Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Резнікової Марії Андріївни

на здобуття ступеня вищої освіти Бакалавра

Система виявлення шкідливого програмного забезпечення типу "троян" віддаленого доступу в середовищі операційної системи Windows

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

КРБКБ.220253.22.02.34 ПЗ

Виконала студентка 4 курсу, група КБ-22-2 _____ Марія РЕЗНІКОВА
Ініціали, прізвище

Керівник д-р філософії _____ Наталія ПЕТЛЯК
Науковий ступінь, вчене звання Ініціали, прізвище

Нормоконтролер д-р філософії _____ Наталія ПЕТЛЯК
Науковий ступінь, вчене звання Ініціали, прізвище

До захисту допускаю:

Зав. кафедри кібербезпеки _____ Юрій КЛЬОЦ
Ініціали, прізвище

3 06 2026р.

Хмельницький, 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

09 лютого 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Резніковій Марії Андріївні

1 Тема роботи Система виявлення шкідливого програмного забезпечення типу "троян" віддаленого доступу в середовищі операційної системи Windows

Керівник роботи д-р філософії, доцент Петляк Наталія Сергіївна

Затверджено наказом ректора університету від 8 01 2026 № 7

2 Строк подання студентом кваліфікаційної роботи на кафедру 1.06.2026

3 Вихідні дані до роботи Проаналізувати особливості функціонування шкідливого програмного забезпечення класу Remote Access Trojan в ОС Windows. Дослідити механізм проникнення, закріплення та приховування RAT у системі. Провести аналіз сучасних методів виявлення шкідливого ПЗ. Сформувати вимоги до системи виявлення RAT. Розробити алгоритм функціонування системи, реалізувати прототип програмного засобу для виявлення ознак RAT та провести оцінку достовірності запропонованого рішення на контрольованому наборі даних.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Аналіз існуючих методів і систем виявлення шкідливого ПЗ. Формування вимог до системи виявлення RAT. Постановка задачі. Вибір методів та засобів аналізу і моделювання. Розробка алгоритму роботи системи виявлення Remote Access Trojan. Розробка прототипу системи виявлення. Експериментальне дослідження працездатності розробленої системи та оцінка достовірності. Висновки.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Структурна схема розробленої системи виявлення RAT. Схема логіки евристичного аналізу та формування Threat Score. Узагальнена блок-схема алгоритму роботи системи виявлення RAT. Блок-схема алгоритму мережевого моніторингу та фільтрації C2-трафіку. Схема прийняття рішення щодо наявності RAT та реагування системи.

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль			Нормоконтроль

7 Дата видачі завдання 09 лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	
Ознайомлення з предметною областю	Лютий	
Дослідження існуючих рішень	Лютий	
Постановка задачі	Лютий	
Формування вимог до системи	Березень	
Розробка алгоритму виявлення RAT	Березень	
Розробка прототипу системи	Квітень	
Проведення тестування у контрольованому середовищі та оцінка показників ефективності	Квітень	
Оформлення пояснювальної записки згідно вимог	Травень	
Оформлення графічної частини	Травень	
Захист КР	Червень	

Студентка _____ Марія РЕЗНІКОВА

Керівник кваліфікаційної роботи _____ Наталія ПЕТЛЯК

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система виявлення шкідливого програмного забезпечення типу троян віддаленого доступу в середовищі операційної системи Windows.

Автор роботи: Резнікова Марія Андріївна.

Керівник роботи: Петляк Наталя Сергіївна.

Пояснювальна записка: 66 с., 2 додатки, 12 рисунків, 3 таблиці, 41 джерело.

Графічна частина: 5 плакатів.

Ключові слова: RAT, шкідливе програмне забезпечення, виявлення загроз, Windows.

Кваліфікаційна робота бакалавра присвячена розробці системи виявлення шкідливого програмного забезпечення типу “троян” віддаленого доступу у середовищі операційної системи Windows.

В роботі проаналізовано сучасні методи виявлення шкідливого програмного забезпечення, визначено особливості функціонування RAT та характерні поведінкові ознаки їх діяльності. На основі проведеного аналізу сформовано вимоги до системи виявлення та розроблено алгоритм, що поєднує моніторинг поведінки процесів і аналіз мережевої активності.

Розроблено архітектуру системи та реалізовано прототип, проведено експериментальну перевірку ефективності запропонованого підходу. Отримані результати підтверджують доцільність використання комплексного підходу для підвищення точності виявлення RAT.

29.05.2026 _____

ABSTRACT

Subject of qualification work: Detection system for Remote Access Trojan malware in the operating system environment.

Author: Reznikova Mariia Andriivna.

Head of work: Petliak Natalia Serhiivna.

Explanatory note: 66 p., 2 appendices, 12 figures, 3 tables, 41 sources

Graphic part: 5 posters.

Keywords: RAT, malware, threat detection, Windows.

The bachelor's qualification work is devoted to the development of a detection system for Remote Access Trojan malware in the Windows operating system environment.

The work analyzed modern malware detection methods and identified the functional characteristics and behaviors patterns of RATs. Based on the analysis, system requirements were formulated and a detection algorithm combining process behavior monitoring and network activity analysis was developed.

A system architecture was proposed, a prototype was implemented, and an experimental evaluation was conducted. The results confirm the effectiveness of a comprehensive approach to improving RAT detection accuracy.

23.05.2026 _____

ЗМІСТ

ВСТУП

Нині інформаційні технології є центральною частиною щоденної роботи державних установ, комерційних організацій і діяльності користувачів. Паралельно з розвитком цифрової інфраструктури зростає і рівень кіберзагроз. Поміж ниж мпецифічно небезпечними є програми класу троянів віддаленого доступу (Remote Access Trojan, надалі у пояснювальній записці – RAT).

RAT є різновидом шкідливого програмного забезпечення, що забезпечує зловмиснику несанкціонований віддалений доступ до зараженої системи, та надає можливість повного контролю над її ресурсами. На відміну від звичайних троянських програм, RAT дозволяють виконувати команди в режимі реального часу, здійснювати моніторинг активності користувача, отримувати доступ до файлів, облікових даних та периферійних пристроїв [1]. Такий функціонал робить їх однією з найбільш критичних загроз для інформаційної безпеки.

Ця проблема є особливо актуальною у контексті операційної системи Microsoft Windows, оскільки вона залишається найпоширенішою платформою як у приватному, так і в корпоративному секторі. Її масове використання зумовлює підвищений інтерес з боку кіберзлочинців. Відомі сімейства RAT, зокрема Agent Tesla, njRAT та інші, активно застосовуються у реальних кібератаках та шпигунстві. Це спричинено використанням механізми приховування, що суттєво ускладнює їх своєчасне виявлення.

Традиційні антивірусні засоби переважно базуються на сигнатурному підході, який передбачає порівняння об’єктів із відомими зразками шкідливого коду. Проте ефективність такого методу знижується у випадку нових або модифікованих версій RAT, сигнатури яких ще відсутні у базах даних.Також, використання зашифрованих каналів зв’язку з серверами керування ускладнює детекцію на рівні мережевого аналізу.

У сучасних статтях значна увага приділяється альтернативним підходам до виявлення RAT, зокрема поведінковому аналізу процесів, аналізу мережевої активності та застосуванню алгоритмів машинного навчання [2]. Зокрема, на

основі поведінкових ознак, продемонстровано ефективність використання алгоритмів класифікації для розпізнавання шкідливого трафіку. Також розробляються гібридні моделі, що поєднують аналіз хост-ознак (процеси, системні виклики, зміни реєстру) та мережевих характеристик, що підвищує точність виявлення складних загроз і зменшує кількість хибних спрацювань [3].

Таким чином, проблема створення системи виявлення RAT у середовищі Windows є актуальною як з практичної, так і з наукової точки зору. Необхідність розробки адаптивних механізмів детекції обумовлена постійною еволюцією шкідливого програмного забезпечення та ускладненням методів його приховування.

Метою кваліфікаційної роботи є розробка системи виявлення RAT у операційній системі Windows на основі аналізу поведінкових та мережевих характеристик.

Об’єктом роботи є процеси функціонування шкідливого програмного забезпечення класу RAT у середовищі ОС Windows.

Предметом роботи є методи та засоби виявлення RAT на основі аналізу системної та мережевої активності.

Практичне значення кваліфікаційної полягає у можливості використання розробленої системи або її окремих компонентів для підвищення рівня виявлення за характерними ознаками RAT у операційній системі Windows.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ТА СИСТЕМ ВИЯВЛЕННЯ REMOTE ACCESS TROJAN

1.1 Загальні концепції та сутність Remote Access Trojan

У сучасному кіберпросторі RAT посідає особливе місце серед шкідливого програмного забезпечення завдяки своїй функціональній універсальності, гнучкості застосування та високому рівню прихованості. На відміну від традиційних вірусів або мережевих хробаків, що орієнтовані на швидке поширення або порушення працездатності системи, RAT призначені для довготривалого, непомітного контролю над інфікованим середовищем [4]. Основною їх характеристикою є забезпечення стабільного каналу віддаленого управління зловмисником із можливістю виконання довільних команд, ексфільтрації даних та інтеграції додаткових компонентів у вже скомпрометовану систему.

Актуальні звіти міжнародних організацій з кібербезпеки свідчать про те, що інструменти віддаленого доступу активно застосовуються як у цільових атаках на державні структури та великі корпорації, так і в масових кампаніях проти малого бізнесу та приватних користувачів. Згідно з аналітикою Cofense, атаки, пов’язані з використанням механізмів віддаленого керування, залишаються одним із найбільш поширених механізмів прихованого несанкціонального доступу. Також вона демонструючи значне зростання активності протягом останнього року [5]. Подібні тенденції підтверджуються даними компанії CrowdStrike, де наголошується на еволюції методів прихованого закріплення у внутрішніх мережах організацій [6, 7].

Концептуально RAT реалізує архітектурну модель «клієнт-сервер». Після інфікування кінцевого пристрою у середовищі Windows формується агент, який встановлює з’єднання з командно-керуючою інфраструктурою. Важливою особливістю сучасних RAT є використання легітимних протоколів прикладного рівня, як-от HTTPS або TLS-захищених сесій, що дозволяє маскувати шкідливу активність під звичайний веб-трафік. Це суттєво ускладнює виявлення за

допомогою звичних засобів фільтрації мережевих пакетів [8].

Сутність RAT не обмежується лише організацією віддаленого з'єднання. У контексті сучасних загроз такі інструменти інтегрують комплекс функціональних можливостей, які можуть бути використані на різних етапах ланцюгу ураження цілі (англ. kill chain), починаючи від первинного проникнення до внутрішнього переміщення мережею та підвищення привілеїв [9-11].

У середовищі Windows механізми закріплення відіграють головну роль у забезпеченні довготривалої присутності шкідливого агента. Сучасні варіанти RAT можуть змінювати параметри автозапуску, модифікувати розділи реєстру, створювати заплановані завдання або інтегруватися у служби системи. Такий підхід дозволяє відновлювати активність після перезавантаження або спроби часткового видалення. Роботи у сфері поведінкового аналізу підтверджують, що саме специфічні механізми закріплення є ключовою відмінною рисою RAT від інших класів шкідливого ПЗ [12].

Важливим аспектом сутності RAT є їх здатність до приховування та обходу засобів захисту [13]. Сучасні реалізації активно використовують методи обфускації та динамічне завантаження компонентів без збереження на диск. Поширення безфайлового шкідливого ПЗ, коли код виконується безпосередньо в RAM з використанням PowerShell або WMI, значно ускладнило методи детекції, що базуються на аналізі файлової системи [14, 15]. Відповідно, концепція системи виявлення RAT має враховувати не лише наявність підозрілого файлу, але й поведінкові прояви процесів у системі.

З точки зору функціональної класифікації, RAT можуть відрізнятися рівнем складності та призначенням. Деякі з них орієнтовані на шпигунство та збір інформації, інші – на забезпечення доступу до внутрішньої інфраструктури організації з метою подальшого впровадження інших шкідливих компонентів. Аналітичні матеріали компанії Palo Alto Networks демонструють, що інструменти віддаленого доступу активно застосовуються у багатофазних атаках як початковий етап компрометації з подальшим виконанням дій пост-експлуатації. Це включає розвідку, переміщення мережею та підготовку до впровадження

програм-вимагачів [16]. Подібні тенденції підтверджуються також аналітичним оглядом компанії Positive Technologies, де зазначається, що RAT залишаються одними з найпоширеніших типів шкідливого ПЗ в атаках на організації та часто використовуються у поєднанні з програмами-вимагачами [17]. Це означає, що виявлення RAT на ранній стадії має критичне значення для запобігання масштабним інцидентам.

Питання шифрування мережевого трафіку є важливим у контексті сучасного аналізу кіберзагроз. Статті про аналіз зашифрованих з'єднань демонструють, що навіть без розшифрування переданих даних [18–20] можна виявляти аномальні ознаки сесії на основі її статистичних параметрів, таких як тривалість з'єднання та розмір пакетів. Це підкреслює необхідність багаторівневого підходу до аналізу RAT, який включає не лише контроль процесів і файлової активності, але й аналіз мережевих характеристик [21].

Згідно з Microsoft Digital Defense Report [22], Windows залишається основною цільовою платформою через її домінування в корпоративному секторі. Переважання Windows на ринку операційних систем підтверджується статистичними даними StatCounter за 2024 рік, що обумовлює її привабливість як цільового середовища для зловмисників.

Важливим є також аспект адаптивності RAT. Сучасні варіанти часто створюються на основі відкритих фреймворків або комерційних конструкторів, що дозволяє зловмисникам швидко модифікувати конфігурацію, змінювати алгоритми шифрування або диверсифікувати інфраструктуру командних центрів [23, 24]. У результаті формується велика кількість варіантів із незначними відмінностями, що ускладнює їх виявлення сигнатурними засобами. Наукові роботи у галузі машинного навчання показують, що аналіз послідовності системних викликів та шаблонів поведінки дозволяє ідентифікувати такі аномалії навіть за відсутності відомих сигнатур [25].

Таким чином, сучасне розуміння сутності розробки системи виявлення RAT повинно базуватися на комплексному підході, що поєднує архітектурний, функціональний та поведінковий аспекти. RAT є не просто шкідливим файлом, а

інтегрованим механізмом віддаленого управління. Він використовує можливості операційної системи Windows для закріплення, приховування та взаємодії з мережевою інфраструктурою. Виявлення цбого типу шкідливого ПЗ потребує врахування сукупності ознак, що проявляються у процесній активності, системних змінах та мережевих з'єднаннях. Зазначені особливості формують основу для подальшого аналізу існуючих засобів протидії, оскільки ефективна система виявлення повинна враховувати багатовимірність поведінки RAT та їх здатність до адаптації. Саме це зумовлює необхідність переходу від вузькоспеціалізованих сигнатурних рішень до гібридних підходів, що поєднують аналіз процесів, мережевого трафіку та поведінкових характеристик, що буде розглянуто у наступних підрозділах.

1.2 Огляд і класифікація сучасних рішень для виявлення RAT

У рамках аналізу та побудови обґрунтованої системи виявлення RAT необхідно провести ґрунтовний огляд існуючих технічних та наукових рішень, які застосовуються або пропонуються для виявлення цього класу загроз. Сучасні підходи до ідентифікації і детекції RAT суттєво різняться за механізмами роботи, рівнем автоматизації та принципами аналізу, а також за ефективністю застосування у реальних умовах роботи операційних систем. Зокрема і у Microsoft Windows. У науковій літературі виділяють кілька основних категорій рішень, що використовуються для протидії RAT-загрозам. Вони включають у себе традиційні сигнатурні рішення, поведінкові та гібридні системи, методи на основі аналізу мережевого трафіку, а також застосування методів машинного навчання та штучного інтелекту [26-28].

Першою категорією таких рішень є сигнатурні підходи, які найчастіше реалізовані в класичних антивірусних продуктах та базових механізмах захисту операційних систем. Цей підход полягає в порівнянні контрольованих об'єктів (файлів, об'єктів пам'яті або поведінкових патернів) з попередньо визначеними

сигнатурами шкідливого коду. Такий механізм є відносно ефективним проти раніше відомих версій шкідливого ПЗ і здатний забезпечити оперативне виявлення відомих штамів RAT. Проте лише тих, котрі вже внесені до баз даних виробника. Проте ключовим обмеженням сигнатурних методів є їхня обмежена здатність ідентифікувати zero-day загрози або нові варіанти RAT. Це спричинено їхньою модифікацію шляхом обфускації чи шифрування [29]. Як зазначено в аналітичних звітах Microsoft Digital Defense, надмірна залежність від сигнатурних баз призводить до втрати здатності протидіяти адаптивним загрозам, а також до високого рівня хибних негативних результатів у випадках модифікованих або кастомізованих RAT.

Поведінкові підходи є противагою традиційного сигнатурного аналізу. Цей метод виявлення базується на спостереженні за аномальною активністю процесів, системних викликів, взаємодією з об'єктами файлової системи та реєстру, що властиво поведінці шкідливого програмного коду. У поведінковому аналізі фіксується не сам код, а саме його прояв у вигляді послідовності поведінкових ознак, які відрізняють RAT від легальних програм. Як показано в наукових статтях, поведінкові моделі продемонстрували значну ефективність у виявленні раніше невідомих або змінених варіантів RAT. Оскільки вони не залежать від жорстких сигнатурних шаблонів, а фокусуються на аномальних аспектах виконання. Одним із прикладів такого підходу є аналіз системних викликів та відстеження нетипових патернів взаємодії з компонентами Windows API. Це дозволяє визначати потенційно шкідливі процеси навіть у разі їхньої відсутності їх у сигнатурних базах [30].

Важливою підкатегорією поведінкового аналізу **є** комбінація поведінкових та статичних ознак, яку називають гібридним підходом. Такий механізм поєднує сильні сторони обох методів із метою підвищення точності детекції та зниження кількості хибних спрацювань [31]. Наприклад, сучасні моделі, які використовують гібридні ознаки, включають одночасний аналіз структурних характеристик файлів та поведінкових патернів у процесі виконання, що дозволяє досягти кращої адаптації до складних та змінених загроз. Цей підхід знаходить

широке застосування у розробці просунутих EDR рішень. Ці рішення орієнтовані на корпоративні середовища, де вимоги до якісної детекції шкідливої активності значно вищі [32].

Ще однією категорією сучасних рішень є методи на основі аналізу мережевого трафіку. Через те, що RAT зазвичай підтримують зв'язок із віддаленим сервером керування та контролю, аналіз мережевої поведінки є перспективним способом виявлення аномалій, пов'язаних з активністю шкідливого ПЗ. Такі методи включають аналіз потоків трафіку, виявлення нетипових шаблонів, частоти звернень, використання незвичних портів або протоколів, а також статистичний аналіз параметрів криптованих каналів [33].

Відповідно до сучасних наукових публікацій також пропонуються підходи, що залучають глибинний аналіз трафіку з використанням машинного навчання. Це дозволяє створювати моделі для класифікації аномальних комунікацій навіть у разі обфускації або шифрування. Незважаючи на перспективність, такі мережеві рішення потребують значних ресурсів для обробки великих масивів даних і часто вимагають централізованого моніторингу. Це може бути неприйнятним для локальних застосувань без корпоративної інфраструктури.

Останніми роками досить активно розвивається застосування алгоритмів машинного навчання та штучного інтелекту для виявлення RAT. Ці підходи побудовані на створенні моделей, які навчаються на великих наборах даних, що містять як нормальну поведінку систем, так і ознаки поведінки шкідливого ПЗ. Застосування таких моделей дозволяє автоматично виявляти складні залежності між параметрами, які традиційні методи не здатні охопити. Наприклад, пропонується побудова моделей на основі нейронних мереж, Random Forest, SVM та інших алгоритмів, що демонструють високу точність у розпізнаванні шкідливої активності, зокрема у випадках прихованих або модифікованих RAT. Варто зазначити, що ефективність ML-підходів суттєво залежить від якості навчальних даних, а також від здатності моделі узагальнювати нові варіанти загроз, що може вимагати регулярного оновлення тренувальних наборів [34].

Окрему групу рішень становлять системи із заглибленим контекстним

аналізом, які поєднують поведінкові та ML-метрики з контекстною інформацією про середовище виконання. Це дозволяє враховувати не лише технічні ознаки шкідливої активності, а й зміну стану системи у часі, зміну взаємодій між процесами та аномальну поведінку у повному контексті роботи ОС. Наприклад, контекстний аналіз поведінки процесів може враховувати закономірності їх життєвого циклу, взаємозв'язки з іншими модулями системи та часові патерни активності [35].

Серед спеціалізованих засобів виявлення RAT також варто згадати Endpoint Detection and Response рішення, які інтегрують механізми поведінкового моніторингу, аналізу пам'яті, інспекції мережевого трафіку та централізованого збирання логів. EDR-системи забезпечують комплексний огляд стану платформи та можуть виявляти складні загрози, включаючи RAT, через кореляцію між локальними індикаторами компрометації та глобальними шаблонами поведінки. Вони зазвичай застосовуються у корпоративних середовищах і характеризуються високою ефективністю, але одночасно мають недоліки з огляду на вартість впровадження та необхідність фахової підтримки.

Крім описаних категорій, досліджується також застосування гібридних підходів із включенням Threat Intelligence Feed, що дозволяє співставляти поведінкові ознаки з актуальними трендами загроз, накопичених у спеціальних базах знань. Це забезпечує додаткове джерело інформації для оцінки ризику та контекстуальної інтерпретації подій, пов'язаних із потенційною активністю RAT.

Отже, існуючі технічні рішеня для виявлення RAT характеризується великою різноманітністю підходів, кожен із яких має свої переваги й обмеження. Сигнатурні методи є ефективними для відомих загроз, але виявляються слабкими проти нових або модифікованих RAT. Поведінкові та гібридні методи забезпечують глибший аналіз, проте потребують складнішої реалізації. Методи на основі мережевого аналізу та машинного навчання відкривають нові можливості для детекції аномалій, але вимагають значних обчислювальних ресурсів і централізації даних. Така класифікація дозволяє побудувати основу для подальшого порівняння цих рішень та формування вимог до запропонованої

системи.

1.3 Порівняльний аналіз сучасних рішень для виявлення RAT

Результати аналізу загроз, наведених у сучасних аналітичних звітах з кібербезпеки, свідчать про зростання частки атак, пов’язаних із використанням RAT у корпоративних та державних інформаційних системах. Особливістю сучасних RAT є їх модульність, здатність до шифрування трафіку, використання легітимних системних компонентів Windows та застосування безфайлових технік виконання коду, що істотно ускладнює їх виявлення традиційними засобами захисту [36]. У зв’язку з цим виникає необхідність порівняльного аналізу існуючих рішень з метою визначення їх функціональних можливостей та виявлення обмежень.

У кваліфікаційній роботі було розглянуто декілька категорій систем: класичні сигнатурні антивіруси, комерційні платформи класу EDR/XDR, open-source рішення моніторингу подій та наукові розробки на основі методів машинного навчання. Такий підхід дозволяє охопити як масові програмні продукти, так і перспективні напрямки розвитку технологій детектування.

Традиційні антивірусні рішення залишаються найпоширенішим засобом захисту робочих станцій у середовищі Windows. Їх функціонування базується переважно на сигнатурному аналізі, що передбачає пошук у файлах і пам’яті відомих шаблонів шкідливого коду. Згідно з результатами незалежних тестувань AV-TEST, такі рішення демонструють високий рівень виявлення відомих зразків шкідливого програмного забезпечення. Проте сигнатурний підхід має принципове обмеження. Він не забезпечує ефективного захисту від модифікованих або раніше невідомих варіантів RAT. Таким чином, традиційний антивірус не може розглядатися як достатній засіб протидії складним сценаріям віддаленого доступу.

Більш розвиненим класом рішень є платформи Endpoint Detection and Response. Платформа Microsoft Defender for Endpoint поєднує сигнатурний та

поведінковий аналіз, моніторинг процесів і мережеву аналітику, що дозволяє ефективно виявляти складні та адаптивні загрози [37]. Впровадження таких систем зазвичай потребує корпоративної ліцензії та інтеграції з централізованою інфраструктурою, що може створювати обмеження для малих підприємств.

Подібні характеристики притаманні й іншим провідним EDR/XDR-платформам, зокрема CrowdStrike Falcon та SentinelOne Singularity [38, 39]. Ці системи активно застосовують поведінкову аналітику та алгоритми машинного навчання для виявлення безфайлових атак і складних ланцюгів виконання шкідливих дій. Вони забезпечують глибокий аналіз процесів, інжекції коду, змін у реєстрі та мережевих з'єднань. Водночас архітектура таких платформ передбачає централізовану обробку даних у хмарному середовищі, що ускладнює їх автономне використання. Більшість складних атак виявляються саме завдяки комплексній хмарній кореляції подій, однак це одночасно підвищує залежність від зовнішніх сервісів.

Open-source рішення Wazuh забезпечує моніторинг журналів подій Windows, контроль цілісності файлів і базовий аналіз аномалій на основі правил. Система може бути інтегрована з SIEM-платформами та використовуватися для централізованого збору логів. Проте її ефективність значною мірою залежить від якості налаштування та наявності серверної інфраструктури. Для невеликих організацій без спеціалізованого персоналу розгортання такої системи може бути надмірно складним.

Також значну категорію становлять методи машинного навчання для детектування RAT. У роботах продемонстровано ефективність аналізу поведінкових послідовностей, API-викликів та мережевих характеристик із використанням алгоритмів Random Forest, XGBoost, SVM та глибинних нейронних мереж. Такі підходи дозволяють виявляти аномальні патерни навіть за відсутності відомих сигнатур, що робить їх перспективними для захисту від модифікованих та прихованих загроз. Водночас більшість досліджень проводиться в лабораторних умовах, без повної інтеграції з операційною системою та без урахування обмежень реального середовища, а також

залишаються відкритими питання ресурсомісткості.

Для систематизації результатів проведено порівняння розглянутих рішень за такими критеріями: наявність сигнатурного аналізу, поведінкового аналізу, аналізу мережевої активності, аналізу процесів, можливість автономної локальної роботи та доступність для малого бізнесу.

Таблиця 1.1 – Порівняльна характеристика сучасних рішень для виявлення RAT

Критерії порівняння	Традиційний антивірус	Microsoft Defender for Endpoint	CrowdStrike / SentinelOne	Wazuh	ML-моделі
Сигнатурний аналіз	+	+	-	-	-
Поведінковий аналіз	-	+	+	+	+
Аналіз мережі	-	+	+	+	±
Аналіз процесів	-	+	+	+	+
Автономна робота	+	+	-	-	±
Доступність для малого бізнесу	+	-	-	+	-

Отримані результати свідчать про відсутність рішення, яке б одночасно забезпечувало повний функціональний набір, автономність та доступність для невеликих організацій. Найбільш функціональні платформи є дорогими та орієнтованими на корпоративний сегмент. Натомість доступні рішення не забезпечують комплексного багаторівневого аналізу.

Таким чином, формується обґрунтована потреба у створенні локальної системи виявлення RAT для ОС Windows, яка поєднувала б сигнатурний аналіз із

поведінковим моніторингом процесів та базовим контролем мережевої активності без залежності від хмарної інфраструктури. Саме такий підхід дозволить забезпечити баланс між ефективністю, автономністю та економічною доцільністю, що є особливо актуальним для малого бізнесу та навчальних установ.

1.4 Формування вимог до розроблювальної системи

Проведений у попередніх підрозділах аналіз сучасного стану загроз, пов’язаних із використанням RAT, а також порівняльна оцінка існуючих рішень дозволяють сформулювати обґрунтовані вимоги до системи, яка розроблятиметься в межах даної роботи. Формування таких вимог повинно базуватися не на абстрактному переліку бажаних характеристик, а на чітко визначених проблемах, що залишаються невирішеними у наявних підходах.

Згідно з аналітичними звітами ENISA та IBM X-Force, сучасні RAT характеризуються багатоступеневою архітектурою, використанням зашифрованих каналів зв’язку, механізмами закріплення в системі та можливістю дистанційного оновлення функціоналу [40, 41]. У багатьох випадках зловмисники застосовують легітимні компоненти Windows для виконання шкідливих дій, що дозволяє обходити традиційні сигнатурні механізми. Такі методи приховування активності ускладнюють виявлення RAT і підкреслюють необхідність застосування поведінкових, мережевих та машинно-навчальних підходів для ефективного захисту кінцевих пристроїв.

У підрозділі 1.3 встановлено, що класичні антивірусні рішення забезпечують лише сигнатурний аналіз, що не є достатнім для виявлення модифікованих або раніше невідомих варіантів RAT. Натомість повнофункціональні EDR/XDR-платформи, такі як Microsoft Defender for Endpoint, CrowdStrike Falcon або SentinelOne Singularity, реалізують комплексний багаторівневий підхід, однак потребують централізованої хмарної інфраструктури

та суттєвих фінансових витрат. Це створює обмеження для їх використання у малому бізнесі, навчальних закладах або невеликих організаціях, що не мають власного центру кібербезпеки.

Результати сучасних наукових досліджень також демонструють високу ефективність поведінкових моделей та алгоритмів машинного навчання для детектування шкідливого програмного забезпечення, проте такі рішення часто залишаються експериментальними і не адаптованими до повноцінної інтеграції в операційну систему Windows з урахуванням обмежень продуктивності та ресурсів.

Виходячи з наведеного, можна сформулювати ключову проблему: відсутність легковагової, автономної системи для ОС Windows, яка поєднувала б базові переваги сигнатурного та поведінкового підходів, забезпечувала б контроль процесної та мережевої активності та при цьому залишалася доступною з точки зору впровадження та експлуатації.

На основі цього визначаються такі концептуальні вимоги до розроблюваної системи.

Передусім система повинна забезпечувати комбінований механізм виявлення. З огляду на те, що сигнатурний аналіз залишається ефективним щодо відомих загроз, доцільно включити базовий механізм перевірки файлів та процесів за набором сигнатур або хеш-значень. Однак цей механізм має бути доповнений поведінковим моніторингом, орієнтованим на аналіз підозрілих дій процесів, таких як створення нових служб, модифікація реєстру, запуск дочірніх процесів із нетиповими параметрами або встановлення мережевих з’єднань із невідомими вузлами. Саме комбінування підходів дозволяє підвищити рівень виявлення без значного ускладнення архітектури системи.

Другою принциповою вимогою є локальна автономність функціонування. Аналіз рішень класу EDR/XDR показує їх залежність від хмарної інфраструктури та централізованої кореляції подій. В рамках даної роботи доцільно орієнтуватися на систему, здатну працювати без обов’язкового підключення до зовнішніх сервісів. Такий підхід підвищує незалежність користувача та зменшує ризики

витоку даних моніторингу системної активності.

Наступною вимогою є можливість аналізу процесної активності на рівні операційної системи Windows. Більшість технік, пов’язаних із функціонуванням RAT, реалізуються через маніпуляції процесами та використання стандартних системних інструментів. Тому система повинна здійснювати моніторинг створення процесів, інжекції коду, змін у реєстрі та підозрілих викликів API. При цьому необхідно забезпечити мінімальний вплив на продуктивність робочої станції.

Важливою складовою є базовий контроль мережевої активності. Сучасні RAT майже завжди передбачають встановлення каналу керування з віддаленим сервером. Навіть якщо трафік зашифрований, аномальні характеристики з’єднання (частота звернень, використання нетипових портів, комунікація з вузлами з низькою репутацією) можуть бути ознаками компрометації. Враховуючи складність повноцінного мережевого аналізу, доцільно реалізувати спрощений механізм фіксації підозрілих з’єднань з можливістю подальшої інтерпретації.

Важливим аспектом є вимога до ресурсної ефективності. Сучасні системи захисту часто використовують складні алгоритми обробки телеметрії, що призводить до високого навантаження на процесор і оперативну пам’ять. У рамках каліфікаційної роботи доцільно орієнтуватися на легковагову архітектуру, яка дозволяє забезпечити достатній рівень виявлення загроз без істотного впливу на продуктивність системи.

Крім технічних характеристик, система повинна відповідати організаційним вимогам, зокрема простоті розгортання та експлуатації. Аналітичні звіти з кібербезпеки підкреслюють, що для малого бізнесу складність впровадження часто є критичним бар’єром. Тому архітектура розроблюваної системи має передбачати мінімальну кількість залежностей, відсутність необхідності складної серверної інфраструктури та інтуїтивно зрозумілий механізм налаштування.

Таким чином, узагальнюючи викладене, можна сформулювати основні функціональні та нефункціональні вимоги до системи виявлення RAT у ОС

Windows:

– використання комбінованого підходу (сигнатурний + поведінковий аналіз);

– моніторинг процесної активності та ключових системних подій;

– базовий контроль мережевих з’єднань;

– автономна локальна робота без обов’язкової хмарної інфраструктури;

– мінімальне навантаження на ресурси системи;

– доступність і придатність для малого бізнесу та навчальних установ.

Реалізація системи, що відповідає зазначеним вимогам, дозволить усунути частину недоліків, виявлених у процесі порівняльного аналізу існуючих рішень, і забезпечить раціональний компроміс між функціональністю, автономністю та економічною доцільністю. Саме ці положення слугуватимуть теоретичною основою для проєктування архітектури системи у другому розділі роботи.

1.5 Постановка задачі

Проведений аналіз показав, що RAT залишаються загрозою та, що існуючі рішення або мають високу вартість і залежність від хмарної інфраструктури, або обмежені сигнатурним підходом, що створює захисний вакуум для малого бізнесу та автономних мереж.

Метою кваліфікаційної роботи є розробка автономної, ресурсоефективної системи виявлення RAT у середовищі ОС Windows, що функціонує на основі гібридної інтеграції евристичного скорингу процесів та фільтрації мережевої телеметрії.

Досягнення визначеної мети реалізується через розв'язання наступних практичних завдань:

– реалізувати модуль збору системної телеметрії для моніторингу активних процесів, верифікації шляхів файлів та ідентифікації обфускованих аргументів командного рядка;

– розробити алгоритм виявлення механізмів персистентності шляхом аудиту критичних гілок системного реєстру Windows та контролю виконання коду з тимчасових директорій користувача;

– спроектувати підсистему контролю мережевої активності, здатну фільтрувати локальний IPv4-трафік, фіксувати Beaconing-аномалії у 10-секундних вікнах та корелювати використання нестандартних RAT-портів у стані ESTABLISHED/LISTEN;

– сформувати математичну модель прийняття рішень на основі евристичного скорингу, що дозволить автоматично класифікувати рівень загрози від 0 до 100 балів за сукупністю виявлених системних аномалій;

– створити програмний прототип, який забезпечить візуалізацію контексту розслідування наявності RAT, управління локальними базами індикаторів компрометації та можливість автоматичного експорту аналітичних звітів.

Успішне виконання поставлених завдань дозволить реалізувати систему виявлення RAT для вирішення задачі їх виявлення, за характерними ознаками. Це дозволить створити інструмент, що заповнить нішу між базовими антивірусами та дорогоми EDR-платформами Також очікується, що розроблена система покращить ідентифікацію прихованої активності зловмисників, в умовах ізольованих мереж.

2 ПРОЄКТУВАННЯ СИСТЕМИ ВИЯВЛЕННЯ RAT

2.1 Постановка задачі та вибір інструментальних засобів

Відповідно до вимог, визначених у першому розділі, система виявлення RAT для ОС Windows повинна забезпечувати ефективне виявлення прихованої активності троянів. Їх особливістю є використання методів обходу захисту та приховування своєї роботи. Зокрема, зловмисники все частіше застосовують підхід «Living off the Land», використовуючи легітимні системні утиліти, наприклад PowerShell або cmd.exe, для виконання шкідливих дій. Через це під час розробки системи було вирішено відмовитися від суто сигнатурного підходу та реалізувати багаторівневу систему виявлення.

Для реалізації такого підходу система повинна виконувати кілька основних задач. Першою є моніторинг процесів та їхніх параметрів. На цьому етапі здійснюється збір інформації про активні процеси, перевірка шляхів до виконуваних файлів і аналіз аргументів командного рядка. Аналіз параметрів запуску дозволяє виявляти безфайлові атаки, коли шкідливий код передається системним інтерпретаторам у закодованому вигляді. Одним із прикладів є використання параметра -enc у PowerShell. Додатково система повинна виявляти спроби маскування процесів шляхом перевірки відповідності назв системних процесів їхнім реальним директоріям запуску.

Наступним важливим завданням є виявлення механізмів персистентності. Для цього програмний агент має перевіряти гілки Run системного реєстру Windows. Це пов’язане з тим, що RAT активно використовують ці записи для автоматичного запуску після перезавантаження системи, приховуючи свої файли у директоріях AppData та Temp.

Ще одним компонентом є контроль мережевої активності та її фільтрація. Система повинна виконувати контекстний аналіз мережевих з’єднань, який включає:

– кореляцію портів;

– фільтрацію локального трафіку;

– виявлення beaconing-активності.

Кореляція портів передбачає виявлення використання мережевих портів, які часто застосовуються RAT-програмами.

Для зменшення кількості хибних спрацьовувань система має відокремлювати зовнішні IP-адреси від внутрішніх мережевих з'єднань.

Виявлення Beaconing-активності базується на фіксації підозріло великої кількості мережевих запитів від одного процесу за короткий проміжок часу. Така поведінка характерна для взаємодії зараженого пристрою з командно-контрольним сервером.

Усі зібрані дані об'єднуються на етапі прийняття рішень за допомогою гібридного евристичного алгоритму. На основі знайдених індикаторів компрометації система формує інтегральний показник ризику для кожного процесу. В додаток до цього, алгоритм враховує не лише окремі ознаки, а і їх поєднання. Наприклад, запуск процесу з тимчасової директорії разом із встановленням зовнішнього мережевого з'єднання підвищує рівень ризику, оскільки така поведінка нетипова для легітимного програмного забезпечення.

Правила нарахування Threat Score у системі наведені в таблиці 2.1.

Таблиця 2.1 – Правила формування Threat Score

Категорія	Ознака компрометації	Нарахований бал
1	2	3
Сигнатурний аналіз	Збіг SHA-256 хешу з локальною базою відомих RAT	100
Аналіз шляху	Запуск з директорій користувача AppData або Temp	+ 30
Маскування	Імена svchost, explorer, lsass, csrss, winlogon поза системними папками System32/Windows	+ 45

Кінець таблиці 2.1

1	2	3
Командний рядок	Прихований запуск PowerShell або завантаження Payload	+60
Командний рядок	Фонове виконання команд	+ 25
Мережа (IP)	З'єднання з адресою, що входить до Blacklist	+ 60
Мережа (порти)	Відкритий відомий RAT-порт	+ 40
Мережа (аномалії)	Наявність зовнішніх з'єднань з публічними IP	+ 15
Мережа (аномалії)	Масові мережеві з'єднання (> 10 зовнішніх IP одночасно)	+ 30
Мережа (beaconing)	Понад 5 мережевих запитів від процесу за останні 10 секунд	+ 40
Персистентність	Присутність шляху до файлу в ключах автозавантаження реєстру	+ 35
Комбінований патерн	Запуск з Temp та встановлено мережеве з'єднання	+ 20
Комбінований патерн	Виклик PowerShell та встановлено мережеве з'єднання	+ 25
Комбінований патерн	Наявність в автозапускута встановлено мережеве з'єднання	+ 20
Виключення	Наявність імені процесу або його хешу в "Білому списку"	0

Варто зазначити, що запропонований підхід є спрощеним і не враховує поведінку, яку використовують сучасні RAT-програми для приховування Beaconing-активності. Попри це, він дозволяє виявляти базові сценарії періодичних звернень до командно-контрольних серверів. Також застосування

статичного списку виключень має певні обмеження через постійні зміни сучасної C2-інфраструктури, тому такий механізм використовується лише як додатковий індикатор підозрілої активності.

Для практичної реалізації системи було обрано мову програмування Python. Такий вибір пов’язаний із наявністю великої кількості бібліотек для роботи з API операційної системи Windows. Зокрема, бібліотека psutil використовується для отримання інформації про процеси та мережеві з’єднання, модуль winreg для аналізу системного реєстру, а ipaddress для класифікації мережевих підключень. Графічний інтерфейс користувача реалізовано за допомогою бібліотеки customtkinter. Це дозволило створити зручний та зрозумілий інтерфейс без надмірного ускладнення програмного коду. Формування аналітичних звітів реалізовано засобами бібліотеки reportlab. Розробка, тестування та налагодження програми виконувалися у середовищі Visual Studio Code з використанням офіційного Python-розширення.

Порогові значення для прийняття рішень визначаються на основі отриманого показника Threat Score, який реалізовано в програмному коді системи. Реакції системи на різні рівні загроз залежно від набраної кількості балів наведені в таблиці 2.2.

Таблиця 2.2 – Рівні інтерпретації Threat Score та функціональна реакція системи

ThreatScore	Класифікація загрози	Реакція системи
0	Безпечний процес	Ігнорування
1 – 39	Легітимний процес	Статус «ІНФО». Реєстрація події для загального моніторингу
40 – 74	Підозріла активність	Статус «ПІДОЗРІЛО». Занесення до звіту та виділення жовтим індикатором

Кінець таблиці 2.2

75 – 100	Критична загроза	Статус «КРИТИЧНО». Візуальне сповіщення, червоний індикатор, детальний опис

Вибір інструментальних засобів для створення прототипу обумовлений необхідністю забезпечення високої продуктивності та мінімального навантаження на системні ресурси. Основною мовою розробки було обрано Python 3.12. Такий вибір пояснюється не лише швидкістю створення програмного забезпечення, а й можливістю подальшого впровадження методів машинного навчання для аналізу аномалій у зашифрованому трафіку без потреби його дешифрування.

Для реалізації системи виявлення RAT використано такі спеціалізовані бібліотеки:

– psutil;
– ipaddress;
– winreg;
– hashlib;
– customtkinter.

Бібліотека ipaddress використовується для фільтрації мережевого трафіку. Вона дозволяє визначати, чи належить IP-адреса до приватного або публічного простору. Це дає змогу відокремлювати внутрішній мережевий трафік від зовнішніх підключень, що важливо для виявлення можливих каналів зв’язку з командно-контрольними серверами та зменшення кількості хибних спрацювань.

Модуль winreg забезпечує роботу з реєстром операційної системи Windows через системний API. Його використання дозволяє перевіряти ключі автозавантаження та виявляти спроби закріплення шкідливого програмного забезпечення в системі. Подібний механізм є одним із найпоширеніших способів забезпечення персистентності RAT.

Бібліотека hashlib використовується для обчислення криптографічних хеш-функцій, зокрема алгоритму SHA-256. Отримані хеші виконуваних файлів порівнюються з локальною базою індикаторів компрометації. Це дозволяє

реалізувати сигнатурний рівень виявлення та знаходити відомі зразки шкідливого програмного забезпечення.

Бібліотека customtkinter застосовується для створення графічного інтерфейсу користувача. Вона дозволяє реалізувати сучасний та зручний GUI з підтримкою асинхронного оновлення даних. Завдяки цьому результати аналізу можуть відображатися в реальному часі без значного навантаження на систему, що підвищує зручність роботи користувача або адміністратора.

Запропонована архітектура поєднує методи статичного аналізу та динамічного поведінкового моніторингу. Такий гібридний підхід дозволяє ефективно виявляти сучасні RAT-програми, зберігаючи автономність роботи агента та незалежність від зовнішніх хмарних сервісів.

2.2 Архітектурна та струкрна організація системи

Проєктована система побудована на багаторівневій модульній архітектурі збору та обробки телеметрії в режимі реального часу. Такий підхід наближений до концепції легких EDR-рішень користувацького рівня. Кожен модуль виконує окрему функцію, що дозволяє зменшити навантаження на операційну систему та оптимізувати використання ресурсів комп’ютера.

Структура розробленого прототипу складається з чотирьох основних функціональних блоків:

- модуль збору телеметрії;
- модуль бази знань;
- аналітичне ядро;
- модуль інтерфейсу та звітності.

Модуль збору телеметрії відповідає за отримання первинних даних про стан операційної системи. Він виконує звернення до системних API для збору інформації про активні процеси, їхню ієрархію, аргументи командного рядка, мережеві підключення та ключі автозавантаження реєстру Windows. Зібрані дані

передаються до інших компонентів для подальшого аналізу.

Модуль бази знань виконує роль локального сховища індикаторів компрометації. До його складу входять бази SHA-256 хешів відомих шкідливих файлів, списки підозрілих IP-адрес, а також білі списки довірених процесів і хешів. Використання такого підходу дозволяє реалізувати сигнатурне виявлення загроз та знизити кількість хибних спрацювань за рахунок виключення легітимного програмного забезпечення.

Центральним компонентом системи є аналітичне ядро, яке реалізує основну логіку виявлення загроз. У цьому модуля виконується гібридний аналіз, що поєднує сигнатурну перевірку з поведінковим та евристичним аналізом. Під час роботи оцінюються аномалії командного рядка, мережеві шаблони активності, частота мережевих звернень, підміна шляхів запуску процесів та спроби закріплення у системному реєстрі. Результатом аналізу є формування інтегрального показника Threat Score та переліку причин, через які процес був визначений як потенційно небезпечний.

Модуль інтерфейсу та звітності забезпечує взаємодію системи з користувачем і відображення результатів роботи. Він дозволяє переглядати виявлені загрози у табличному вигляді, отримувати детальну інформацію про підозрілі процеси та формувати звіти у текстовому й документному форматах. Окрім цього, модуль реалізує механізм сповіщень про критичні події, що дозволяє швидше реагувати на можливі інциденти інформаційної безпеки.

Логічна взаємодія та вектори потоків даних між компонентами представлені на структурній схемі, що зображена на рис. 2.1.

Першим етапом роботи системи є нормалізація даних, під час якої метадані від сенсорів приводяться до єдиного формату. Наприклад, мережевий модуль за допомогою бібліотеки ipaddress класифікує підключення як внутрішні або зовнішні ще до початку оцінки ризику. Після цього аналітичне ядро виконує кореляцію подій і виявляє складні поведінкові шаблони. Наприклад поєднання запуску процесу з тимчасової директорії та Beaconing-активності. Остаточне рішення формується на основі показника Threat Score. Це дозволяє

відокремлювати звичайні системні аномалії від цілеспрямованої активності RAT.

Для збору телеметрії процесів у Windows використовується бібліотека psutil, яка забезпечує доступ до атрибутів процесів у реальному часі. Особлива увага приділяється аналізу аргументів командного рядка та директорій запуску. Легітимні системні утиліти зазвичай працюють зі стандартними параметрами зі своїх основних каталогів, тоді як RAT-програми часто використовують нетипові сценарії запуску. Наприклад, застосування параметра -enc у PowerShell або команд типу downloadstring може свідчити про спробу виконання безфайлової атаки. Додатково запуск системних компонентів поза директорією System32 розцінюється системою як ознака маскування або впровадження шкідливого коду.

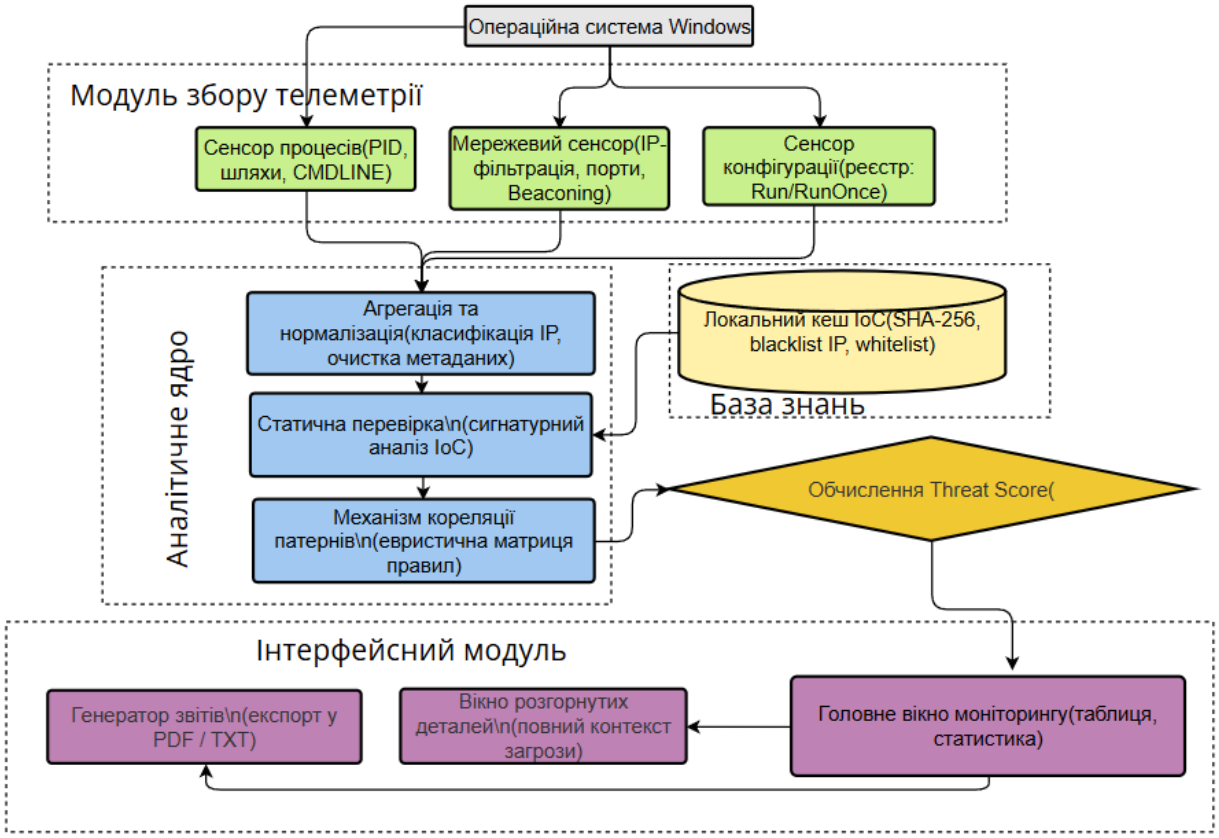


Рис. 2.1 –Структурна схема розробленої системи виявлення RAT

Підсистема мережевого моніторингу аналізує метадані сокетів через системний API та відстежує поточні стани мережевих з'єднань. Стан LISTEN може свідчити про роботу класичного Bind RAT, який очікує підключення

зловмисника на певному порту. У свою чергу, стан ESTABLISHED характерний для Reverse RAT, що самостійно встановлює з'єднання з командно-контрольним сервером і таким чином обходить стандартні міжмережеві екрани. Аналіз цих станів дозволяє пов'язувати мережеву активність із конкретними процесами та враховувати її під час розрахунку рівня загрози.

Для забезпечення автономної роботи в ізольованих мережах система містить модуль локальної бази знань. Він працює з текстовими файлами, у яких зберігаються криптографічні хеші відомих загроз та адреси шкідливих хостів. Використання локальних баз дозволяє уникнути затримок, пов'язаних із запитами до хмарних антивірусних сервісів, а також мінімізує ризик витоку телеметрії про внутрішню інфраструктуру. Завдяки цьому розроблений прототип може використовуватися в малому бізнесі та установах із підвищеними вимогами до конфіденційності даних.

2.3 Розробка алгоритмів збору та аналізу системної інформації

Розробка надійної алгоритмічної основи для збору та аналізу системної інформації є одним із основних етапів створення системи. Саме на цьому рівні формується здатність програмного засобу виявляти приховану активність RAT. Відповідно до обраної гібридної моделі виявлення загроз, алгоритм збору телеметрії працює за принципом ітеративного сканування. Запуск сканування в окремому асинхронному потоці дозволяє швидко отримувати актуальний зріз стану операційної системи та виявляти активні або закріплені у системі загрози.

Початковий етап алгоритму передбачає аналіз активних процесів ОС Windows. Завдяки використанню бібліотеки psutil система автоматично збирає та нормалізує основні атрибути кожного процесу: PID, шлях до виконуваного файлу та аргументи командного рядка. Основна цінність розробленого підходу полягає у глибокому аналізі контексту запуску процесів і перевірці їхньої поведінки.

Алгоритм виявляє нетипові сценарії виконання процесів. Наприклад, якщо

системний процес, такий як svchost.exe або explorer.exe, запускається поза стандартними системними директоріями System32 або Windows, система розцінює це як спробу маскування процесу та додає +45 балів ризику. Окремо аналізується командний рядок процесу. Використання параметра -enc у PowerShell або команд типу downloadstring визначається як ознака прихованого запуску шкідливого Payload і додає +60 балів до Threat Score [30]. Також враховується фонове виконання команд через cmd.exe /c, що збільшує рівень ризику ще на +25 балів.

Наступним етапом є моніторинг механізмів персистентності. Для цього система використовує модуль winreg та виконує аналіз ключів автозавантаження у гілці HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run. Програма отримує список файлів, доданих до автозапуску, та порівнює ці дані з інформацією про активні процеси. Якщо процес одночасно запускається з тимчасової директорії та присутній у реєстрі автозавантаження, він позначається як підозрілий за патерном Persistence, що суттєво підвищує його Threat Score.

Статичний аналіз реалізується шляхом обчислення SHA-256 хешів виконуваних файлів. Отримані значення одразу порівнюються з локальною базою індикаторів компрометації. Такий підхід дозволяє швидко виявляти відомі сімейства RAT і виконує роль першого рівня захисту. Якщо хеш відсутній у базі, система переходить до поведінкового аналізу процесу.

Оскільки RAT потребують постійного зв’язку з командно-контрольним сервером, важливою частиною алгоритму є моніторинг мережевої активності. За допомогою psutil.net_connections система аналізує активні сокети зі статусами ESTABLISHED та LISTEN.

Розроблений алгоритм мережевого аналізу включає:

– фільтрацію локального трафіку;
– перевірку C2-адрес;
– кореляцію портів;
– виявлення Beaconing-активності.

Поєднання даних про запуск процесу, його розташування, аргументи

командного рядка та мережеву активність дозволяє формувати багатовимірний набір ознак. Саме комплексний аналіз різних параметрів дає змогу системі класифікувати програмне забезпечення як шкідливе навіть у випадках використання крипторів або пакувальників. Це є однією з головних переваг розробленої системи порівняно з класичними антивірусами.

Для наочного пояснення процесу кореляції даних та формування Threat Score схему логіки евристичного аналізу наведено на рис. 2.2. Вона демонструє, як різні поведінкові аномалії впливають на загальний рівень ризику процесу.

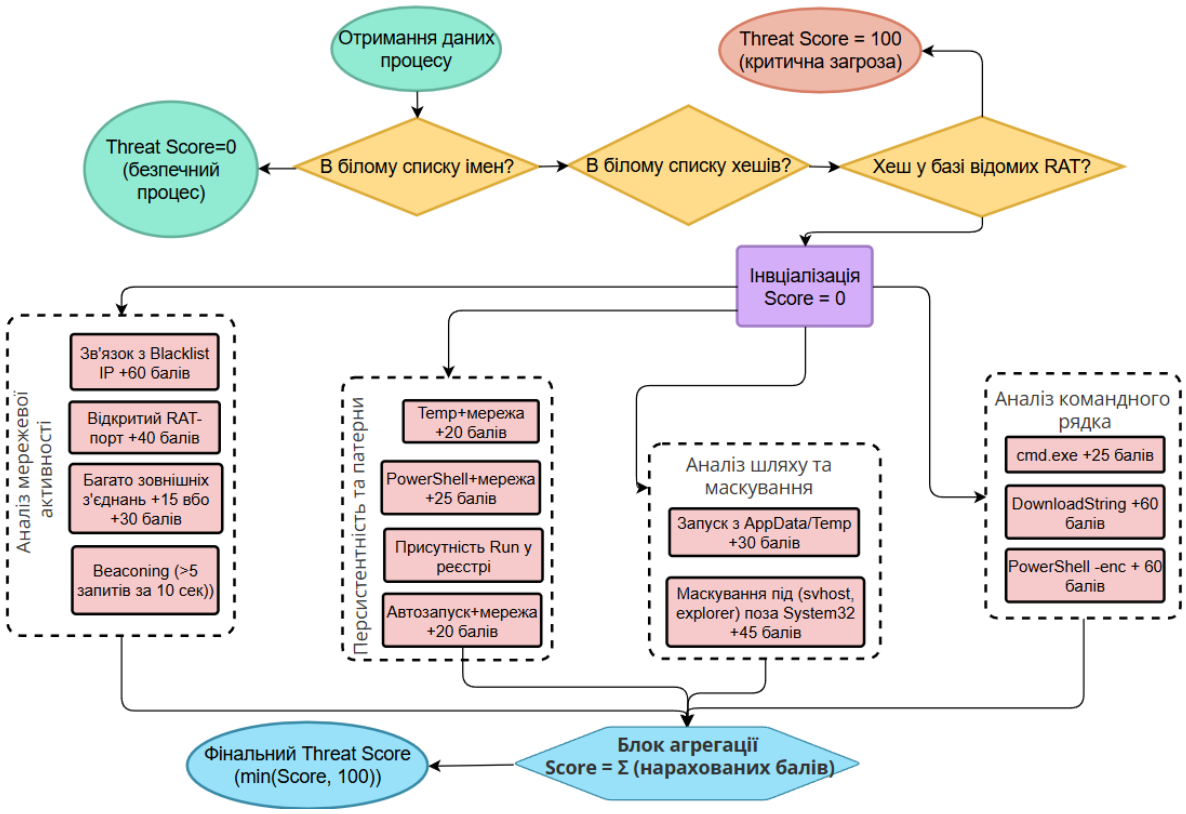


Рис. 2.2 – Схема логіки евристичного аналізу та формування Threat Score

Як показано на рисунку 2.2, процес прийняття рішень є нелінійним. Кожен напрям аналізу, перевірка командного рядка, директорій запуску, мережевої активності та системного реєстру, працює паралельно та генерує власні штрафні бали при виявленні підозрілих ознак. Усі значення передаються до центрального модуля агрегації, де об’єднуються в єдиний показник ризику.

Оскільки сучасне шкідливе програмне забезпечення може одночасно

демонструвати кілька небезпечних патернів, наприклад запуск із тимчасової папки, відкриття підозрілого порту та використання PowerShell, сумарний рівень ризику може перевищувати 100 балів. Для стандартизації результатів система застосовує функцію обмеження min(score, 100), що дозволяє зберігати Threat Score у діапазоні від 0% до 100%.

На основі фінального значення аналітичне ядро класифікує процес від безпечного до критично небезпечного. Такий підхід забезпечує високу точність виявлення загроз та зменшує кількість хибнопозитивних спрацювань, дозволяючи ефективно відокремлювати реальні індикатори компрометації від звичайних системних подій.

2.4 Алгоритм аналізу мережевої активності

Для забезпечення цілісного розуміння логіки роботи розробленого програмного забезпечення було сформовано узагальнений алгоритм його функціонування. На відміну від специфічних алгоритмів мережевої фільтрації чи евристичного скорингу, загальний алгоритм описує повний життєвий цикл програми від моменту ініціалізації до експорту результатів розслідування.

Робота системи розпочинається з етапу ініціалізації, під час якого програма звертається до файлової системи для завантаження локальних баз знань. Відбувається читання списків виключень, бази хешів відомих загроз (known_rat_hashes.txt) та чорного списку IP-адрес (malicious_ips.txt). У разі відсутності цих файлів система автоматично створює їх базові шаблони, що забезпечує стійкість програми до помилок конфігурації.

Після успішної ініціалізації запускається графічний інтерфейс користувача, і програма переходить у режим очікування команди на початок аналізу. При ініціюванні перевірки користувачем, система формує ізольований фоновий потік, щоб не блокувати роботу інтерфейсу під час виконання аналітичних операцій.

Першим кроком фази збору даних є зняття глобальної телеметрії: програма

сканує гілки системного реєстру Windows для формування масиву шляхів автозавантаження та опитує системний API для отримання зрізу активних мережевих з'єднань. Далі алгоритм ітерує за всіма активними процесами. Для кожного об'єкта виконується первинна фільтрація за «Білим списком». Об’єкти, що не пройшли фільтрацію, піддаються наступному аналізу: система вилучає PID, аргументи командного рядка та обчислює SHA-256 хеш. Якщо статичний сигнатурний аналіз не виявив збігів у базі IoC, метадані передаються до евристичного ядра для розрахунку Threat Score.

Графічна інтерпретація описаного узагальненого алгоритму представлена на рисунку 2.3.

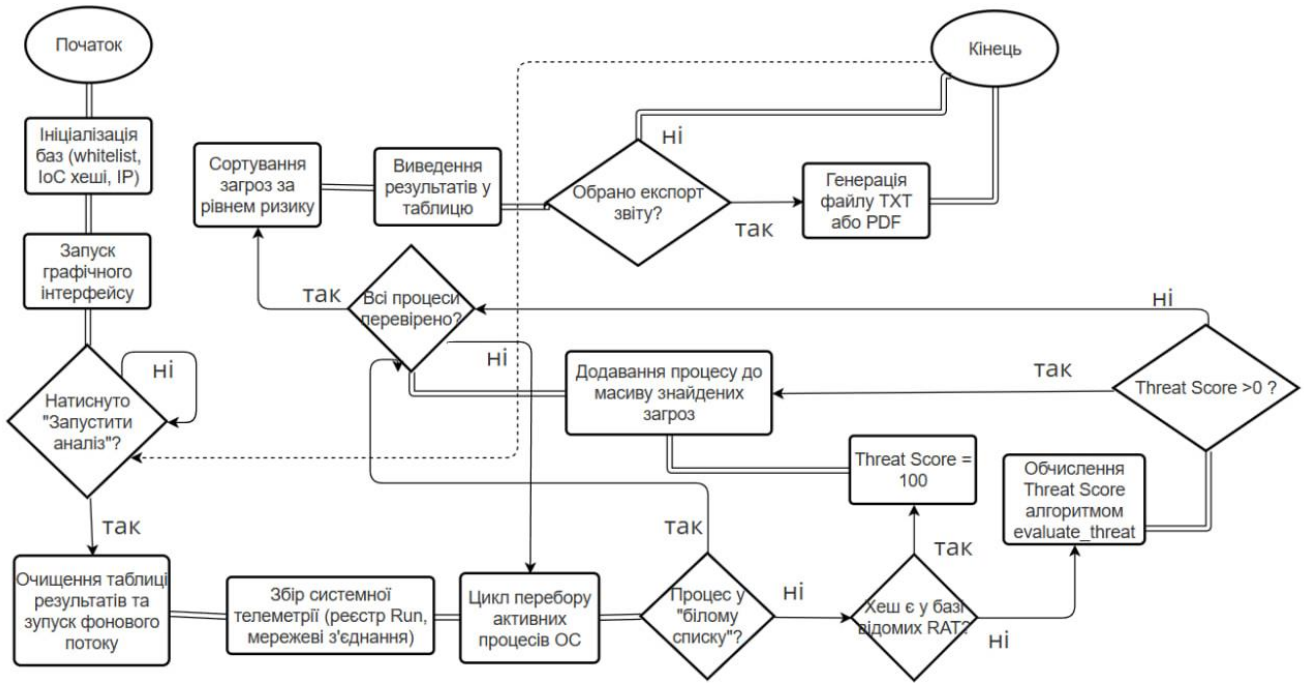


Рис. 2.3 – Узагальнена блок-схема алгоритму роботи системи виявлення RAT

Алгоритм мережевого моніторингу є одним із найважливіших аналітичних компонентів, оскільки життєвий цикл RAT фундаментально залежить від зв'язку з C2-інфраструктурою. Використовуючи метод net_connections(kind='inet') бібліотеки psutil, програмний засіб здійснює перехоплення метаданих сокетів зі статусами ESTABLISHED та LISTEN. Важливою складовою є застосування

модуля ipaddress для фільтрації локального трафіку через функцію is_external_ip, що дозволяє фокусуватися виключно на зовнішніх інтернет-адресах.

Особлива увага приділяється аналізу портової активності та таймінгів. Алгоритм звіряє порти з масивом SUSPICIOUS_PORTS та реалізує механізм виявлення Beaconing-активності через структуру connection_history. Якщо процес ініціює понад 5 зовнішніх підключень у 10-секундному вікні, система автоматично ідентифікує це як аномальний патерн зв'язку з сервером зловмисника. Логіка обробки мережевих метаданих представлена на рисунку 2.4.

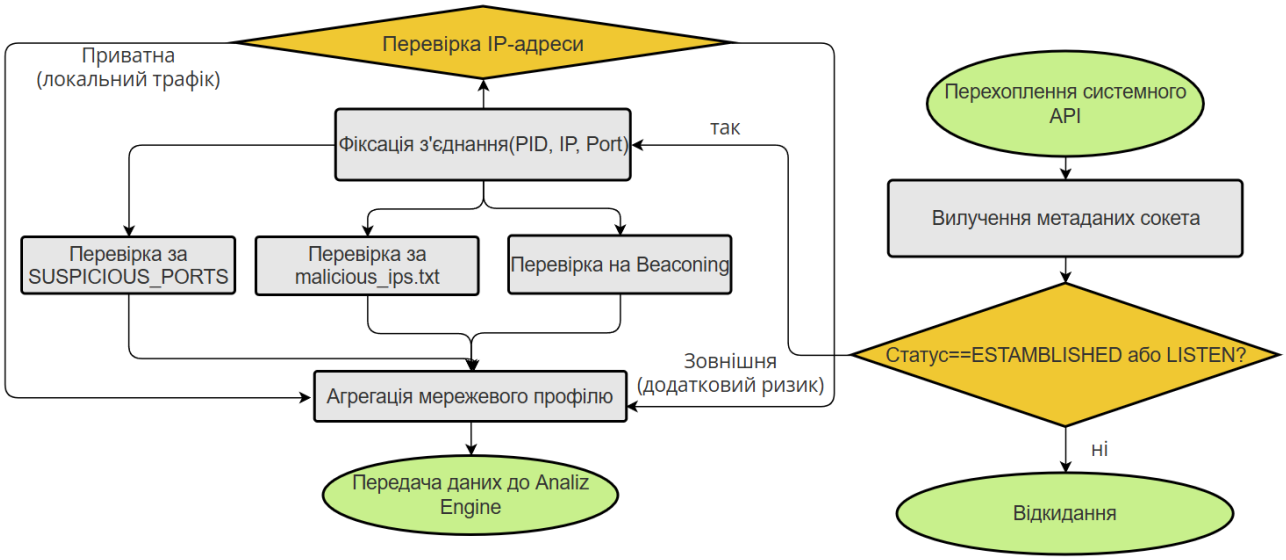


Рис. 2.4 – Блок-схема алгоритму мережевого моніторингу та фільтрації C2-трафіку

Як демонструє блок-схема мережевого моніторингу, кінцевий результат аналізу сокетів безпосередньо інтегрується в загальний показник Threat Score. Такий підхід дозволяє виявляти складні кореляційні патерни, наприклад, коли процес, запущений із тимчасової директорії Temp, одночасно намагається встановити з'єднання через підозрілий порт 5552.

Розробка підсистеми прийняття рішень логічно завершує етап проєктування. Центральний компонент агрегує отримані коефіцієнти у фінальний бал ризику. Оскільки сумарна кількість штрафних балів може перевищувати 100,

алгоритм застосовує функцію обмеження min(score, 100), що забезпечує стабільність виводу. Автоматизоване рішення про маркування процесу як «Підозрілого» або «Критичної загрози» базується на накопиченні критичної маси доказів, а не на одному тригері, що значно підвищує точність детектування у порівнянні зі стандартними антивірусами.

2.5 Розробка підсистеми прийняття рішень

Розробка підсистеми прийняття рішень є концептуальним апексом архітектурного проєктування системи захисту, оскільки саме цей аналітичний модуль відповідає за конвертацію розрізнених «сирих» телеметричних даних у конкретний, математично обґрунтований вердикт щодо рівня загрози операційній системі.

Комплексний алгоритм прийняття рішення щодо класифікації процесу та відповідної реакції системи, що базується на обчисленому показнику Threat Score та перевірці локальних баз індикаторів компрометації, наочно представлено на рисунку 2.5:

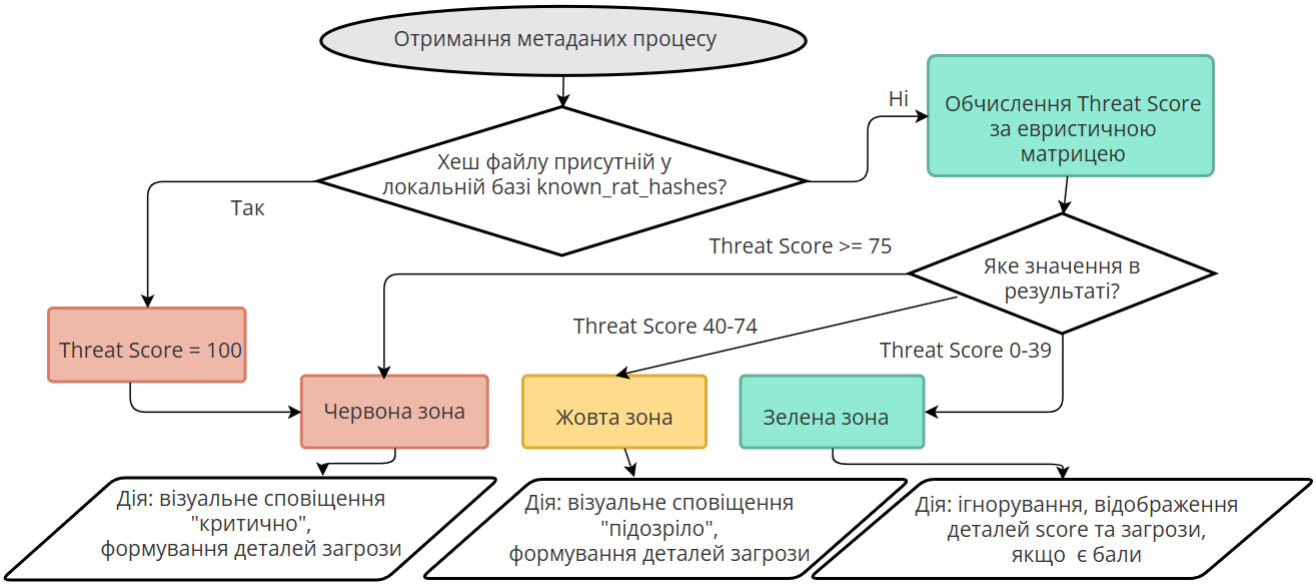


Рис. 2.5 – Схема прийняття рішення щодо наявності RAT та реагування системи

Алгоритм функціонування даної підсистеми базується на парадигмі агрегації телеметрії у єдиний багатовимірний вектор ознак для кожного активного процесу в середовищі Windows. Процес автоматизованої оцінки розпочинається з ініціалізації базового балу загрози, який динамічно коригується та акумулюється залежно від результатів роботи локальних сенсорів. Перевагою такого підходу є використання евристичного аналізу. Це мінімізує недоліки суто сигнатурного детектування.

Згідно з правилами підрахунку фінальних балів, вагові коефіцієнти в системі розподіляються між ізольованими факторами та їхніми комбінаціями. Менші, але статистично суттєві бали ризику нараховуються за окремі ознаки аномальної поведінки. До таких відносяться:

– запуск виконуваного файлу з користувацьких директорій %AppData%, %Temp%;

– фонове виконання командних скриптів;

– використання техніки маскування, коли процес намагається імітувати системні назви на кшталт svchost.exe, lsass.exe чи explorer.exe, але його виконуваний файл знаходиться поза системною директорією System32;

– наявність поодиноких з'єднань із зовнішніми публічними IP-адресами.

Проте, найвищі штрафні коефіцієнти нараховуються за комплексні патерни, які описують класичний життєвий цикл бекдору: закріплення в системі, встановлення C2-каналу та приховане виконання Payload. В основі цієї логіки лежить механізм багатофакторної кореляції. З математичної точки зору, система розглядає кожен процес як вектор стану з багатьма вимірами. Якщо аномалія фіксується лише в одному вимірі, ризик зростає лінійно, але якщо система фіксує перетин аномалій у кількох вимірах одночасно, то система активує додаткові правила кореляції. У коді розробленого прототипу це реалізовано через умовні конструкції агрегації (наприклад, перевірка прапорців in_autorun та has_net), які додають спеціальні бали за комбінований патерн. Це наближає розроблену логіку до роботи комерційних SIEM-систем, які шукають не події, а ланцюжки подій.

Математична модель прийняття остаточного рішення реалізується через

розподіл результататів аналізу на три зони ризику. Розрахований показник завжди обмежується верхньою межею за допомогою функції min(score, 100). Зони розподіляються наступним чином:

- «Зелена зона»;
- «Жовта зона»;
- «Червона зона».

«Зелена зона» охоплює процеси з показником ризику 0-39. Виявлені незначні відхилення у цій категорії не розглядаються як загроза. Деталі чому їм нараховано бали, якшо їх більше 1, відображається у фінальній таблиці звіту сканування.

«Жовта зона» представляє проміжний рівень ризику з загольною оцінкою 40-74 бали. У ній процеси класифікуються як підозрілі. Такі випадки можуть включати легітимні програми, що виконуються в нетипових умовах, наприклад запуск із тимчасових директорій або прояв нетипової мережевої активності. Процеси цієї категорії підлягають обов’язковій реєстрації у журналі подій та відображаються в інтерфейсі для подальшого аналізу.

«Червона зона» відповідає високому рівню ризику ≥ 75 балів та свідчить про наявність поведінкових або сигнатурних ознак, характерних для шкідливого програмного забезпечення, зокрема RAT. Досягнення цього порогового значення, позначає процес критиичним, а також формується детальний звіт із зазначенням причин детектування.

Також є окремий сценарій виявлення хешу SHA-256 процесу в локальній базі відомих індикаторів компрометації. У такому випадку застосовується правило "миттєвого детектування", що безпосередньо присвоює процесу максимальний бал та ігнорує подальший поведінковий розрахунок.

Головним досягненням реалізації такої моделі оцінки є забезпечення автономності прийняття рішень через класифікацію загроз без потреби у зв’язку з хмарними системами.

3 РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОЇ СИСТЕМИ

3.1 Розробка прототипу програмного засобу

Програмний засіб розроблено на мові програмування Python 3.12. Вибір саме цієї версії обґрунтований оптимізованою роботою інтерпретатора, покращеною обробкою винятків та наявністю системи бібліотек для взаємодії з Windows API.

Основним компонентом створеного програмного забезпечення є агент моніторингу, керований через графічний інтерфейс. Для забезпечення безперервного збору телеметрії без зависання графічної оболонки, архітектуру побудовано на базі асинхронного виконання.

Збір системних метаданих базується на інтеграції комплексу спеціалізованих бібліотек. Головним рушієм збору виступає модуль psutil. У коді реалізовано функцію циклічного сканування, яка ітерує системну таблицю активних задач, вилучаючи ідентифікатор, шлях до виконуваного файлу, а також аргументи командного рядка. Це необхідне стратегічне архітектурне рішення з огляду на те, що більшість сучасних RAT активно використовують техніки безфайлових атак, передаючи шкідливі інструкції безпосередньо в оперативну пам'ять системним утилітам без створення підозрілих файлів на жорсткому диску, і саме аналіз командного рядка дозволяє це виявити.

Для забезпечення автономності роботи програмного засобу в архітектуру інтегровано функцію init_files(), яка автоматично ініціалізує локальні конфігураційні бази даних під час першого запуску системи. У випадку відсутності службових файлів known_rat_hashes_file або malicious_ips_file програма автоматично створює їх та записує базові тестові індикатори компрометації. Такий підхід забезпечує коректне функціонування аналітичного ядра навіть у «чистому» середовищі без попередньої конфігурації та усуває ризик аварійного завершення роботи через відсутність необхідних службових структур.

Паралельно реалізовано модуль аналізу реєстру на базі бібліотеки winreg. Програмний код здійснює парсинг ключів автозавантаження для виявлення спроб

шкідливого коду закріпитися в системі. Увага алгоритму фокусується на гілці HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run, оскільки запис у цей розділ не вимагає від зловмисника підвищених адміністративних привілеїв, що робить його найпопулярнішим вектором для досягнення прихованої персистентності. Наступний фрагмент коду демонструє реалізацію збору даних автозавантаження та активних мережевих сокетів:

```
for conn in psutil.net_connections(kind='inet'):
    if conn.pid and conn.status in ['ESTABLISHED', 'LISTEN']:
        port = conn.raddr.port if conn.raddr else conn.laddr.port
        ip = conn.raddr.ip if conn.raddr else None
        net.setdefault(conn.pid, []).append({'port': port, 'ip': ip})
        connection_history.setdefault(conn.pid, []).append(now)
        connection_history[conn.pid] = [t for t in connection_history[conn.pid] if now - t < 10]
key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run")
for i in range(winreg.QueryInfoKey(key)[1]):
    autorun.append(winreg.EnumValue(key, i)[1].lower())
```

Важливою оптимізаційною перевагою прототипу є механізм фільтрації локального трафіку. Для того щоб система не реагувала на легітимні звернення до локальних принтерів чи роутерів, у коді реалізовано функцію is_external_ip, яка за допомогою бібліотеки ipaddress відсіює приватні адреси, дозволяючи аналітичному ядру фокусуватися виключно на публічних з'єднаннях, що продемонстровано нижче:

```
def is_external_ip(ip):
    try:
        return not ipaddress.ip_address(ip).is_private
    except:
        return False
```

Дана функція автоматично перевіряє належність IP-адреси до діапазонів, зарезервованих стандартом RFC 1918 (наприклад, 192.168.x.x). Таке інженерне рішення дозволило знизити обчислювальне навантаження на евристичне ядро, що дозволило суттєво знизити обсяг аналізованого мережевого шуму.

Усі зібрані програмними модулями дані направляються до головного обчислювального компонента – евристичного ядра, реалізованого у функції

evaluate_threat. Саме тут розраховується підсумковий бал загрози Threat Score для кожного окремого процесу. Окремим рівнем захисту в архітектурі прототипу виступає механізм hash-based detection, реалізований через криптографічне хешування виконуваних файлів алгоритмом SHA-256. Для кожного активного процесу функція get_file_hash() виконує поблочне зчитування вмісту файлу та формує унікальний цифровий відбиток, який використовується для порівняння з локальною базою відомих індикаторів компрометації KNOWN_RAT_HASHES_FILE. У випадку повного збігу хешу процес миттєво отримує максимальний рівень небезпеки без подальшого евристичного аналізу. Такий підхід забезпечує гарантоване виявлення вже відомих RAT-зразків та мінімізує ймовірність обходу системи через зміну назви виконуваного файлу. Одночасно реалізовано механізм HASH_WHITELIST_FILE, який дозволяє повністю виключати перевірені легітимні файли зі сканування незалежно від їх поточного імені або директорії розташування.

Система виконує глибокий крос-контекстний аналіз: перевіряє факт маскування під системні процеси, звіряє мережеві порти з масивом SUSPICIOUS_PORTS та аналізує командний рядок на наявність обфускованих інструкцій PowerShell. Унікальність розробленого ядра полягає у синергетичному підході до оцінювання: одинична аномалія може дати лише 15-20 балів, але комбінація факторів активує додаткові штрафні мультиплікатори. Наприклад, запуск з папки Temp одночасно зі встановленням зовнішнього з'єднання. При цьому фінальний результат завжди математично нормалізується функцією min(score, 100) для збереження єдиної та зрозумілої шкали ризику. Наступний фрагмент коду демонструє частину логіки скорингу:

```
if "powershell" in cmd and "-enc" in cmd:
    score += 60
    reasons.append("Прихований запуск PowerShell (Encoded)")
base_name = name.replace(".exe", "")
if base_name in ["svchost", "explorer", "lsass", "csrss", "winlogon"]:
    if "system32" not in path and "windows" not in path:
        score += 45
        reasons.append("Маскування під системний процес")
```

Фактично реалізоване евристичне ядро працює як повноцінний модуль поведінкової кореляції подій. На відміну від класичних сигнатурних антивірусів, які аналізують окремі ознаки загроз, система досліджує взаємозв'язки між подіями в часі та їхній контекст. Деякі дії, наприклад запуск файлів із директорії Temp або відкриття мережевих з'єднань, можуть бути цілком легітимними (як у випадку інсталяторів або месенджерів). Тому такі події отримують лише невеликі штрафні значення, щоб зменшити кількість хибних спрацювань.

Основна цінність алгоритму полягає у перевірці комбінацій різних подій. Система пов'язує мережеву активність з іншими потенційно небезпечними ознаками та оцінює їх у сукупності. У підході реалізовано кілька ключових правил підвищення ризику:

– Execution + Network (in_temp and has_net);

– Persistence + Network (in_autorun and has_net);

– Scripting + Network ("powershell" in cmd and has_net).

Execution + Network (in_temp and has_net) означає запуск процесу з тимчасової директорії одночасно з активним мережевим підключенням, що може свідчити про підозрілу активність.

Persistence + Network (in_autorun and has_net) фіксує ситуацію, коли процес присутній у автозавантаженні та паралельно взаємодіє з мережею, що характерно для закріплення шкідливого ПЗ.

Scripting + Network ("powershell" in cmd and has_net) визначає використання PowerShell або інших інтерпретаторів разом із мережевою активністю, що часто зустрічається у сценаріях C2-комунікації та безфайлових атак.

Кожна з таких комбінацій додає додаткові штрафні бали до загального показника Threat Score. Завдяки цій кореляції система може виявляти навіть складні багатокомпонентні RAT-атаки, водночас ігноруючи звичайну безпечну поведінку системи.

Для взаємодії з оператором безпеки розроблено сучасний графічний інтерфейс за допомогою бібліотеки customtkinter. Використання цієї надбудови дозволило реалізувати гнучку темну тему оформлення, що суттєво знижує зорове

навантаження на аналітиків під час тривалого моніторингу та повністю відповідає сучасним стандартам UX/UI дизайну. Інтерфейс виводить статистику просканованих об'єктів та динамічну таблицю виявлених загроз. Це відображається з кольоровою розбивкою на критичні загрози, підозрілу активність та загальну кількість. Загальний вигляд головного вікна розробленого прототипу під час сканування системи представлено на рисунку 3.1.

Для підвищення ефективності роботи оператора у програмі реалізовано механізм автоматичного сортування результатів аналізу за рівнем небезпеки. Після завершення сканування усі виявлені процеси тимчасово накопичуються у структурі found_processes, після чого сортуються за спаданням Threat Score за допомогою функції sort(key=lambda x: x[0], reverse=True). Це дозволяє системі відображати найбільш небезпечні процеси у верхній частині таблиці, забезпечуючи пріоритетний аналіз критичних інцидентів та мінімізуючи час реагування адміністратора на потенційні атаки.

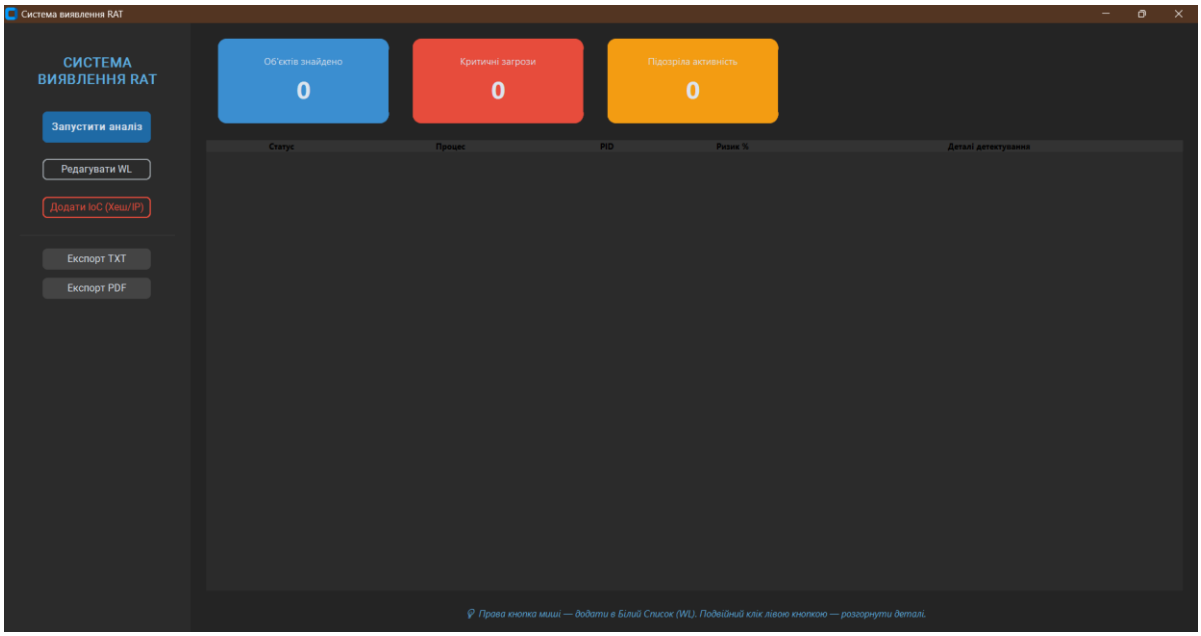


Рис. 3.1 – Графічний інтерфейс розробленої системи виявлення RAT

Для забезпечення високого рівня деталізації інцидентів, у програмі реалізовано інтерактивний механізм: при подвійному натисканні на підозрілий процес відкривається вікно деталей. У ньому оператору надається вичерпний

структурований перелік усіх знайдених аномалій та патернів, які сформували фінальний Threat Score. Така прозорість роботи алгоритму є критично важливою вимогою для систем класу EDR, оскільки вона дозволяє адміністратору чітко зрозуміти логіку спрацювання захисту та уникнути помилкового блокування легітимного робочого програмного забезпечення. Як виглядає це вікно зображено на рис. 3.2.

[

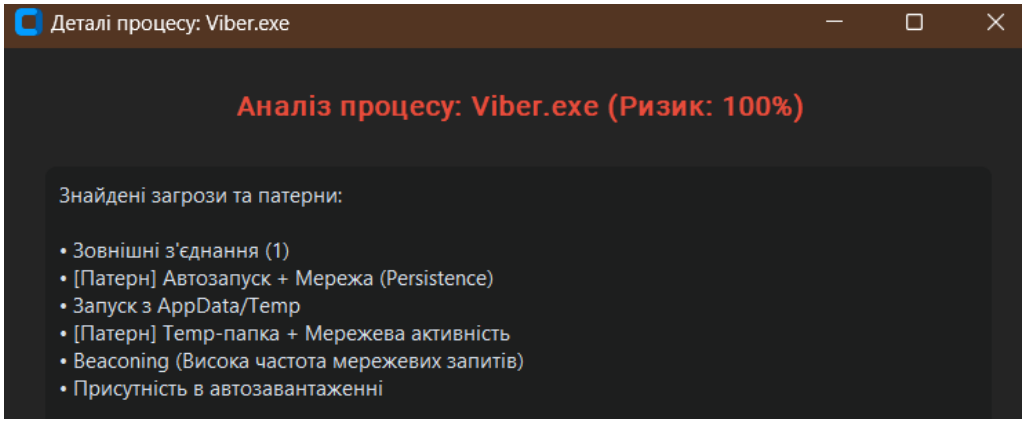


Рис. 3.2 – Візуалізація розгорнутого контексту виявленої загрози

Застосований підхід до програмування повною мірою забезпечує виконання вимог технічного завдання. Розроблений прототип демонструє високий рівень автономності, дозволяє експортувати результати розслідування у формати PDF/TXT та підтверджує свою легковаговість, забезпечуючи високу швидкість сканування без використання хмарних баз даних.

Окрім базового моніторингу, графічний інтерфейс системи містить розширений інструментарій для управління локальними базами знань та генерації звітності. На бічній панелі керування розміщено кнопки для взаємодії з «білими списками» та локальними індикаторами компрометації. Оператор має змогу через спеціальне діалогове вікно в режимі реального часу додавати підозрілі хеші або IP-адреси до списку блокування. Це зображено на рис. 3.3. Для запобігання випадковому пошкодженню конфігураційних баз через людський фактор, у логіку діалогового вікна вбудовано перевірку на основі регулярних виразів. Цей механізм суворо контролює формат введених даних, вимагаючи точної

відповідності 64-символьному стандарту SHA-256 або правильному синтаксису IPv4.Також редагувати перелік довірених додатків у вікні керування базами, що видно на рис. 3.4.

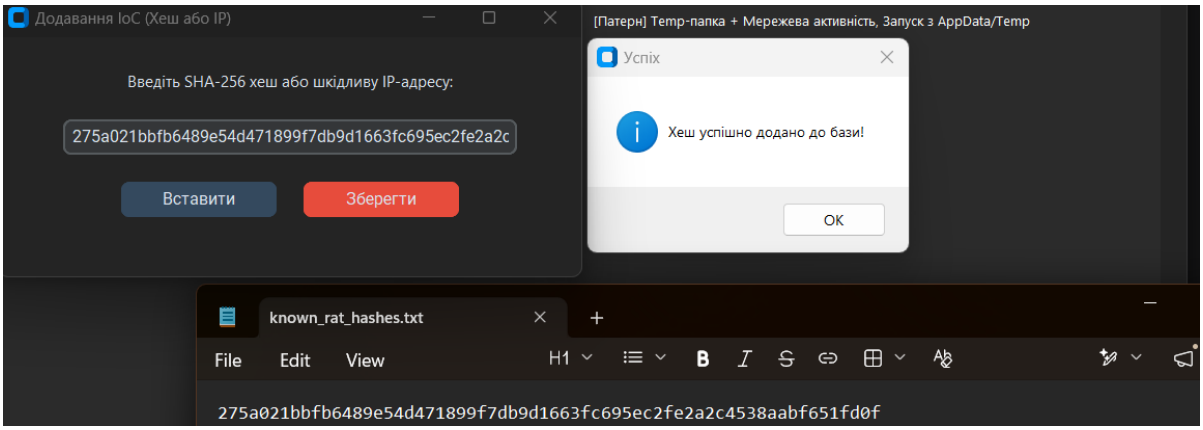


Рис. 3.3 – Вікно ручного додавання індикаторів компрометації

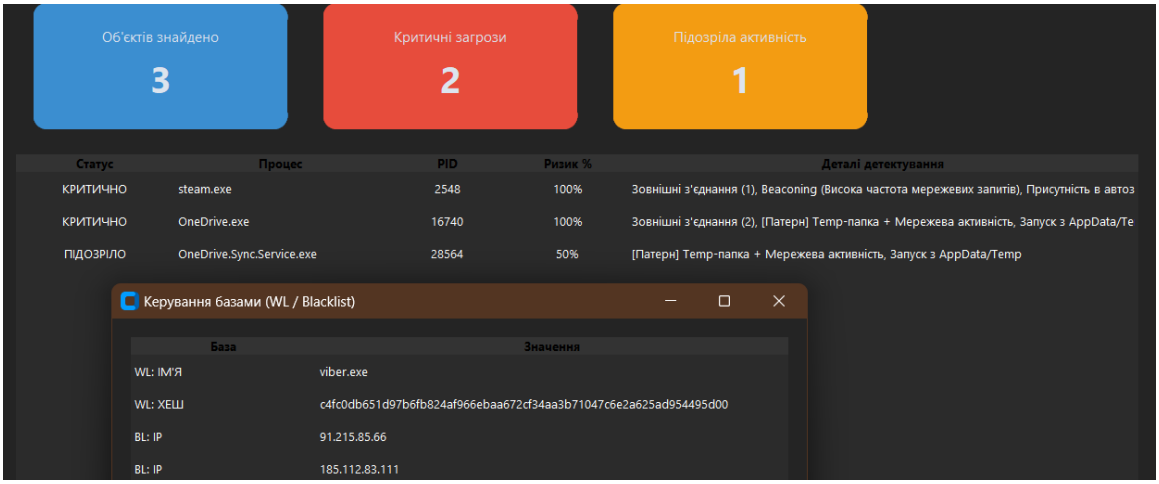


Рис. 3.4 – Інтерфейс управління локальними базами довірених об'єктів

Для оптимізації роботи реалізовано інтерактивне контекстне меню, яке викликається натисканням правої кнопки миші по обраному рядку таблиці виявлених загроз, що можна переглянути на рис. 3.5. Воно дозволяє миттєво додати безпечний процес або обчислений хеш його виконуваного файлу до довірених, тим самим виключаючи його з подальшого сканування та мінімізуючи рівень хибнопозитивних спрацювань без необхідності ручного втручання адміністратора у текстові файли конфігурації.

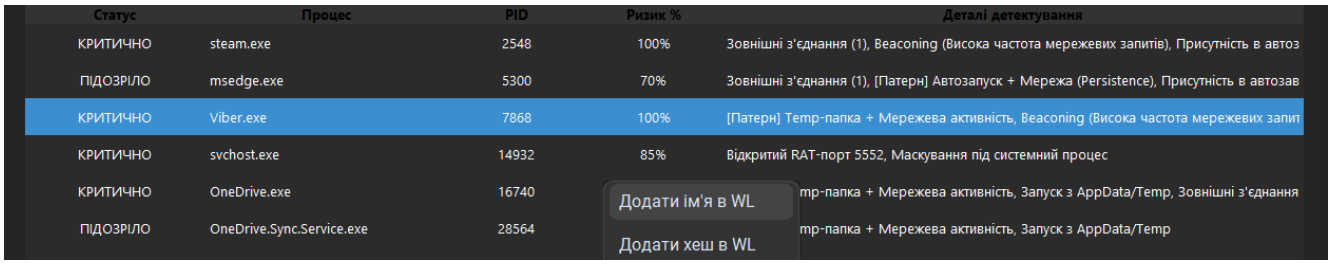


Рис. 3.5 – Контекстне меню швидкого додавання процесу до списку виключень

Також на рисунку 3.5 продомонстровано, що процес програми Viber більше не підсвічується критичним. Важливим архітектурним рішенням є впровадження двох незалежних механізмів виключень: за іменем процесу та за криптографічним хешем виконуваного файлу. Такий підхід дозволяє більш точно ідентифікувати довірені програми та зменшити ризик помилкових спрацювань, оскільки враховуються як поверхневі ознаки процесу, так і його фактична унікальна ідентифікація. Наявність двох окремих баз зумовлена необхідністю балансування між зручністю адміністрування та суворим рівнем безпеки. «Білий список імен» є зручним для легітимного прикладного програмного забезпечення, що часто оновлюється, оскільки позбавляє оператора необхідності оновлювати базу після кожного офіційного патчу. Проте цей метод має теоретичну вразливість до атак типу Process Spoofing, коли шкідливе ПЗ приховується під довіреним ім'ям. Для нівелювання цього ризику та захисту критично важливих системних файлів у систему інтегровано «Білий список хешів». Цей механізм значно підвищує точність ідентифікації легітимного об'єкта незалежно від його поточної назви чи директорії розташування, хоча і вимагає повторного додавання хешу у разі виходу нової версії програми. Такий дворівневий підхід надає адміністратору максимальну гнучкість у налаштуванні політик безпеки кінцевої точки.

Збереження результатів розслідування інцидентів забезпечується вбудованим модулем експорту, який автоматично генерує деталізовані звіти про поточне сканування у форматах TXT та PDF із залученням спеціалізованої бібліотеки reportlab. Згенеровані документи містять хронологічну послідовність виявлених інцидентів із зазначенням точного часу фіксації, PID ідентифікатора

процесу та вичерпного переліку спрацьованих евристичних правил. Це дозволяє легко інтегрувати результати роботи розробленого агента у корпоративні системи управління інцидентами, використовувати їх як доказову базу під час цифрової розслідки та звітувати про виявлені аномалії.

3.2 Проведення експеременту

Завершальним етапом пророблено роботи є комплексна інтерпретація та оцінка достовірності експериментальних даних, отриманих під час тестування прототипу в середовищі Windows. Результати апробації підтвердили високу наукову та практичну ефективність розробленої гібридної, сигнатурно-евристичної, моделі детектування. Експеримент проводився на реальній операційній системі з активним мережевим підключенням, що дозволило перевірити здатність агента аналізувати живий трафік та процеси у динаміці, а не в ізольованому середовищі пісочниці.

Під час проведення експерименту було запущено розроблений скрипт-імітатор шкідливої поведінки dummy_rat.py на фоні роботи легітимних фонових додатків. Аналітичному ядру системи вдалося успішно ідентифікувати імітатор RAT, коректно маркувавши його як критичну загрозу. Система нарахувала імітатору сумарний Threat Score понад 85 балів. Імітатор RAT симулював часткову persistence-поведінку через копіювання виконуваного файлу до директорії AppData, однак не модифікував ключі автозавантаження реєстру Windows. Такий результат був досягнутий завдяки спрацюванню двох ключових евристичних патернів:

– факт маскування виконуваного файлу під системну службу svchost;

– відкриття нестандартного RAT-порту 5552 зі статусом LISTEN для прийому вхідних підключень.

Наступним розглянемо такий реалізований механізм, як LISTEN detection. Під час аналізу мережевих сокетів система перевіряє не лише факт наявності TCP-

з'єднання, а і його поточний стан. У функції scan() здійснюється фільтрація об'єктів psutil.net_connections(kind='inet') за статусами ESTABLISHED та LISTEN. Саме стан LISTEN є критично важливим індикатором для RAT-загроз, оскільки свідчить про готовність процесу приймати вхідні керуючі підключення від C2-сервера або оператора атаки. У проведеному експерименті dummy_rat успішно відкривав порт 5552 у режимі прослуховування, що стало одним із ключових факторів нарахування критичного Threat Score.

Згідно з імплементованою логікою функції evaluate_threat, маскування імені процесу без його фізичного розміщення у легітимній системній папці system32 генерує 45 балів ризику, а виявлення відкритого порту з масиву SUSPICIOUS_PORTS додає ще 40 балів. Їх сума (85 балів) автоматично переводить процес у статус «КРИТИЧНО» (оскільки Score >= 75), що переконливо демонструє здатність математичної моделі безпомилково виявляти загрози лише за поведінковими маркерами, навіть без наявності їхнього криптографічного хешу в базі індикаторів компрометації.

Результати цього можна переглянути на рис. 3.6:

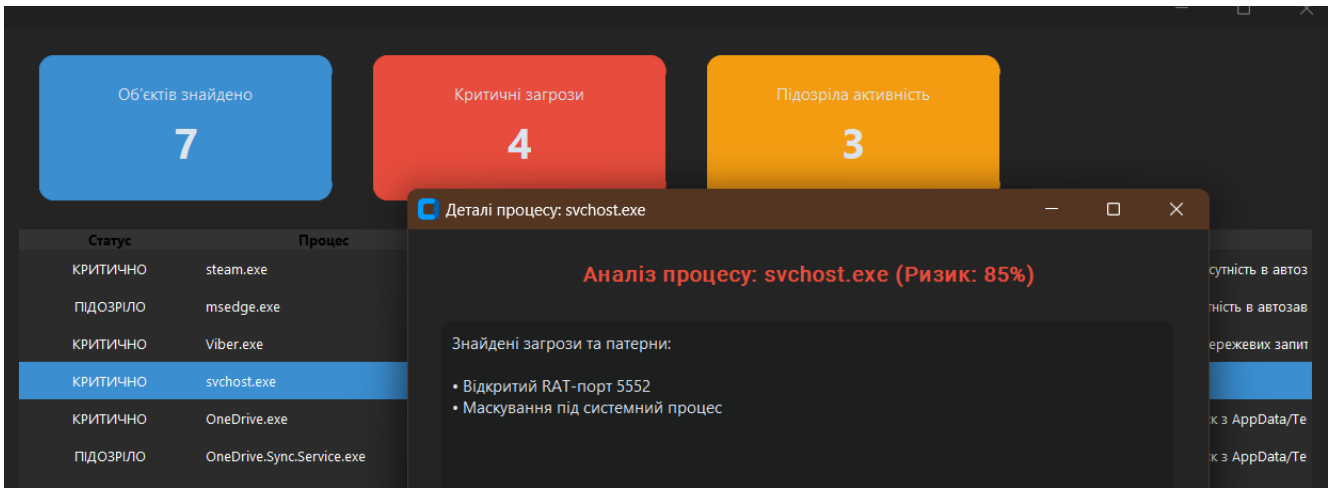


Рис. 3.6 – Результат експерименту

Одночасно з цим, розроблений алгоритм успішно диференціював активність легітимних додатків, не допустивши критичних хибнопозитивних спрацювань. Наприклад, месенджер Viber, який активно підтримував зовнішні з'єднання та був

встановлений у директорії AppData, отримав статус «Критичого» процесу. Це підтвердило життєздатність розробленої буферної «червоної зони», яка дозволяє оператору звернути увагу на активність процесу, не блокуючи його автоматично. Легітимний месенджер акумулював бали за запуск із користувацької директорії, наявність зовнішніх з'єднань та високу частоту мережевих запитів, що в сумі дало високий бал. Оскільки це евристичний бал, а не жорсткий збіг за IoC базою, система надає адміністратору розгорнутий звіт причин у графічному інтерфейсі. Це дозволяє проаналізувати контекст і прийняти рішення про додавання безпечної програми до WhiteList у два кліки.

Для формалізації висновків щодо надійності системи було використано метрику точності, яка характеризує здатність програмного засобу мінімізувати кількість помилкових тригерів на безпечне ПЗ. Детальний аналіз логів системи дозволив встановити, що найвагоміший внесок у зниження рівня False Positives внесла реалізована функція фільтрації мережевого трафіку is_external_ip(). Відкидання локальних IP-адрес дозволило проігнорувати безпечний інформаційний шум, зокрема системні транзитні пакети та запити до локального шлюзу. Цей метод, що використовує перевірку ipaddress.ip_address(ip).is_private, ефективно ігнорує широкомовні запити локальної мережі, які постійно генерує більшість легітимних служб Windows. У результаті алгоритм скорингу не перевантажується зайвими даними та аналізує лише ті процеси, що намагаються встановити маршрутизацію з глобальною мережею Інтернет.

Порівняно з традиційними сигнатурними антивірусами, розроблений EDR-агент виявив значно вищу стійкість до варіативності загроз. Завдяки реалізації механізму перевірки аргументів командного рядка, система здатна виявляти окремі поведінкові ознаки безфайлових атак через аналіз аргументів командного рядка. Аналітичне ядро гарантовано ініціює критичний алерт при фіксації спроби виконання закодованих команд PowerShell або завантаження віддалених payload-файлів, що залишається "сліпою зоною" для більшості базових засобів захисту. Зокрема, виявлення в структурі об'єкта cmdline таких специфічних патернів як "-enc" або інструкції "downloadstring" миттєво додає 60 балів ризику. Оскільки

більшість класичних антивірусів сканують виключно файли на жорсткому диску, розроблений прототип, що інспектує параметри запуску процесів безпосередньо в оперативній пам'яті ОС, переконливо доводить свою перевагу у виявленні сучасних LotL-атак.

Окремим вектором вимірювань стала оцінка ресурсомісткості агента. Під час пікового навантаження розроблений на базі psutil прототип споживав не більше 3-15% ресурсів центрального процесора та потребував менше 60 МБ оперативної пам'яті. Можна переглянути на рис. 3.7:

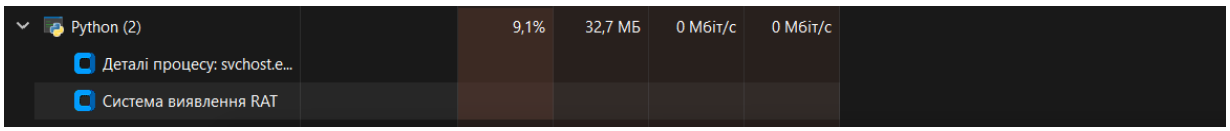


Рис. 3.7 – Навантаження на CPU

Такі показники енергоефективності були досягнуті завдяки селективному збору телеметрії. Метод psutil.process_iter(['pid', 'name', 'exe', 'cmdline']) зчитує лише чотири критично необхідні атрибути замість повного ресурсомісткого дампу пам'яті процесу. Попереднє кешування результатів зчитування реєстру та історії мережевих підключень повністю виключає необхідність повторного звернення до повільного системного API Windows під час кожної ітерації аналітичного циклу.

Отримані дані повністю підтверджують обраний варіант: поєднання збору системної телеметрії з гнучкою багатокритеріальною моделлю Threat Score дозволяє створити ефективний ешелон захисту кінцевої точки. Практична значущість результатів полягає у доведенні можливості побудови автономних засобів безпеки, які здатні забезпечувати високий рівень детектування складних LotL-атак та RAT-агентів при мінімальному навантаженні на обчислювальні ресурси хоста.

3.3 Оцінка достовірності та аналіз отриманих результатів

Задля провірки заапропонованої системи не лише на словах було проведено оцінку достовірності отриманих експериментальних даних. Результати апробації розробленого прототипу підтвердили високу наукову та практичну ефективність запропонованого гібридного підходу до детектування загроз. Завдяки реалізації механізмів багатофакторної кореляції мережевої активності, моніторингу процесів та перевірки цілісності конфігурацій реєстру, системі вдалося успішно ідентифікувати змодельовану RAT-активність, коректно відрізнивши її від штатної роботи легітимних фонових додатків. Функція евристичного аналізу evaluate_threat одночасно обробляє масив параметрів процесу, словник поточних мережевих з'єднань та масив ключів автозавантаження. Таке об'єднання дозволяє системі виявляти складні комбіновані патерни, такі як одночасна наявність процесу в автозапуску та його звернення до зовнішньої мережі ([Патерн] Автозапуск + Мережа), що неможливо ідентифікувати при ізольованому аналізі лише одного параметра. Успішне детектування на фоні вимкнених штатних засобів захисту операційної системи свідчить про спроможність евристичного ядра самостійно виділяти шкідливі ознаки на основі аналізу аномалій поведінки.

Для формалізації висновків щодо надійності системи було використано концептуальний підхід на основі стандартних статистичних метрик, які є загальноприйнятими у сучасних системах виявлення вторгнень. Першою ключовою метрикою виступила точність, яка характеризує здатність програмного засобу диференціювати загрози від легітимної активності та мінімізувати кількість помилкових спрацювань на безпечне програмне забезпечення. Другою метрикою стала повнота, що відображає здатність алгоритму виявляти цільові шкідливі патерни. Оскільки під час тестування прототип не згенерував критичних хибних тривог на такі програми як Viber чи OneDrive і водночас безпомилково ідентифікував імітатор RAT, присвоївши йому критичний рівень небезпеки, можна констатувати високий рівень збалансованості розробленого евристичного алгоритму. Висока точність досягається завдяки гнучкій системі накопичення

балів: легітимний додаток, що працює з папки AppData та використовує мережу, отримає лише часткові штрафні бали (наприклад, 30 балів за директорію та 15 за зовнішні з'єднання), що класифікує його як «ПІДОЗРІЛО», але не блокує автоматично. Додатково, імплементація перевірок за дворівневим білим списком (структури current_wl та current_hwl) гарантує нульовий рівень хибних спрацювань для підтверджено безпечного ПЗ, оскільки перевірка у функції миттєво припиняється (return 0, []) при знаходженні збігу.

Детальний аналіз отриманих результатів дозволив встановити, що найбільш вагомий внесок у процес детектування внесли розроблені правила, спрямовані на виявлення нестандартних мережевих портів, аналіз аргументів командного рядка та ідентифікацію аномальних шляхів запуску виконуваних файлів. Зокрема, кореляція факту маскування процесу під системну службу з наявністю активного TCP-з'єднання зі статусом LISTEN дозволила системі миттєво класифікувати загрозу. У програмному коді ця кореляція реалізована через строгі умовні конструкції: перевірка базового імені на збіг із системними (наприклад, svchost чи explorer) за умови відсутності легітимного шляху ("system32" not in path) генерує 45 балів ризику, а наявність відкритого порту з масиву SUSPICIOUS_PORTS додає ще 40 балів. У сумі це дає 85 балів, що впевнено долає встановлений поріг у 75 балів для отримання статусу «КРИТИЧНО». Порівняно з традиційними сигнатурними методами, які демонструють низьку ефективність проти модифікованих або поліморфних варіантів шкідливого ПЗ, розроблений прототип виявив значно вищу стійкість до варіативності загроз. Архітектура розробленого прототипу поєднує два принципово різних механізми детектування: сигнатурний та евристичний. Сигнатурний рівень базується на точному порівнянні хешів файлів та IP-адрес із локальними базами індикаторів компрометації. Такий підхід забезпечує практично стовідсоткову точність виявлення вже відомих загроз, проте є малоефективним проти нових або модифікованих зразків шкідливого ПЗ. Натомість евристичний рівень аналізує поведінкові характеристики процесів: мережеву активність, нестандартні порти, обфусковані PowerShell-команди, механізми persistence та аномальні директорії запуску. Саме комбінація цих двох

підходів дозволила реалізувати гібридну модель EDR-класу, здатну одночасно забезпечувати високу точність сигнатурного аналізу та стійкість до Zero-Day загроз і поліморфних RAT-модифікацій.Це пояснюється фокусуванням алгоритму на інваріантних поведінкових патернах, які є незмінними для самої природи функціонування троянів віддаленого доступу – необхідності закріплення в системі та підтримки каналу зв'язку. Навіть якщо криптографічний хеш виконуваного файлу відсутній у базі відомих індикаторів KNOWN_RAT_HASHES_FILE (що є нормою раніше для невідомих ризиків), евристичне ядро все одно здатне його виявити за рахунок таких динамічних індикаторів, як Beaconing (фіксація понад 5 звернень до сервера за 10 секунд у кеші connection_history) або виконання обфускованих інструкцій PowerShell.

Отримані дані підтверджують гіпотезу про те, що поєднання легкої системної телеметрії з гнучкою пороговою моделлю прийняття рішень дозволяє створити ефективний ешелон захисту кінцевої точки. Практична значущість результатів полягає у доведенні можливості побудови автономних засобів безпеки, які здатні забезпечувати високий рівень достовірності виявлення складних загроз при мінімальному навантаженні на обчислювальні ресурси хоста. Мінімізація навантаження стала можливою завдяки використанню оптимізованих запитів API (метод psutil.process_iter), які цілеспрямовано вилучають лише 4 необхідні атрибути процесу, уникаючи ресурсомісткого повного сканування оперативної пам'яті. Зручний графічний інтерфейс розробленої системи дозволяє оператору в режимі реального часу спостерігати за динамічною зміною показників ризику, аналізувати розгорнутий контекст знайдених аномалій та оперативно реагувати на виявлені індикатори компрометації. Уся зібрана аналітична база (масив reasons, що містить деталі детектування) автоматично інтегрується у структуру візуального дерева self.tree та може бути збережена за допомогою модулів автоматизованої генерації звітів у форматах TXT та PDF для подальшого проведення цифрової форензики.

3.4 Висновки до розділу

Завершальним етапом є комплексна інтерпретація та оцінка достовірності експериментальних даних, отриманих під час тестування прототипу в середовищі Windows. Результати апробації підтвердили високу наукову та практичну ефективність розробленої гібридної, сигнатурно-евристичної, моделі детектування. Ця модель успішно об'єднала статичний аналіз за індикаторами компрометації із динамічним профілюванням системних викликів за допомогою бібліотек psutil та winreg, довівши свою працездатність на практиці.

Під час проведення експерименту було запущено розроблений скрипт-імітатор шкідливої поведінки dummy_rat.py на фоні роботи легітимних фонових додатків (месенджери Viber та Telegram, клієнт хмарної синхронізації OneDrive). Аналітичному ядру системи вдалося успішно ідентифікувати імітатор RAT, коректно маркувавши його як критичну загрозу. Система нарахувала імітатору максимальний Threat Score. Такий результат був досягнутий завдяки спрацюванню комплексу евристичних патернів:

– маскування виконуваного файлу під системну службу;

– відкриття нестандартного RAT-порту зі статусом LISTEN для прийому вхідних підключень;

– комбінований патерн активного мережевого з'єднання під час запуску з тимчасової директорії.

Аналітичне ядро системи успішно виявило імітатор RAT та коректно класифікувало його як критичну загрозу. У результаті аналізу система сформувала для процесу Threat Score на рівні 85 балів.

Маскування процесу дало +45 балів ризику, а виявлення підозрілого мережевого порту ще +40 балів. Отримані результати підтвердили здатність системи накопичувати та корелювати поведінкові індикатори, ефективно виявляючи бекдор навіть за умов часткової симуляції механізмів закріплення в системі.

Одночасно з цим, розроблений алгоритм успішно диференціював

активність легітимних додатків, не допустивши критичних хибнопозитивних спрацювань. Наприклад, месенджер Viber, який активно підтримував зовнішні з'єднання та був встановлений у директорії AppData, отримав статус «КРИТИЧНОГО» процесу. Набравши 45 балів (30 за специфічну директорію та 15 за зовнішні з'єднання), програма подолала базовий поріг, але не досягла критичної позначки у 75 балів. Це підтвердило життєздатність розробленої буферної «жовтої зони», яка дозволяє оператору звернути увагу на активність процесу, не класифікуючи його автоматично як критичну загрозу, а також скористатися створеним контекстним меню GUI для його швидкого додавання до списку виключень WHITELIST_FILE.

Для формалізації висновків щодо надійності системи було використано метрику точності, яка характеризує здатність програмного засобу мінімізувати кількість помилкових тригерів на безпечне ПЗ. Детальний аналіз логів системи дозволив встановити, що найвагоміший внесок у зниження рівня False Positives внесла реалізована функція фільтрації мережевого трафіку is_external_ip(). Відкидання локальних IP-адрес дозволило проігнорувати безпечний інформаційний шум, зокрема системні транзитні пакети та запити до локального шлюзу.

Завдяки використанню властивості is_private з об'єкта ipaddress, алгоритм надійно відсік увесь трафік внутрішніх підмереж, дозволивши модулю evaluate_threat обробляти виключно WAN-з'єднання. Це архітектурне рішення стало ключовим фактором у забезпеченні високої чистоти аналітичних даних.

Порівняно з традиційними сигнатурними антивірусами, розроблений EDR-агент виявив значно вищу стійкість до варіативності загроз. Завдяки реалізації механізму перевірки аргументів командного рядка, система здатна ідентифікувати навіть тактики безфайлових інжекцій. Аналітичне ядро гарантовано ініціює критичний алерт при фіксації спроби виконання закодованих команд PowerShell (-enc) або завантаження віддалених payload-файлів, що залишається "сліпою зоною" для більшості базових засобів захисту. Аналіз масиву cmdline кожного активного процесу в реальному часі довів, що розроблений програмний засіб

здатний перехоплювати інструкції зловмисника ще на етапі їх передачі в оперативну пам'ять системним інтерпретаторам, нівелюючи будь-які спроби обфускації шкідливого навантаження.

Ще одним напрямом вимірювань стала оцінка ресурсомісткості агента. Під час пікового навантаження розроблений на базі psutil прототип споживав не більше 3-15% ресурсів центрального процесора та потребував менше 60 МБ оперативної пам'яті. Такі низькі показники споживання апаратних ресурсів пояснюються відмовою від створення важких драйверів рівня ядра на користь багатопотокового сканування простору користувача за прямим запитом оператора, а також використанням механізмів кешування історії активності у словнику connection_history.

Отримані результати підтверджують доцільність поєднання збору системної телеметрії з гнучкою багатокритеріальною моделлю Threat Score дозволяє сформувати ефективний рівень захисту кінцевої точки. Практична цінність роботи полягає в доведенні можливості створення автономних засобів безпеки, які здатні виявляти складні LotL-атаки та RAT-агенти без значного навантаження на ресурси хоста.

Додатковою перевагою є наявність вбудованих модулів автоматизованої генерації звітів (export_pdf, export_txt). Це робить розроблений прототип придатним для інтеграції в корпоративні процеси реагування на кіберінциденти, а також для використання під час збору та фіксації цифрових доказів.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було створено автономний інструмент для виявлення шкідливого програмного забезпечення типу RAT для Windows. Мету досягнуто, всі поставлені завдання виконано. З отриманих результатів та виконаної роботи можна зробити такі висновки.

Аналіз сучасних кіберзагроз показав, що трояни віддаленого доступу стали значно складнішими та небезпечнішими. Вони не лише отримують прихований контроль над системою, а й активно маскуються під легітимні процеси, наприклад svchost.exe, explorer.exe або інші системні компоненти. Зловмисники все частіше використовують стандартні інструменти Windows для виконання шкідливих дій, що ускладнює їх виявлення. Через це класичні антивірусні рішення, які базуються переважно на сигнатурах, не завжди здатні ефективно реагувати на такі загрози. У той же час сучасні EDR-системи демонструють високу точність, однак їх вартість, складність впровадження та значне навантаження на ресурси системи обмежують використання серед звичайних користувачів та малого бізнесу.

З метою подолання цих недоліків була розроблена легка система захисту у форматі сканера за запитом. Вона орієнтована на швидкий аналіз стану системи без постійного фонової роботи. На відміну від класичних антивірусів, запропоноване рішення поєднує кілька підходів до аналізу загроз. Зокрема, воно враховує поведінкові характеристики процесів, перевіряє ключі автозавантаження в системному реєстрі та аналізує активні мережеві з’єднання, які можуть свідчити про наявність прихованої активності.

Ключовим елементом розробки став евристичний алгоритм оцінювання ризику. Його принцип полягає у накопичувальній системі балів, де кожна підозріла дія процесу підвищує загальний рівень загрози. Наприклад, використання PowerShell у прихованому режимі, запуск виконуваних файлів із тимчасових директорій або встановлення з’єднань із нетиповими мережевими портами розглядаються як фактори підвищеного ризику. Такий підхід дозволяє виявляти потенційно шкідливу активність навіть без наявності відомих сигнатур.

Програмний прототип реалізовано мовою Python, що забезпечило гнучкість розробки та швидкість реалізації основного функціоналу. Для взаємодії з користувачем створено зручний графічний інтерфейс на основі CustomTkinter, який робить роботу з програмою інтуїтивно зрозумілою. Для підвищення продуктивності під час перевірки великої кількості процесів використано багатопоточну обробку, що дозволило уникнути «зависань» інтерфейсу. Додатково реалізовано механізм білих списків за іменем файлу та його SHA-256 хешем, що зменшує кількість хибних спрацювань. Результати аналізу можуть експортуватися у вигляді структурованих PDF-звітів, що зручно для подальшого перегляду або архівування.

Практичне тестування в середовищі Windows підтвердило коректність роботи системи. Під час запуску спеціально створеного тестового скрипта, який імітував поведінку RAT, програма успішно виявила загрозу та присвоїла їй критичний рівень ризику. Це свідчить про ефективність обраного підходу до евристичного аналізу. Окремо було проведено оцінку продуктивності, яка показала низьке споживання ресурсів: середнє навантаження становить близько 50 МБ оперативної пам’яті під час активного сканування, що є прийнятним навіть для застарілих або малопотужних систем.

Отримані результати підтверджують, що розроблена система може використовуватися як базовий інструмент для оперативної перевірки комп’ютерів на наявність прихованих загроз. Вона поєднує простоту використання, низькі вимоги до ресурсів і достатній рівень ефективності для первинного аналізу безпеки системи. У перспективі функціональність рішення може бути розширена шляхом інтеграції з онлайн-сервісами перевірки файлів, зокрема VirusTotal, а також впровадження алгоритмів машинного навчання для підвищення точності виявлення складних і раніше невідомих зразків шкідливого програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Aburbeian A. M., Fernández-Veiga M., Hasasneh A. Improving Remote Access Trojans Detection: A Comprehensive Approach Using Machine Learning and Hybrid Feature Engineering. AI. 2025. Vol. 6. Art. 237. DOI: 10.3390/ai6090237(дата звернення: 4.03.2026).

2. Recent Advancements in Machine Learning Models for Malware Detection: A Systematic Literature Review / N. I. Hasanah et al. Eng. Proc. 2025. Vol. 107. Art. 78. DOI: 10.3390/engproc2025107078 (дата звернення: 4.03.2026).

3. Detecting Remote Access Trojan (RAT) Attacks based on Different LAN Analysis Methods / S. J. Rashid, S. A. Baker, O. I. Alsaif, A. I. Ahmad. Engineering, Technology & Applied Science Research. 2024. Vol. 14, No. 5. P. 17294–17301. DOI: 10.48084/etasr.8422 (дата звернення: 4.03.2026).

4. Ravichandran N., Tewaraja T., Rajasegaran V., Kumar S. S., Gunasekar S. K. L., Sindiramutty S. R. Comprehensive Review Analysis and Countermeasures for Cybersecurity Threats: DDoS, Ransomware, and Trojan Horse Attacks (31 p.). Preprints.org. URL: https://www.preprints.org/frontend/manuscript/80168c384a7bf5d76161f742961aa2c5/download_pub (дата звернення: 8.03.2026).

5. Cofense. Q2 2024 Phishing Intelligence Trends Review (17 p.). Cofense – Email Security Solutions. URL: https://cofense.com/getmedia/495d5be3-3a25-4368-a274-114720af3d75/Cofense_Q2-Report.pdf (дата звернення: 8.03.2026).

6. Oh C., Ha J., Roh H. A Survey on TLS-Encrypted Malware Network Traffic Analysis Applicable to Security Operations Centers. Applied Sciences. 2022. Vol. 12. Art. 155. DOI: 10.3390/app12010155. https://www.mdpi.com/2076-3417/12/1/155 (дата звернення: 9.03.2026).

7. CrowdStrike. 2024 Global Threat Report (61 p.). CrowdStrike: Adversary-Focused Cybersecurity. URL: https://github.com/jacobdjwilson/awesome-annual-security-reports/blob/main/Annual%20Security%20Reports/2024/Crowdstrike-Global-Threat-Report-2024.pdf (дата звернення: 12.03.2026).

8. Kazi M. A. Detecting Malware C&C Communication Traffic Using Artificial Intelligence Techniques. Journal of Cybersecurity and Privacy. 2025. Vol. P. 1 -34. DOI: 10.3390/jcp5010004. https://www.mdpi.com/2624-800X/5/1/4 (дата звернення: 12.03.2026).

9. Cybersafety. Remote Access Trojans (RATs): How RAT malware works, real risks, and advanced detection strategies. Cybersafety: Protect your present and secure your future. URL: https://cibersafety.com/en/Trojan-remote-access-RAT-detection/ (дата звернення: 20.03.2026).

10. Singh A. P., Singh M. A Comparative Review of Malware Analysis and Detection in HTTPs Traffic. International Journal of Computing and Digital Systems. 2021. Vol. 10. P. 111–123. DOI: 10.12785/ijcds/100111 (дата звернення: 4.03.2026).

11. SentinelOne. Singularity Platform: One intelligent platform. Unprecedented speed. Infinite scale. SentinelOne. URL: https://www.crowdstrike.com/en-us/resources/reports/business-value-of-the-crowdstrike-falcon-platform/ (дата звернення: 21.03.2026).

12. Malware Detection Based on API Call Sequence Analysis: A Gated Recurrent Unit–Generative Adversarial Network Model Approach / N. Owoh et al. Future Internet. 2024. Vol. 16. Art. 369. DOI: 10.3390/fi16100369 (дата звернення: 21.03.2026).

13. MITRE ATT&CK. Persistence, Tactic TA0003 – Enterprise. MITRE ATT&CK®: Adversary Tactics and Techniques and Common Knowledge. URL: https://attack.mitre.org/tactics/TA0003/ (дата звернення: 24.03.2026).

14. Dehfouli Y., Lashkari A. H. Memory Analysis for Malware Detection: A Comprehensive Survey Using the OSCAR Methodology. ACM Computing Surveys. 2025. Vol. 58. Art. 86. DOI: 10.1145/3764580 (дата звернення: 5.04.2026).

15. Jauhiainen J. Ensuring system integrity and security on limited environment systems (77 p.). University of Turku, Department of Computing. 2021. URL: https://www.utupub.fi/server/api/core/bitstreams/9c5f0388-7865-4a5e-a0a2-f0713098c1a6/content (дата звернення: 5.04.2026).

16. Sherazi S. N. A., Qureshi A. Hybrid Analysis Model for Detecting Fileless

Malware. Electronics. 2025. Vol. 14, Iss. 15. 3134. DOI: 10.3390/electronics14153134 (дата звернення: 6.04.2026).

17. Nagaraj K., Thothathri V., Mohamed N., Ramesh R. Digital Doppelgangers: Anatomy of Evolving Impersonation Campaigns Distributing Gh0st RAT. Unit 42 – Palo Alto Networks. 2025. URL: https://unit42.paloaltonetworks.com/impersonation-campaigns-deliver-gh0st-rat/ (дата звернення: 7.04.2026).

18. Golushko A. Cybersecurity threatscape: Q4 2024 – Q1 2025. Positive Technologies. 2025. URL: https://global.ptsecurity.com/en/research/analytics/cybersecurity-threatscape-q4-2024-q1-2025/ (дата звернення: 10.04.2026).

19. Singh K., Kashyap A., Cherukuri A. K. Interpretable Anomaly Detection in Encrypted Traffic Using SHAP with Machine Learning Models. arXiv.org. 2025. P. 19. URL: https://arxiv.org/abs/2505.16261 (дата звернення: 12.04.2026).

20. Daum S., Shapira T., Bremler-Barr A., Hay D. Non-uniformity is All You Need: Efficient and Timely Encrypted Traffic Classification With ECHO. arXiv.org. 2024. P. 20. URL: https://arxiv.org/abs/2406.01852 (дата звернення: 12.04.2026).

21. Підгорний П., Лаврик Т. Виявлення аномалій у зашифрованому мережевому трафіку за допомогою глибокого навчання. Кібербезпека: освіта, наука, техніка. 2025. С. 89–105. DOI: 10.28925/2663-4023.2025.29.897 (дата звернення: 20.04.2026).

22. Gomez G., Kotzias P., Dell'Amico M., Bilge L., Caballero J. Unsupervised Detection and Clustering of Malicious TLS Flows. Security and Communication Networks. 2023. Vol. 2023. Art. 3676692. DOI: 10.1155/2023/3676692 (дата звернення: 21.04.2026).

23. Faísco M. S. G. C. Real-Time Classification of Malware using a High-Interaction Honeypot with Machine Learning (84 p.). Universidade de Lisboa, Faculdade de Ciências. 2025. URL: https://repositorio.ulisboa.pt/bitstreams/c6131cc5-ab3e-46da-b4e9-add30b791042/download (дата звернення: 22.04.2026).

24. Saha B., Shukla S. K. MalGEN: A Generative Agent Framework for

Modeling Malicious Software in Cybersecurity (11 p.). arXiv.org. 2025. URL: https://arxiv.org/html/2506.07586v1 (дата звернення: 23.04.2026).

25. Chukwuani E. N., Odunsi O. R., Ikemefuna C. D. Machine learning techniques for real-time malware classification and threat detection in distributed systems. World Journal of Advanced Research and Reviews. 2025. Vol. 26. P. 2378–2398. DOI: 10.30574/wjarr.2025.26.3.2433 (дата звернення: 23.04.2026).

26. Ferdous J., Islam R. A Survey on Machine Learning Techniques in Multi-platform Malware Detection: Securing PC, Mobile Devices, IoT, and Cloud Environments (45 p.). Preprints.org. 2024. URL: https://www.preprints.org/frontend/manuscript/0ccf75af1beeee58268345e4f3ea2e5b/download_pub (дата звернення: 25.04.2026).

27. Vasani V., Bairwa A. K., Joshi S., Pljonkin A., Kaur M., Amooon M. Comprehensive Analysis of Advanced Techniques and Vital Tools for Detecting Malware Intrusion. Electronics. 2023. Vol. 12. Art. 4299. DOI: 10.3390/electronics12204299 (дата звернення: 26.04.2026).

28. Есаулов О., Аносов А. Дослідження методів та розробка рекомендацій щодо застосування методів та засобів захисту від Remote Access Trojan. Безпека інформаційно-комунікаційних систем. 2025. С. 50–54. URL: https://elibrary.kubg.edu.ua/id/eprint/54364/1/10O_Esaulov_A_Anosov_scis_2025.pdf (дата звернення: 26.04.2026).

29. Ndibe O. S. Integrating Machine Learning with Digital Forensics to Enhance Anomaly Detection and Mitigation Strategies. International Journal of Advance Research Publication and Reviews. 2025. Vol. 02. P. 365–388. URL: https://www.researchgate.net/publication/392129551 (дата звернення: 26.04.2026).

30. Chatzoglou E., Kambourakis G. C3: Leveraging the Native Messaging Application Programming Interface for Covert Command and Control. Future Internet. 2025. Vol. 17. Art. 172. DOI: 10.3390/fi17040172 (дата звернення: 2.05.2026).

31. Schönle D., Reich C. Evaluation of AI Attack Mitigation: From Citrix Bleed to Self-Evolving Malware: Modernising Aerospace Cyber Defence with AI (27 p.). SSRN. 2025. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5341759

(дата звернення: 2.05.2026).

32. Mennuni M. An Analysis of SOC Monitoring Systems (135 p.). Politecnico di Torino, Department of Control and Computer Engineering (DAUIN). 2023. URL: https://webthesis.biblio.polito.it/29588/1/tesi.pdf (дата звернення: 4.05.2026).

33. Nyathi M. Detection of HTTPS malware traffic without decryption (99 p.). University of Cape Town, Department of Computer Science. 2022. URL: https://open.uct.ac.za/items/2fed2f9b-ff7e-4412-997a-3582809870e5 (дата звернення: 4.05.2026).

34. Chung M.-H., Yang Y., Wang L. et al. Implementing Data Exfiltration Defense in Situ: A Survey of Countermeasures and Human Involvement (37 p.). ACM Computing Surveys. 2023. Vol. 55. Art. 303. DOI: 10.1145/3582077 (дата звернення: 5.05.2026).

35. Habibzadeh A., Feyzi F., Atani R. E. Large Language Models for Security Operations Centers: A Comprehensive Survey (16 p.). arXiv.org. 2025. URL: https://arxiv.org/abs/2509.10858 (дата звернення: 5.05.2026).

36. Gyamfi N. K., Goranin N., Ceponis D., Cenys H. A. Automated System-Level Malware Detection Using Machine Learning: A Comprehensive Review. Applied Sciences. 2023. Vol. 13. Art. 11908. DOI: 10.3390/app131911908 (дата звернення: 7.05.2026).

37. AV-TEST. Test antivirus software for Windows 11 – April 2025. AV-TEST: The Independent IT-Security Institute. 2025. URL: https://www.av-test.org/en/antivirus/home-windows/windows-11/april-2025 (дата звернення: 7.05.2026).

38. CrowdStrike. The CrowdStrike Falcon Platform: Unified Agentic Security. CrowdStrike. URL: https://www.crowdstrike.com/en-us/resources/reports/business-value-of-the-crowdstrike-falcon-platform/ (дата звернення: 7.05.2026).

39. IBM. X-Force Threat Intelligence Index 2026. IBM. 2026. URL: https://www.ibm.com/reports/threat-intelligence (дата звернення: 7.05.2026).

40. ENISA (European Union Agency for Cybersecurity). ENISA Threat Landscape 2024 (131 p.). ENISA. 2024. URL:

https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024 (дата звернення: 8.05.2026).

41. Примаченко Д., Голобородько С., Святська Н. та ін. EDR ТА XDR ЯК ОСНОВНІ ТЕХНОЛОГІЇ ЗАХИСТУ КІНЦЕВИХ ТОЧОК. Кібербезпека: освіта, наука, техніка. 2025. № 4 (28). С. 343–352. DOI: 10.28925/2663-4023.2025.28.808 (дата звернення: 8.05.2026).

ДОДАТОК А

Лістинг коду ПЗ системи

```
import psutil
import hashlib
import winreg
import customtkinter as ctk
from tkinter import messagebox, filedialog, ttk
import threading
import os
import re
import time
import ipaddress
from datetime import datetime
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet

ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

WHITELIST_FILE = "whitelist.txt"
HASH_WHITELIST_FILE = "hash_whitelist.txt"
KNOWN_RAT_HASHES_FILE = "known_rat_hashes.txt"
MALICIOUS_IPS_FILE = "malicious_ips.txt"

SUSPICIOUS_PORTS = [1604, 5552, 7777, 4444, 5555, 6666, 8080, 8888, 9999, 1177, 3360]

connection_history = {}

def load_set_from_file(filename):
    if not os.path.exists(filename):
        return set()
    with open(filename, "r", encoding="utf-8") as f:
        return set(line.strip().lower() for line in f if line.strip())

def save_set_to_file(filename, data_set):
    try:
        with open(filename, "w", encoding="utf-8") as f:
            for entry in sorted(data_set):
                f.write(entry + "\n")
    except PermissionError:
        pass
```

```
def init_files():
    if not os.path.exists(KNOWN_RAT_HASHES_FILE):
        save_set_to_file(KNOWN_RAT_HASHES_FILE, {"e3b0c44298fc1c149afbf4c8996fb924"})
    if not os.path.exists(MALICIOUS_IPS_FILE):
        save_set_to_file(MALICIOUS_IPS_FILE, {"185.112.83.111", "91.215.85.66"})

def get_file_hash(file_path):
    try:
        if not file_path or not os.path.exists(file_path): return None
        h = hashlib.sha256()
        with open(file_path, "rb") as f:
            for chunk in iter(lambda: f.read(4096), b""):
                h.update(chunk)
        return h.hexdigest()
    except: return None

def is_external_ip(ip):
    try:
        return not ipaddress.ip_address(ip).is_private
    except:
        return False

def evaluate_threat(proc_info, net_data, autorun_paths, current_wl, current_hwl, known_hashes, malicious_ips):
    score = 0
    reasons = []

    path = str(proc_info.get('exe')).lower()
    name = str(proc_info.get('name')).lower()
    pid = proc_info.get('pid')
    cmdline_list = proc_info.get('cmdline')

    if not path or path == 'none': return 0, []
    if name in current_wl or path in current_wl: return 0, []

    f_hash = get_file_hash(path)
    if f_hash and f_hash in current_hwl: return 0, []
    if f_hash in known_hashes:
        return 100, ["Відомий RAT (знайдено в базі IoC)"]

    cmd = ""
    if cmdline_list:
```

```
    cmd = " ".join(cmdline_list).lower()

if "powershell" in cmd and "-enc" in cmd:
    score += 60
    reasons.append("Прихований запуск PowerShell (Encoded)")
if "downloadstring" in cmd:
    score += 60
    reasons.append("Завантаження шкідливого Payload")
if "cmd.exe" in cmd and "/c" in cmd:
    score += 25
    reasons.append("Фонове виконання команд (cmd /c)")

in_temp = False
if "appdata" in path or "temp" in path:
    in_temp = True
    score += 30
    reasons.append("Запуск з AppData/Temp")

base_name = name.replace(".exe", "")
if base_name in ["svchost", "explorer", "lsass", "csrss", "winlogon"]:
    if "system32" not in path and "windows" not in path:
        score += 45
        reasons.append("Маскування під системний процес")

has_net = False
external_conns = 0

if pid in net_data:
    has_net = True
    for c in net_data[pid]:
        port = c['port']
        ip = c['ip']

        if ip in malicious_ips:
            score += 60
            reasons.append(f"Зв'язок з Blacklist IP ({ip})")

        if port in SUSPICIOUS_PORTS:
            score += 40
            reasons.append(f"Відкритий RAT-порт {port}")

        if ip and is_external_ip(ip):
```

```
                external_conns += 1

        if external_conns > 0:
            if external_conns > 10:
                score += 30
                reasons.append(f"Мережева аномалія: {external_conns} зовнішніх з'єднань")
            else:
                score += 15
                reasons.append(f"Зовнішні з'єднання ({external_conns})")

    if pid in connection_history:
        if len(connection_history[pid]) > 5:
            score += 40
            reasons.append("Beaconing (Висока частота мережевих запитів)")

    in_autorun = False
    for auto in autorun_paths:
        if name in auto or path in auto:
            in_autorun = True
            score += 35
            reasons.append("Присутність в автозавантаженні")

    if in_temp and has_net:
        score += 20
        reasons.append("[Патерн] Temp-папка + Мережева активність")
    if "powershell" in cmd and has_net:
        score += 25
        reasons.append("[Патерн] PowerShell + Мережа (можливий C&C зв'язок)")
    if in_autorun and has_net:
        score += 20
        reasons.append("[Патерн] Автозапуск + Мережа (Persistence)")

    reasons = list(set(reasons))
    return min(score, 100), reasons

class ModernRATDetector(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Система виявлення RAT")
        self.geometry("1250x780")
        init_files()
```

```
        self.report_list = []

        self.grid_columnconfigure(1, weight=1)
        self.grid_rowconfigure(0, weight=1)

        self.sidebar = ctk.CTkFrame(self, width=240, corner_radius=0)
        self.sidebar.grid(row=0, column=0, sticky="nsew")

        self.logo = ctk.CTkLabel(self.sidebar, text="СИСТЕМА\nВИЯВЛЕННЯ RAT",
                                 font=ctk.CTkFont(size=18, weight="bold"),
                                 text_color="#5dade2")
        self.logo.pack(pady=(40, 20))

        self.btn_scan = ctk.CTkButton(self.sidebar, text="Запустити аналіз",
                                      height=40, font=ctk.CTkFont(weight="bold"),
                                      command=self.start_scan)
        self.btn_scan.pack(pady=10, padx=20)

        self.btn_edit_wl = ctk.CTkButton(self.sidebar, text="Редагувати WL",
                                         fg_color="transparent", border_width=2,
                                         command=self.open_whitelist_editor)
        self.btn_edit_wl.pack(pady=10, padx=20)

        self.btn_add_sig = ctk.CTkButton(self.sidebar, text="Додати IoC (Хеш/IP)",
                                         fg_color="transparent", border_width=2,
                                         border_color="#e74c3c", text_color="#e74c3c",
                                         hover_color="#3a1c1c",
                                         command=self.add_ioc_dialog)
        self.btn_add_sig.pack(pady=10, padx=20)

        ctk.CTkFrame(self.sidebar, height=2, fg_color="#333333").pack(fill="x", padx=20, pady=10)
        self.btn_export_txt = ctk.CTkButton(self.sidebar, text="Експорт TXT", fg_color="#444444",
hover_color="#555555", command=self.export_txt)
        self.btn_export_txt.pack(pady=5, padx=20)
        self.btn_export_pdf = ctk.CTkButton(self.sidebar, text="Експорт PDF", fg_color="#444444",
hover_color="#555555", command=self.export_pdf)
        self.btn_export_pdf.pack(pady=5, padx=20)

        self.main_content = ctk.CTkFrame(self, corner_radius=15, fg_color="transparent")
        self.main_content.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")

        self.stats_frame = ctk.CTkFrame(self.main_content, fg_color="transparent")
```

```
        self.stats_frame.pack(fill="x", pady=(0, 20))

        self.stat_total = self.create_stat_card(self.stats_frame, "Об'єктів знайдено", "0", "#3b8ed0")
        self.stat_crit = self.create_stat_card(self.stats_frame, "Критичні загрози", "0", "#e74c3c")
        self.stat_warn = self.create_stat_card(self.stats_frame, "Підозріла активність", "0", "#f39c12")

        self.setup_table()

        self.hint_label = ctk.CTkLabel(self.main_content,
                                       text=" Права кнопка миші – додати в білий список. Подвійний клік лівою кнопкою – розгорнути деталі.",
                                       font=("Segoe UI", 12, "italic"),
                                       text_color="#5dade2")
        self.hint_label.pack(pady=(15, 0))

        self.context_menu = ctk.CTkFrame(self, width=160, corner_radius=8, border_width=1, fg_color="#333333")
        ctk.CTkButton(self.context_menu, text="Додати ім'я в WL", fg_color="transparent", hover_color="#444444", anchor="w", command=self.add_name_context).pack(fill="x", padx=5, pady=2)
        ctk.CTkButton(self.context_menu, text="Додати хеш в WL", fg_color="transparent", hover_color="#444444", anchor="w", command=self.add_hash_context).pack(fill="x", padx=5, pady=2)

        self.tree.bind("<Button-3>", self.show_context_menu)
        self.bind("<Button-1>", self.hide_context_menu)

    def setup_table(self):
        style = ttk.Style()
        style.theme_use("default")
        style.configure("Treeview", background="#2b2b2b", foreground="white", fieldbackground="#2b2b2b", borderwidth=0, font=("Segoe UI", 10), rowheight=35)
        style.configure("Treeview.Heading", background="#333333", foreground="black", relief="flat", font=("Segoe UI", 10, "bold"))
        style.map("Treeview", background=[('selected', '#3b8ed0')])

        self.tree = ttk.Treeview(self.main_content, columns=("status", "name", "pid", "score", "reason"), show='headings')

        self.tree.heading("status", text="Статус")
        self.tree.heading("name", text="Процес")
        self.tree.heading("pid", text="PID")
        self.tree.heading("score", text="Ризик %")
        self.tree.heading("reason", text="Деталі детектування")

        self.tree.column("status", width=120, anchor="center")
```

```
        self.tree.column("name", width=180)
        self.tree.column("pid", width=80, anchor="center")
        self.tree.column("score", width=80, anchor="center")
        self.tree.column("reason", width=500, stretch=True)

        self.tree.pack(fill="both", expand=True)

        self.tree.bind("<Double-1>", self.show_details)

    def show_details(self, event):
        selected = self.tree.focus()
        if not selected:
            return

        values = self.tree.item(selected)['values']
        name = values[1]
        score = values[3]
        reasons = values[4]

        formatted_reasons = str(reasons).replace(", ", "\n• ")

        dialog = ctk.CTkToplevel(self)
        dialog.title(f"Деталі процесу: {name}")
        dialog.geometry("600x300")
        dialog.attributes("-topmost", True)

        ctk.CTkLabel(dialog, text=f"Аналіз процесу: {name} (Ризик: {score})", font=ctk.CTkFont(size=16, weight="bold"),
text_color="#e74c3c").pack(pady=(20, 10))

        textbox = ctk.CTkTextbox(dialog, width=550, height=180, font=("Segoe UI", 12))
        textbox.pack(pady=10, padx=20)
        textbox.insert("0.0", f"Знайдені загрози та патерни:\n\n• {formatted_reasons}")
        textbox.configure(state="disabled")

    def create_stat_card(self, parent, title, value, color):
        card = ctk.CTkFrame(parent, width=220, height=110, fg_color=color, corner_radius=12)
        card.pack(side="left", padx=15)
        card.pack_propagate(False)
        ctk.CTkLabel(card, text=title, font=("Segoe UI", 12)).pack(pady=(15, 0))
        lbl_val = ctk.CTkLabel(card, text=value, font=("Segoe UI", 32, "bold"))
        lbl_val.pack()
        return lbl_val
```

```
    def add_ioc_dialog(self):
        dialog = ctk.CTkToplevel(self)
        dialog.title("Додавання IoC (Хеш або IP)")
        dialog.geometry("450x220")
        dialog.attributes("-topmost", True)

        ctk.CTkLabel(dialog, text="Введіть SHA-256 хеш або шкідливу IP-адресу:", font=("Segoe UI", 12)).pack(pady=(20, 10))
        entry = ctk.CTkEntry(dialog, width=350, placeholder_text="Хеш (64 симв.) або IPv4...")
        entry.pack(pady=5)

        def paste_from_clipboard():
            try:
                content = dialog.clipboard_get()
                entry.delete(0, 'end')
                entry.insert(0, content)
            except: pass

        def submit():
            val = entry.get().strip().lower()
            if len(val) == 64 and re.match(r'^[0-9a-f]+$', val):
                hashes = load_set_from_file(KNOWN_RAT_HASHES_FILE)
                hashes.add(val)
                save_set_to_file(KNOWN_RAT_HASHES_FILE, hashes)
                messagebox.showinfo("Успіх", "Хеш успішно додано до бази!")
                dialog.destroy()
            elif re.match(r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$', val):
                ips = load_set_from_file(MALICIOUS_IPS_FILE)
                ips.add(val)
                save_set_to_file(MALICIOUS_IPS_FILE, ips)
                messagebox.showinfo("Успіх", "IP-адресу додано до Blacklist!")
                dialog.destroy()
            else:
                messagebox.showerror("Помилка", "Невірний формат! Введіть SHA-256 (64 символи) або коректний IPv4.")

        btn_frame = ctk.CTkFrame(dialog, fg_color="transparent")
        btn_frame.pack(pady=15)
        ctk.CTkButton(btn_frame, text="Вставити", width=120, fg_color="#34495e", command=paste_from_clipboard).pack(side="left", padx=10)
```

```
        ctk.CTkButton(btn_frame, text="Зберегти", width=120, fg_color="#e74c3c", hover_color="#c0392b", command=submit).pack(side="left", padx=10)

    def show_context_menu(self, event):
        item = self.tree.identify_row(event.y)
        if item:
            self.tree.selection_set(item)
            self.context_menu.place(x=event.x_root - self.winfo_rootx(), y=event.y_root - self.winfo_rooty())

    def hide_context_menu(self, event):
        self.context_menu.place_forget()

    def add_name_context(self):
        selected = self.tree.focus()
        if selected:
            name = self.tree.item(selected)['values'][1]
            wl = load_set_from_file(WHITELIST_FILE)
            wl.add(name.lower())
            save_set_to_file(WHITELIST_FILE, wl)
            messagebox.showinfo("Whitelist", f"Процес '{name}' додано до довірених.")
            self.hide_context_menu(None)

    def add_hash_context(self):
        selected = self.tree.focus()
        if selected:
            pid = int(self.tree.item(selected)['values'][2])
            try:
                proc = psutil.Process(pid)
                f_hash = get_file_hash(proc.exe())
                if f_hash:
                    hwl = load_set_from_file(HASH_WHITELIST_FILE)
                    hwl.add(f_hash)
                    save_set_to_file(HASH_WHITELIST_FILE, hwl)
                    messagebox.showinfo("Whitelist", "Хеш файлу додано до безпечних.")
            except: pass
            self.hide_context_menu(None)

    def open_whitelist_editor(self):
        editor = ctk.CTkToplevel(self)
        editor.title("Керування базами (WL / Blacklist)")
        editor.geometry("600x500")
        editor.attributes("-topmost", True)
```

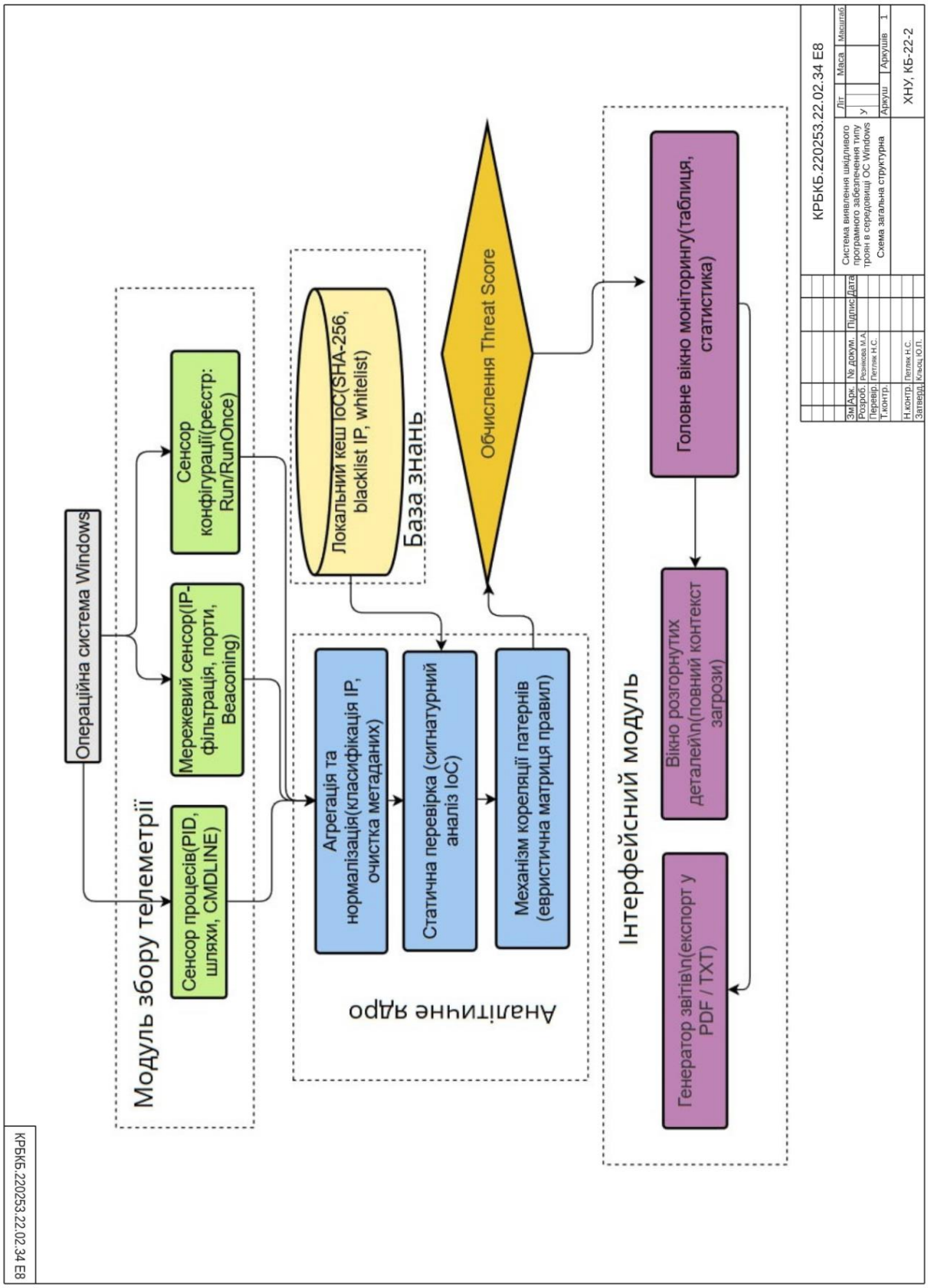
```
        listbox = ttk.Treeview(editor, columns=("type", "value"), show='headings')
        listbox.heading("type", text="База")
        listbox.heading("value", text="Значення")
        listbox.column("type", width=120)
        listbox.column("value", width=430)
        listbox.pack(padx=20, pady=20, fill="both", expand=True)

        def refresh():
            for i in listbox.get_children(): listbox.delete(i)
            for item in load_set_from_file(WHITELIST_FILE): listbox.insert("", "end", values=("WL: IM'Я", item))
            for item in load_set_from_file(HASH_WHITELIST_FILE): listbox.insert("", "end", values=("WL: ХЕШ", item))
            for item in load_set_from_file(MALICIOUS_IPS_FILE): listbox.insert("", "end", values=("BL: IP", item))

        def delete():
            selected = listbox.focus()
            if not selected: return
            vals = listbox.item(selected)['values']
            if vals[0] == "WL: IM'Я":
                wl = load_set_from_file(WHITELIST_FILE); wl.discard(str(vals[1])); save_set_to_file(WHITELIST_FILE, wl)
            elif vals[0] == "WL: ХЕШ":
                hwl = load_set_from_file(HASH_WHITELIST_FILE); hwl.discard(str(vals[1])); save_set_to_file(HASH_WHITELIST_FILE, hwl)
            elif vals[0] == "BL: IP":
                bl = load_set_from_file(MALICIOUS_IPS_FILE); bl.discard(str(vals[1])); save_set_to_file(MALICIOUS_IPS_FILE, bl)
            refresh()

        refresh()
        ctk.CTkButton(editor, text="Видалити запис", fg_color="#e74c3c", hover_color="#c0392b", command=delete).pack(pady=20)

    def scan(self):
        for i in self.tree.get_children(): self.tree.delete(i)
        self.report_list.clear()

        current_wl = load_set_from_file(WHITELIST_FILE)
        current_hwl = load_set_from_file(HASH_WHITELIST_FILE)
        known_hashes = load_set_from_file(KNOWN_RAT_HASHES_FILE)
        malicious_ips = load_set_from_file(MALICIOUS_IPS_FILE)

        net, autorun = {}, []
```

```
now = time.time()

try:
    for conn in psutil.net_connections(kind='inet'):
        if conn.pid and conn.status in ['ESTABLISHED', 'LISTEN']:
            port = conn.raddr.port if conn.raddr else conn.laddr.port
            ip = conn.raddr.ip if conn.raddr else None
            net.setdefault(conn.pid, []).append({'port': port, 'ip': ip})

            connection_history.setdefault(conn.pid, []).append(now)
            connection_history[conn.pid] = [t for t in connection_history[conn.pid] if now - t < 10]

    key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, r"Software\Microsoft\Windows\CurrentVersion\Run")
    for i in range(winreg.QueryInfoKey(key)[1]):
        autorun.append(winreg.EnumValue(key, i)[1].lower())
except: pass

c_crit, c_warn, c_total = 0, 0, 0
found_processes = [] # Створюємо список для тимчасового зберігання результатів

for proc in psutil.process_iter(['pid', 'name', 'exe', 'cmdline']):
    try:
        score, reasons = evaluate_threat(proc.info, net, autorun, current_wl, current_hwl, known_hashes, malicious_ips)

        if score > 0:
            if score >= 75:
                status = "КРИТИЧНО"
                c_crit += 1
            elif score >= 40:
                status = "ПІДОЗРІЛО"
                c_warn += 1
            else:
                status = "ІНФО"

            row = (status, proc.info['name'], proc.info['pid'], f"{score}%", '', ''.join(reasons))

            # Зберігаємо результат у список у вигляді кортежу (score, рядок_для_таблиці, назва_процесу)
            found_processes.append((score, row, proc.info['name']))
            c_total += 1
    except: pass
```

```
        found_processes.sort(key=lambda x: x[0], reverse=True)

        for score, row_data, proc_name in found_processes:
            # Додаємо у візуальну таблицю
            self.tree.insert("", "end", values=row_data)

            # Додаємо до списку експорту
            status = row_data[0]
            pid = row_data[2]
            reasons_str = row_data[4]
            self.report_list.append(f"[{datetime.now().strftime('%H:%M:%S')}] {status} | {proc_name} | PID: {pid} | Ризик: {score}% | Причини: {reasons_str}")

        self.stat_total.configure(text=str(c_total))
        self.stat_crit.configure(text=str(c_crit))
        self.stat_warn.configure(text=str(c_warn))

    def start_scan(self):
        threading.Thread(target=self.scan, daemon=True).start()

    def export_txt(self):
        path = filedialog.asksaveasfilename(defaultextension=".txt", initialfile="Scan_Report.txt")
        if path:
            with open(path, "w", encoding="utf-8") as f:
                f.write("\n".join(self.report_list))

    def export_pdf(self):
        path = filedialog.asksaveasfilename(defaultextension=".pdf", initialfile="Scan_Report.pdf")
        if path:
            doc = SimpleDocTemplate(path)
            styles = getSampleStyleSheet()
            content = [Paragraph("Звіт системи виявлення RAT", styles["Title"]), Spacer(1, 12)]
            for line in self.report_list:
                content.append(Paragraph(line, styles["Normal"]))
                content.append(Spacer(1, 6))
            doc.build(content)

if __name__ == "__main__":
    app = ModernRATDetector()
    app.mainloop()
```

ДОДАТОК Б

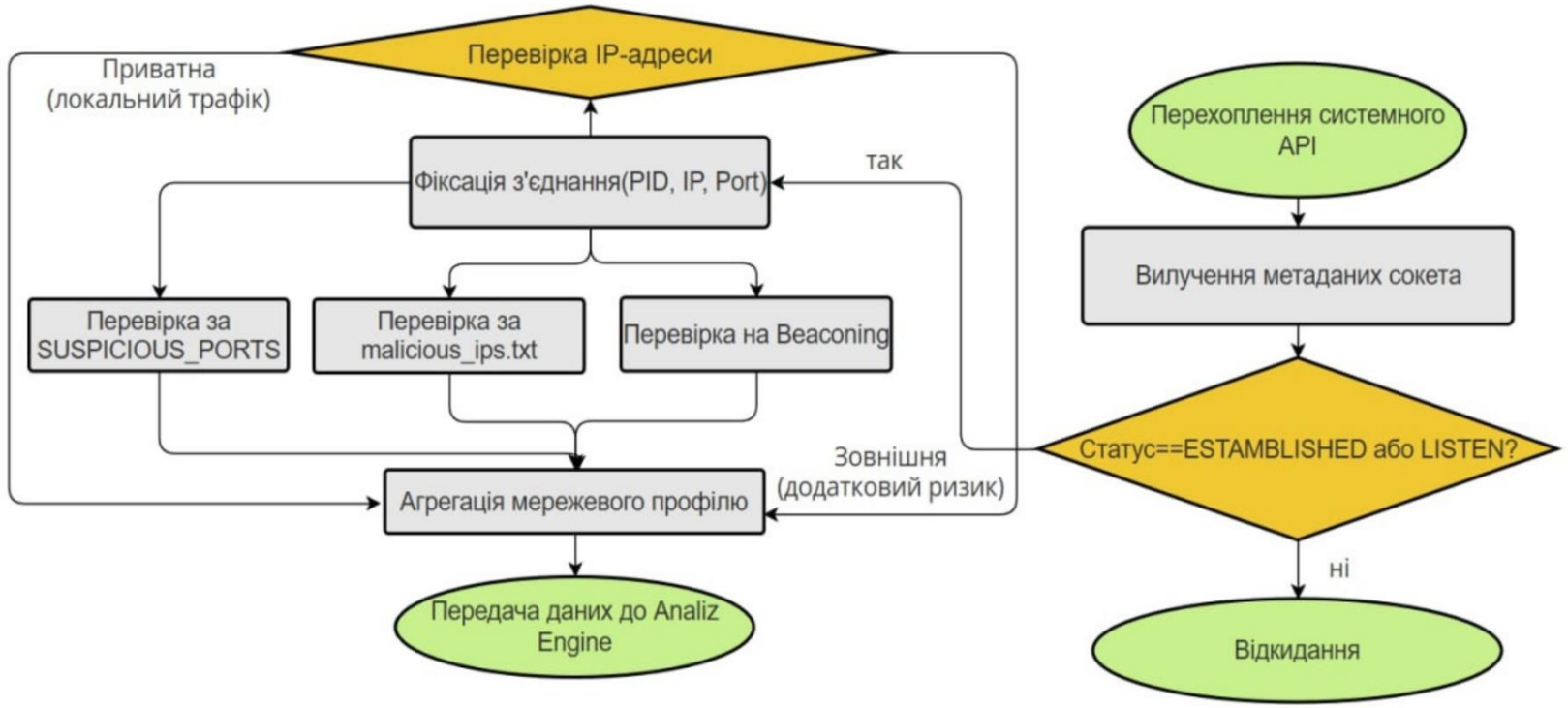
Копія графічної частини



КРБКБ.220253.22.02.34 Е8

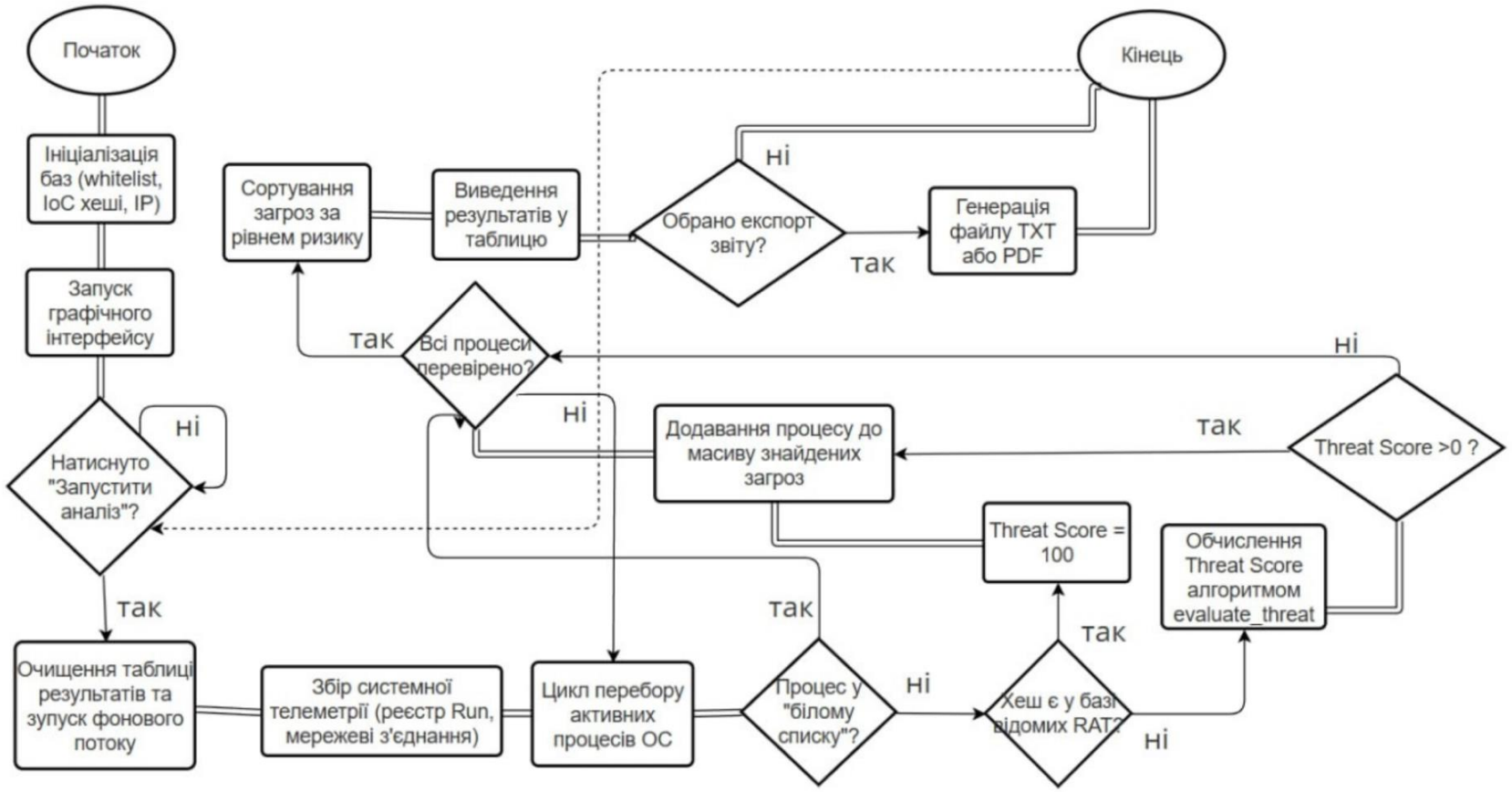
					КРБКБ.220253.22.02.34 Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливого програмного забезпечення типу троян в середовищі ОС Windows	Літ: У; Маса; Масштаб
Розроб.		Резнікова М.А.				
Перевір.		Петляк Н.С.			Схема прийняття рішення	Аркуш; Аркушів 1
Т.контр.						
Н.контр.		Петляк Н.С.				ХНУ, КБ-22-2
Затверд.		Кльоц Ю.П.				

КРБКБ.220253.22.02.34 Е8

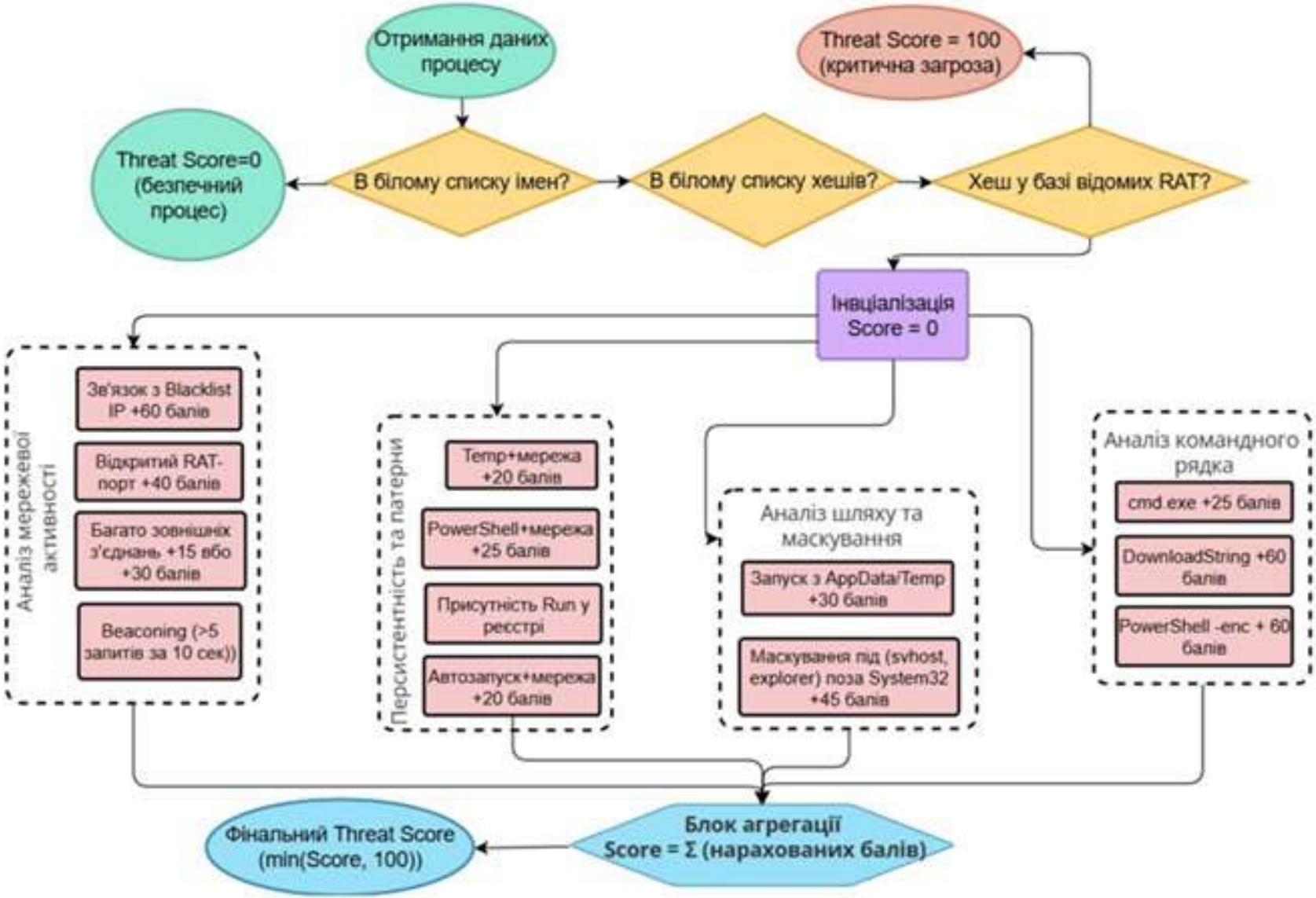


					КРБКБ.220253.22.02.34 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливого програмного забезпечення типу троян в середовищі ОС Windows	Літ.	Маса	Масштаб
Розроб.		Резнікова М.А.				У		
Перевір.		Петляк Н.С.			Схема алгоритму мережевого моніторингу	Аркуш	Аркушів 1	
Т.контр.								
Н.контр.		Петляк Н.С.				ХНУ, КБ-22-2		
Затверд.		Кльоц Ю.П.						

КРБКБ.220253.22.02.34 Е8



					КРБКБ.220253.22.02.34 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливого програмного забезпечення типу троян в середовищі ОС Windows	Літ	Маса / Масштаб
Розроб.		Резнікова М.А.				У	
Перевір.		Петляк Н.С.			Схема алгоритму роботи системи виявлення RAT	Аркуш	Аркушів 1
Т.контр.							
Н.контр.		Петляк Н.С.				ХНУ, КБ-22-2	
Затверд.		Кльоц Ю.П.					



					КРБКБ.220253.22.02.34 Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливого програмного забезпечення типу троян в середовищі ОС Windows. Схема алгоритичного аналізу системи	Літ. / Маса / Масштаб
Розроб.		Резников М.А.				У
Перевір.		Петлик Н.С.				Аркуш 1 / Аркушів 1
Т.контр.						
Н.контр.		Петлик Н.С.				ХНУ, КБ-22-2
Затверд.		Кільчь Ю.П.				