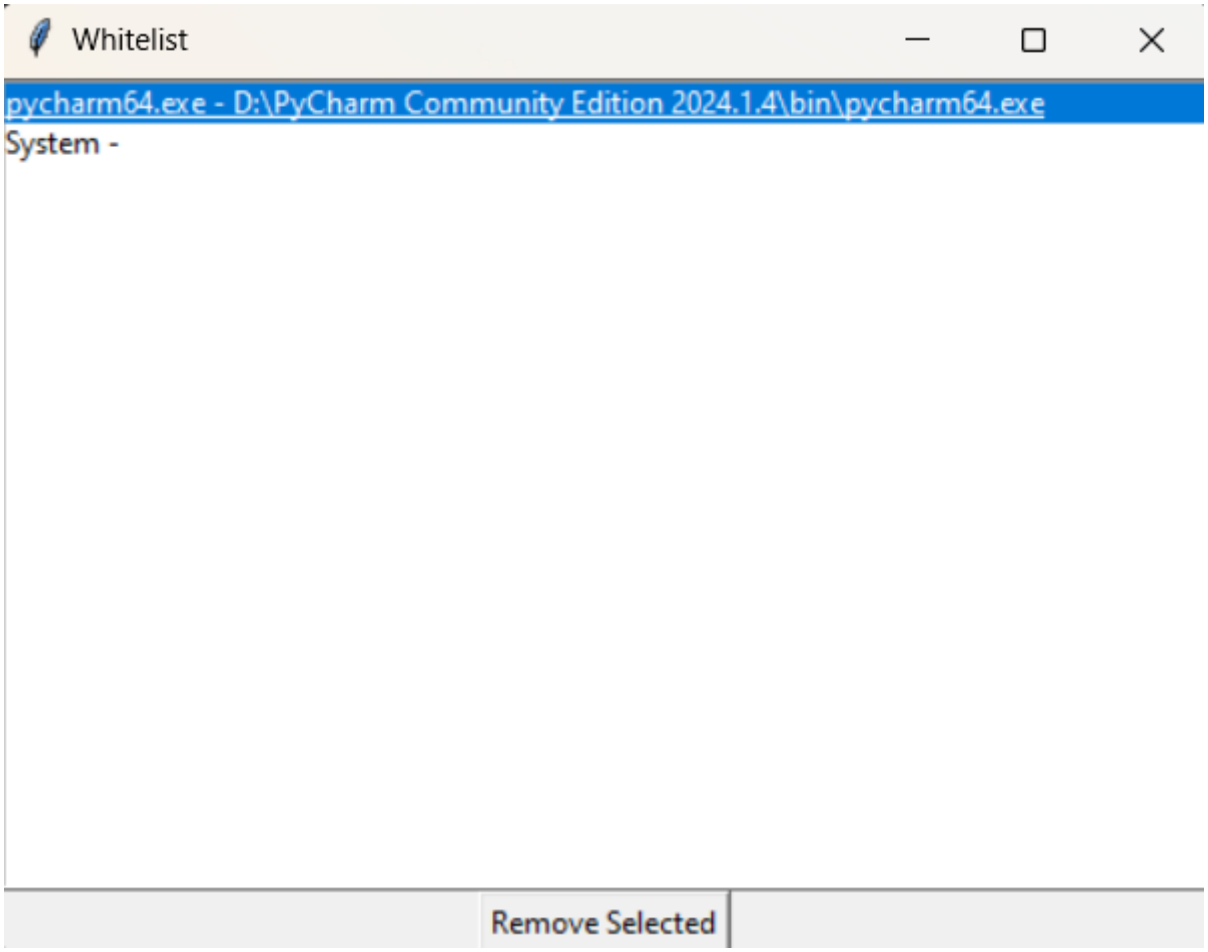

Рисунок 4.10 - Вікно "Whitelist"

5 ПОТЕНЦІАЛ ТА ПЕРСПЕКТИВИ ПРОЕКТУ

5.1 Потенціал подальшого розвитку

Швидка еволюція шкідливого програмного забезпечення, яке адаптується до нових методів захисту, є одним із ключових викликів у сучасній кібербезпеці. Віруси та інші загрози стають дедалі складнішими, використовуючи техніки обфускації, шифрування та динамічного модифікування коду, що ускладнює їхнє виявлення традиційними засобами. У таких умовах запропонована система виявлення аномалій, заснована на машинному навчанні, відкриває нові можливості для ефективного моніторингу процесів із мінімальними витратами системних ресурсів.

Традиційні антивірусні програми виконують локальну перевірку кожного процесу в режимі реального часу, що є ресурсоємним завданням, особливо на менш потужних системах. У цьому контексті запропонована система пропонує альтернативний підхід, який базується на використанні серверної інфраструктури для виконання основних обчислювальних завдань.

Одним із ключових напрямків удосконалення системи є її здатність до персоналізації. Адаптивний підхід передбачає регулярний збір вибірок даних із системи користувача та їхнє передавання на сервер для подальшого навчання моделі. Завдяки цьому модель набуває здатності враховувати специфічні особливості кожної системи, що, у свою чергу, дозволяє значно зменшити кількість хибнопозитивних спрацювань.

Таке адаптивне навчання сприяє підвищенню точності виявлення аномалій, оскільки система здатна розрізняти типові процеси, характерні для конкретного користувача, від нетипової активності, яка може бути пов'язана зі шкідливим програмним забезпеченням. Однак цей підхід також має свої

обмеження, зокрема необхідність захисту переданих даних, щоб уникнути компрометації конфіденційної інформації.

Запропонований підхід значною мірою спирається на перенесення обчислювальних операцій із клієнтської частини системи на сервер. Локальний комп'ютер виконує роль агента, який відповідає за збір даних про активність процесів у реальному часі. Зібрані дані періодично надсилаються на сервер, де відбувається їхній аналіз за допомогою високопродуктивних моделей машинного навчання.

Такий підхід забезпечує значне зниження навантаження на центральний процесор і оперативну пам'ять клієнтської системи. У результаті підвищується загальна продуктивність комп'ютера, що є особливо важливим для користувачів із менш потужним обладнанням. Разом із тим, централізація обчислень вимагає надійної серверної інфраструктури, яка повинна бути здатною обробляти дані від великої кількості клієнтів у реальному часі.

Унікальною перевагою запропонованої системи є її здатність до динамічного оновлення моделей машинного навчання. З огляду на те, що нові віруси та техніки їхньої стелс-активності постійно з'являються, система може інтегрувати механізми автоматичного навчання на основі нових зразків шкідливого програмного забезпечення.

Це дозволяє підтримувати актуальність моделі та підвищувати її ефективність у боротьбі з новими загрозами. Крім того, постійне навчання моделі на основі великої кількості зразків сприяє зменшенню залежності від статичних методів аналізу, які можуть бути неефективними проти сучасного шкідливого програмного забезпечення.

Подальший розвиток системи може включати інтеграцію з існуючими антивірусними програмами та SIEM-системами (Security Information and Event Management). Це забезпечить:

- Централізоване управління безпекою;
- Можливість об'єднання даних із різних джерел для точнішого аналізу;

- Зменшення дублювання функцій між різними системами.

Система виявлення шкідливого програмного забезпечення, яка використовує персоналізовані моделі моніторингу та розподілені обчислення, є перспективним напрямком розвитку кібербезпеки. Поєднання високої продуктивності, адаптивності до змін і здатності знижувати системні витрати робить її однією з найбільш ефективних стратегій боротьби з сучасними кіберзагрозами.

5.2 Персоналізація моделей через серверний моніторинг

Ідея персоналізованого моніторингу на основі машинного навчання передбачає збір анонімізованих даних про поведінку процесів на комп’ютерах користувачів та їхній динамічний аналіз. У великомасштабному проєкті ця інформація могла б регулярно надсилатися на сервер, де централізовано навчалася б модель, адаптована до потреб окремих користувачів або навіть груп користувачів зі схожими характеристиками системи. Це має декілька ключових переваг.

1) Індивідуальні профілі поведінки. Серверний компонент міг би створювати й постійно оновлювати моделі для кожного користувача, враховуючи особливості роботи саме його системи. Це дозволить покращити точність виявлення загроз, оскільки модель зможе адаптуватися до змін у звичних для користувача процесах.

2) Покращена продуктивність. Оскільки обробка основного навантаження відбуватиметься на сервері, це знижує навантаження на локальну машину користувача. Локальні ресурси будуть зайняті лише базовим моніторингом, а складні процеси аналізу та навчання моделі виконуватимуться на сервері, підвищуючи загальну продуктивність.

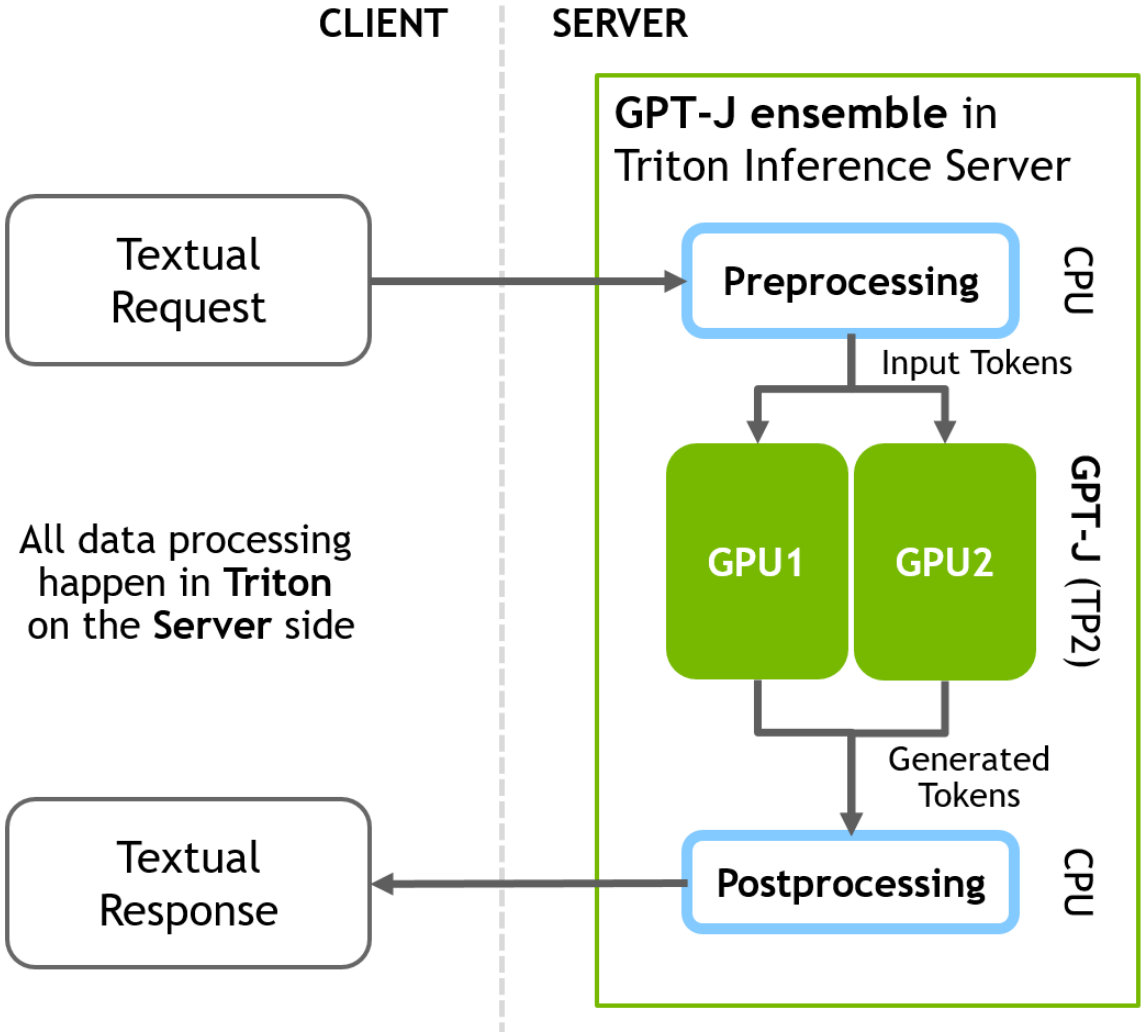

Рисунок 5.1 — Приклад серверного моніторингу

5.3 Інтеграція мережевого моніторингу

Одним із перспективних напрямків розширення функціональності проєкту є додавання мережевого моніторингу для виявлення загроз, що поширюються мережею. Зокрема, це стосується черв'яків, spyware та іншого шкідливого ПЗ, яке може передавати дані або взаємодіяти з іншими системами через інтернет.

1) Виявлення аномального трафіку. Додаток міг би відстежувати активність мережевого трафіку на рівні окремих процесів, виявляючи

нетипову активність, як-от часті з'єднання із зовнішніми IP або великий обсяг даних, що передаються через мережу.

2) Додаткові бібліотеки. Інструменти, такі як scapy або socket, можуть використовуватися для захоплення й аналізу мережевого трафіку на низькому рівні, що дозволить виявляти загрози, орієнтовані на передачу даних.

3) Запобігання мережевим атакам. Окрім виявлення шкідливого ПЗ, що передає дані, мережевий моніторинг також може захищати від DoS/DDoS-атак, виявляючи високий рівень вхідного або вихідного трафіку від окремих процесів.

5.4 Можливі покращення функцій виявлення шкідливого ПЗ

Подальше вдосконалення системи може включати реалізацію більш потужних алгоритмів і розширення функціональності.

Глибоке навчання для точнішого аналізу. Використання нейронних мереж або інших методів глибокого навчання дозволило б системі виявляти складніші патерни поведінки шкідливого ПЗ. Це може бути корисним для виявлення загроз, які важко розпізнати традиційними методами машинного навчання. Наприклад, рекурентні нейронні мережі (RNN) можуть використовуватися для виявлення часових залежностей у поведінці шкідливих процесів.

Адаптивні пороги та правила виявлення. Налаштування та адаптація порогів для певних видів процесів, наприклад, відокремлення базових параметрів для ігор, робочих програм чи браузерів. Це дозволить моделі зберігати точність без надмірної кількості помилкових спрацювань.

Розширений білий список. Білий список міг би автоматично формуватися на основі процесів, які часто з’являються у системі, що значно знизило б кількість хибних тривог. Функціональність білого списку могла б

включати параметри для автоматичного налаштування і фільтрації на основі поведінки кожного користувача.

5.5 Інтеграція з іншими системами кіберзахисту

Щоб створити комплексну систему захисту, може бути використана інтеграція з іншими інструментами кібербезпеки.

Інтеграція з антивірусними програмами дозволила б системі отримувати актуальні дані про нові загрози і реагувати на них більш оперативно.

Фаєрволи та системи виявлення вторгнень (IDS). Підключення до фаєрволів і IDS дозволить системі синхронізувати дані про підозрілу активність із захисними модулями на мережевому рівні. Це забезпечить миттєвий захист від атак, що приходять із зовнішніх мереж.

Централізована панель управління. У великих корпоративних середовищах така система може бути централізовано керована з панелі управління, яка дозволяє переглядати статус всіх пристроїв, підключених до мережі, і отримувати звіти про загрози у реальному часі.

ВИСНОВКИ

В результаті виконання завдання на передатестаційну практику було розроблено систему для моніторингу процесів з використанням методів машинного навчання, яка здатна виявляти потенційно шкідливе програмне забезпечення на основі аномальної поведінки. Проект продемонстрував, що навіть з обмеженими ресурсами можна створити дієву систему для захисту комп'ютерів від загроз, використовуючи ефективні алгоритми машинного навчання та сучасні бібліотеки Python.

Основними результатами виконання практичної роботи стали:
1) Розробка функціонального прототипу: Створено інтуїтивний інтерфейс, який дозволяє користувачеві здійснювати моніторинг активних процесів, завершувати або призупиняти підозрілі процеси, а також запускати аналіз процесів на предмет аномальної поведінки.
2) Реалізація моделі машинного навчання: Було обрано напівконтрольований підхід до навчання для виявлення аномалій у поведінці процесів. Цей підхід показав себе оптимальним для системи з обмеженим обсягом маркованих даних, що дозволило забезпечити високу гнучкість у визначенні нових загроз.

Прототип продемонстрував здатність ефективно виявляти аномалії в локальному середовищі. Проте для подальшого підвищення ефективності система може бути доповнена персоналізованими моделями через серверний моніторинг та інтеграцією мережевого аналізу для більш комплексного захисту.

Незважаючи на досягнуті результати, проект має певні обмеження. Основним є обмеження доступу до сучасних зразків шкідливого ПЗ, що зумовило вибір динамічного, а не статичного аналізу для виявлення загроз.

Окрім того, система не включає моніторинг мережевих з'єднань, що знижує її здатність виявляти загрози, пов’язані з мережевою активністю.

Для майбутніх етапів проекту можливе впровадження персоналізованих моделей на основі даних користувача, що підвищить адаптивність системи. Інтеграція мережевого моніторингу та вдосконалення алгоритмів машинного навчання, зокрема через використання глибокого навчання, можуть значно розширити функціональні можливості системи і підвищити її точність у виявленні складних загроз.

Виконана робота продемонструвала, що методи машинного навчання можуть стати ефективним інструментом для моніторингу і захисту комп'ютерів від шкідливих програм, забезпечуючи швидке реагування та адаптацію до нових видів загроз.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Internet security threat report, 2021/05/10, Symantec Corporation, Tech. Rep. // Symantec Internet Security Threat Report Attack Trends for Q3 and Q4 2021 — 47 с.

2. Sebastien Ziegler, "AI in Cybersecurity: Applications and Best Practices" // IEEE Access, vol. 7, 2020 - 140-157 с.

3. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, // MIT Press, 2016 - 599с.

4. Vincent Vanhoucke, "Efficient Machine Learning for Cybersecurity" // Proceedings of the Conference on Applied Machine Learning for Cyber Defense, 2018 - 240 с.

5. J. Saxe, K. Berlin, "Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features" // 28th Annual Conference on Computer Security Applications (ACSAC), 2019 [Електронний ресурс] /arxiv.org , URL: https://arxiv.org/abs/1508.03096

6. MITRE ATT&CK, base of adversary tactics // [Електронний ресурс] / MITRE.org, URL: https://attack.mitre.org/

7. Machine Learning in Python, API Reference // [Електронний ресурс] / scikit-learn.org , URL: https://scikit-learn.org/stable/