МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

РОМАНІВ Олег Володимирович

Алгоритми виявлення та захисту від зловмисного програмного забезпечення / Malware Detection and Protection Algorithms

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -21
О.В. Романів

Науковий керівник
к.т.н., доцент С.В.Івасьєв

Кваліфікаційну роботу
Допущено до захисту:

«____» _____ ___2024 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ – 2024

Факультет комп'ютерних інформаційних технологій

Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
«____» _____ 2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Романів Олег Володимирович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми виявлення та захисту від зловмисного програмного забезпечення / Malware Detection and Protection Algorithms
керівник роботи к.т.н., доцент Івасьєв С.В.
затверджені наказом по університету від 12 грудня 2023 року № 753
2. Строк подання студентом закінченої кваліфікаційної роботи 12 грудня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- провести аналіз захисту інформаційних систем від шкідливого програмного забезпечення.
- проаналізувати антивірусне програмне забезпечення.
- проаналізувати міжмережеві екрани.
- дослідити статичний аналіз шікливого ПЗ.
- дослідити підходи динамічного аналізу шікливого ПЗ.
- дослідити сигнатурний аналіз.
5. Перелік графічного матеріалу у роботі:
– ахітектура системи IDS/IPS;
– типи шкідливого програмного забезпечення;
– принципи роботи NGFW;
– статичний аналіз ШПЗ;
- використання Cuckoo Sandbox.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз предметної області	12.2023 р. – 03.2024 р.	
2	Техніки дослідження ШПЗ	03.2024 р. – 06.2024 р.	
3	Проектування та розробка засобу виявлення ШПЗ	06.2024 р. – 11.2024 р.	

Студент _____ О.В. Романів
(підпис)

Керівник роботи _____ С.В. Івасьєв.
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми виявлення та захисту від зловмисного програмного забезпечення» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» освітньо-професійної програми «Кібербезпека» написана обсягом 52 сторінки і містить 26 ілюстрацій, 1 таблицю, 2 додатки та 32 джерел за переліком посилань.

Метою кваліфікаційної роботи є розробка ефективних алгоритмів виявлення та захисту від зловмисного програмного забезпечення для забезпечення високого рівня безпеки інформаційних систем. Проведено аналіз сучасних методів виявлення шкідливого програмного забезпечення, таких як сигнатурний, статичний та динамічний аналіз. Досліджено ефективність правил YARA для визначення потенційно небезпечних файлів та проаналізовано можливості інтеграції із сервісом VirusTotal для розширення можливостей аналізу.

Розроблено програмний засіб, який поєднує можливості статичного і динамічного аналізу файлів, застосування правил YARA та використання API-сервісів. Реалізовано функції багатопотокового аналізу, генерації звітів у форматах PDF і CSV, а також журналювання подій. Проведено тестування програмного засобу, яке підтвердило його ефективність та стабільну роботу. Виявлено шкідливі файли в тестовій вибірці та продемонстровано можливості для подальшого розвитку системи, зокрема розширення підтримуваних форматів файлів та інтеграція додаткових сервісів кібербезпеки.

Ключові слова: шкідливе програмне забезпечення, кібербезпека, YARA, VirusTotal, сигнатурний аналіз.

ABSTRACT

The qualification work on the topic "Algorithms for Detection and Protection Against Malware" for obtaining the Master's degree in the specialty 125 "Cybersecurity and Information Protection" of the educational and professional program "Cyber Security" is written in the volume of 52 pages and contains 26 illustrations, 1 tables, 1 appendix, and 32 sources in the list of references.

The purpose of the qualification work is to develop effective algorithms for detecting and protecting against malware to ensure a high level of information system security.An analysis of modern methods for detecting malware, such as signature-based, static, and dynamic analysis, was conducted.

The efficiency of YARA rules for identifying potentially dangerous files was investigated, and the capabilities of integrating with the VirusTotal service were analyzed. A software tool was developed that combines the capabilities of static and dynamic file analysis, YARA rule application, and API service integration. Functions of multi-threaded analysis, report generation in PDF and CSV formats, and event logging were implemented.The software was tested, confirming its effectiveness and stable performance. Malware in the test dataset was identified, demonstrating the potential for further development, including support for additional file formats and integration with additional cybersecurity services.

Keywords: malware, cybersecurity, YARA, VirusTotal, signature-based analysis.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі інформаційні технології є невід'ємною частиною життя кожної людини та суспільства в цілому. Водночас, зі стрімким розвитком цифрових технологій зростає кількість кіберзагроз, які ставлять під загрозу безпеку персональних даних, конфіденційність інформації та стабільність функціонування комп'ютерних систем. Одним із найбільших викликів у цій сфері є шкідливе програмне забезпечення (malware), яке постійно вдосконалюється, використовуючи нові техніки обходу традиційних засобів захисту. Актуальність теми магістерської роботи обумовлена необхідністю впровадження сучасних рішень для боротьби зі шкідливим ПЗ, яке завдає значної шкоди як окремим користувачам, так і великим організаціям. Традиційні методи виявлення, такі як сигнатурний аналіз, уже не є достатньо ефективними, що вимагає використання новітніх підходів, зокрема поведінкового аналізу, машинного навчання та інших технологій.

Мета і завдання дослідження. Метою магістерської роботи є дослідження, розробка та впровадження алгоритмів виявлення та захисту від зловмисного програмного забезпечення, які забезпечують надійний рівень безпеки інформаційних систем. Для досягнення цієї мети було поставлено низку завдань: аналіз сучасних підходів до протидії шкідливому ПЗ, розробка програмного забезпечення з використанням інструментів YARA та VirusTotal, а також оцінка ефективності запропонованих рішень.

Об’єкт дослідження – процес дослідження шкідливого програмного забезпечення.

Предмет дослідження – алгоритми сигнатурного аналізу та виявлення шкідливого програмного забезпечення.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: теоретичний аналіз, експериментальні методи, порівняльний аналіз, алгоритмічний синтез, статистичний аналіз.

Наукова новизна одержаних результатів. Удосконалено алгоритм сигнатурного аналізу для виявленння шкідливого програмного забезпечення з використанням Yara.

Практичне значення отриманих результатів. В рамках роботи проведено комплексне дослідження сучасних методів аналізу, таких як статичний, динамічний та сигнатурний аналіз, і розроблено прототип системи, яка інтегрує ці підходи для ефективного виявлення кіберзагроз.

Публікації та апробація КР проведена в двох конференціях (додаток А).

1. Чубей О, Романів О., Івасьєв С. Алгоритми виявлення та захисту від зловмисного програмного забезпечення. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп’ютерно-інтегровані технології» (АКІТ - 2024), Тернопіль, 2024. -С. 73-76.

2. Романів О., Кобиця В., Лизун Я., Автоматизована перевірка програмних засобів для виявлення шкідливого впливу на ОС. Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024.С. 146-150.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Сучасні підходи до захисту інформаційних систем від шкідливого програмного забезпечення

У сучасному світі кіберзагрози набувають усе більш витончених форм, тому потреба в ефективних засобах захисту інформаційних систем стає критичною. Розробка і впровадження сучасних технологій для протидії шкідливому програмному забезпеченню (ШПЗ) є важливим аспектом кібербезпеки, що вимагає постійного вдосконалення методів і підходів. Шкідливе програмне забезпечення може включати віруси, черв’яки, трояни, шпигунські програми та програми-вимагачі, які здатні завдати серйозної шкоди як окремим користувачам, так і великим організаціям. Тому сучасні засоби захисту мають бути інтегровані в багатошарові системи безпеки, що забезпечують захист на всіх рівнях[1]. Сучасні підходи до захисту інформаційних систем базуються на комплексних рішеннях, які об'єднують різноманітні технології та підходи, від антивірусних програм до використання систем виявлення вторгнень. Сучасні антивірусні програми є найпоширенішим засобом захисту і включають в себе функції перевірки файлів і програм на предмет підозрілих активностей або коду, який потенційно може бути шкідливим. Важливою характеристикою цих програм є використання баз даних сигнатур, які містять зразки відомих шкідливих програм. Однак через постійне вдосконалення методів шкідливих програм, антивірусні рішення змушені використовувати поведінковий аналіз, що дозволяє виявляти загрози, які не мають відомих сигнатур. Це особливо важливо в боротьбі з поліморфними вірусами, що змінюють свою структуру, щоб уникнути виявлення.

Антивірусне програмне забезпечення постійно еволюціонує. Більшість сучасних антивірусних рішень не лише забезпечують захист від відомих загроз, але й мають функції штучного інтелекту, які аналізують

поведінку додатків у реальному часі. Наприклад, вони можуть розпізнавати програми, які намагаються отримати доступ до системних файлів або виконати небезпечні операції. Це дозволяє ефективно боротися з новими загрозами, які ще не були ідентифіковані в базах даних. Крім того, сучасні антивірусні програми можуть оновлювати свої бази даних через хмарні сервіси, що забезпечує швидке реагування на нові виклики кібербезпеки.

Міжмережеві екрани також відіграють важливу роль у захисті інформаційних систем. Вони дозволяють контролювати весь вхідний і вихідний трафік, блокуючи спроби вторгнень та забезпечуючи безпеку мережі. Сучасні міжмережеві екрани можуть працювати на різних рівнях: від мережевого рівня до рівня додатків. Це забезпечує можливість гнучкого налаштування політик безпеки. Наприклад, міжмережеві екрани можуть аналізувати вміст трафіку в реальному часі, виявляючи підозрілу активність, таку як спроби несанкціонованого доступу або передачу шкідливих даних. У сучасних умовах розвиток технологій призвів до появи так званих «наступних поколінь» міжмережевих екранів (Next-Generation Firewalls), які поєднують у собі можливості аналізу вмісту (рисунок 1.1), виявлення загроз та автоматичного блокування підозрілих з'єднань.

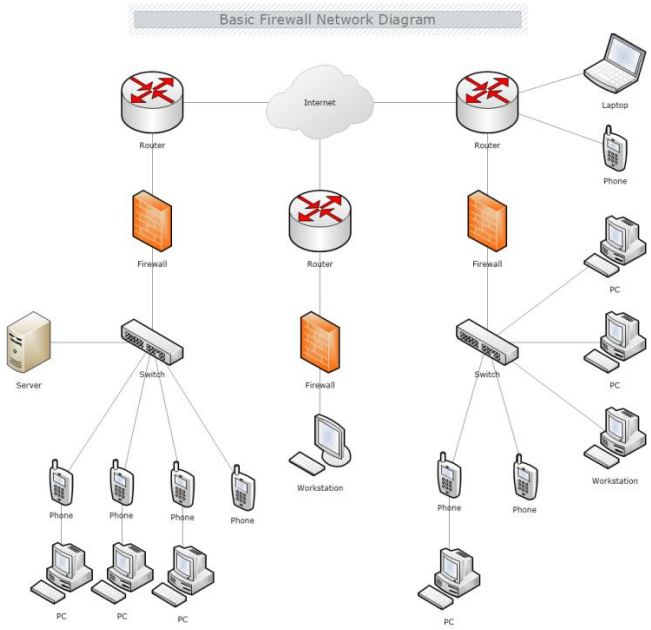

Рисунок 1.1 - Еволюцію антивірусних технологій

Системи виявлення та запобігання вторгненням (IDS/IPS) є ще одним важливим елементом захисту інформаційних систем (рисунок 1.2). Вони дозволяють аналізувати трафік і виявляти спроби вторгнень, які можуть бути пропущені іншими засобами безпеки. Системи IDS виконують функцію моніторингу та повідомлення про можливі загрози, тоді як системи IPS можуть автоматично блокувати підозрілий трафік. Розвиток технологій машинного навчання та штучного інтелекту значно покращив ефективність цих систем[2]. Вони здатні виявляти складні атаки, аналізуючи величезні обсяги даних і розпізнаючи шаблони, що свідчать про можливу загрозу. Водночас впровадження таких систем потребує значних ресурсів, тому їх використання актуальне в основному для великих організацій, які можуть дозволити собі такі витрати.

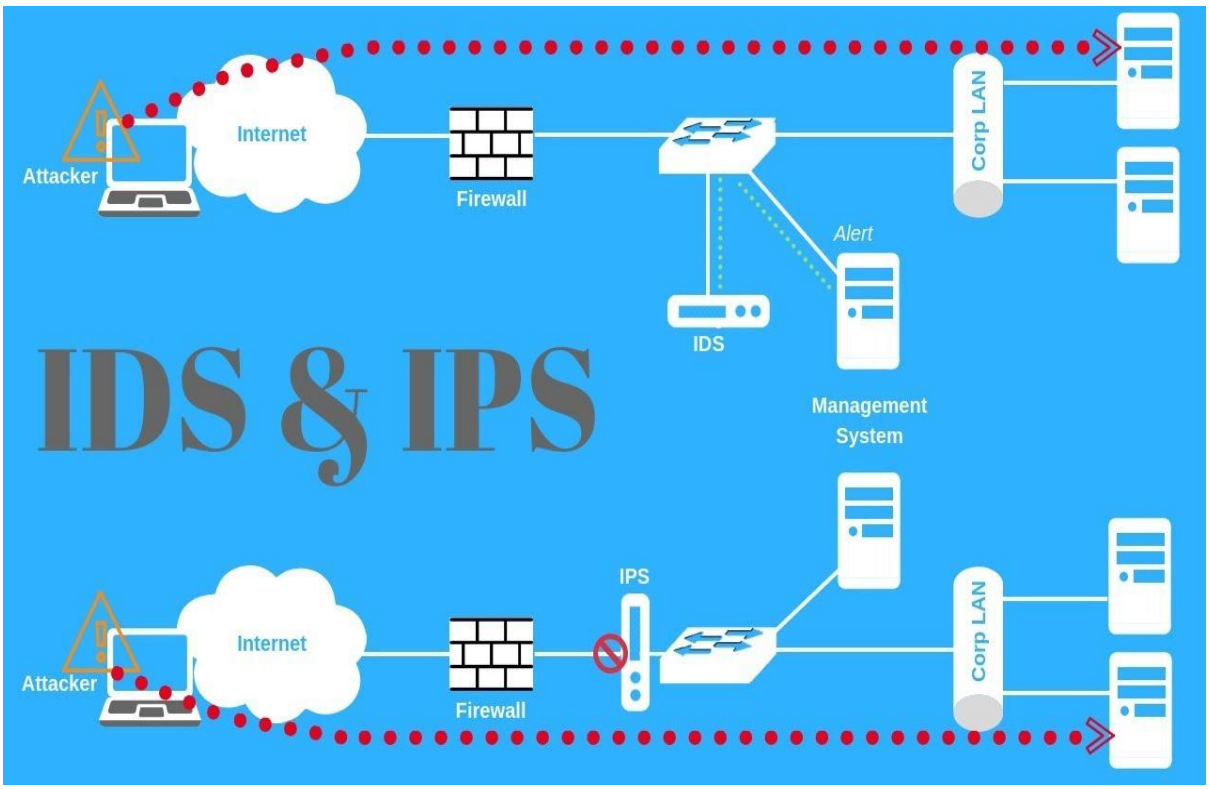

Рисунок 1.2 - Ахітектура системи IDS/IPS

Захист переданих даних є ще одним важливим аспектом кібербезпеки. Методи шифрування дозволяють зберігати конфіденційність інформації навіть у разі її перехоплення. Алгоритми, такі як AES і RSA , забезпечують високий рівень захисту, завдяки чому їх використовують у фінансових установах, урядових агенціях і великих корпораціях. Проте з розвитком квантових обчислень з’являються нові виклики: існує загроза, що сучасні алгоритми шифрування можуть стати менш надійними, тому фахівці з кібербезпеки працюють над створенням нових методів, стійких до квантових атак[3].

В останні роки штучний інтелект став важливим інструментом у боротьбі зі шкідливим ПЗ. Системи, що базуються на ШІ, аналізують дані в реальному часі і виявляють загрози на основі аналізу поведінки користувачів і додатків (рисунок 1.3). Це дозволяє виявляти навіть ті загрози, які раніше не зустрічалися. Наприклад, ШІ може розпізнавати

фішингові атаки, аналізуючи текст електронних листів і виявляючи невідповідності, які можуть свідчити про шахрайство.

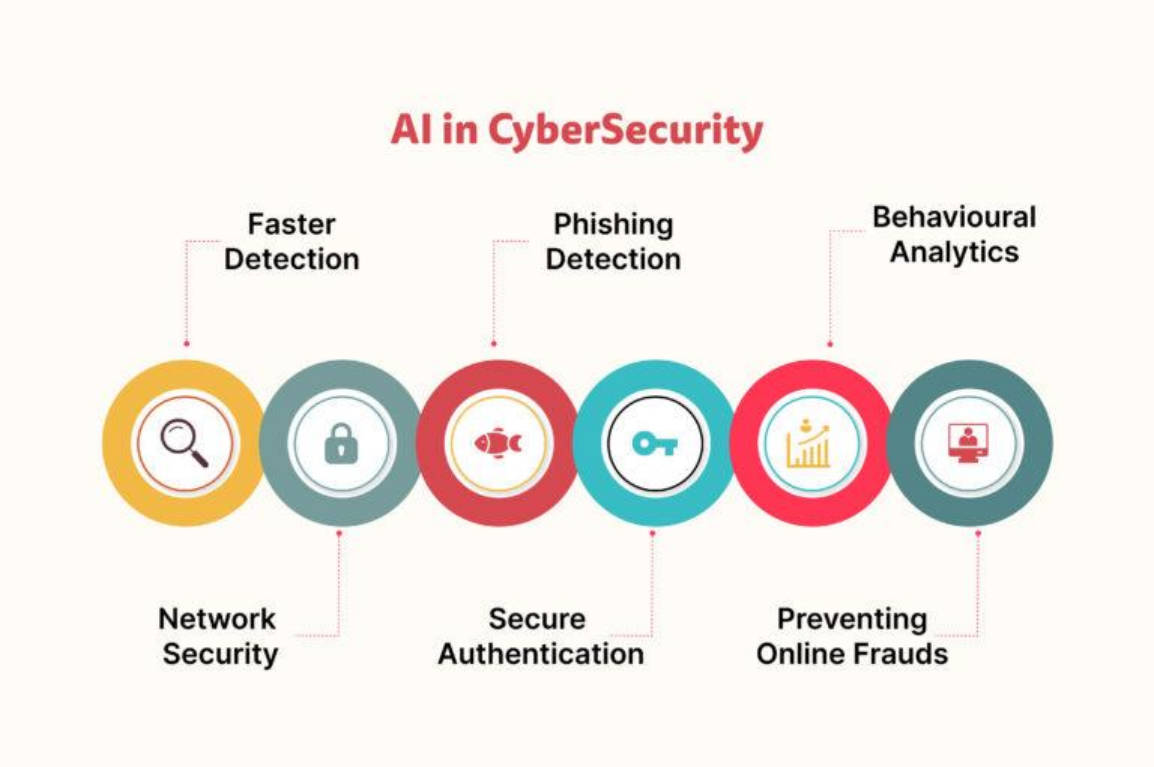

Рисунок 1.3 - Приклад використання ШІ в системах кібербезпеки

Також ШІ використовується для створення профілів користувачів, що допомагає виявляти спроби несанкціонованого доступу. Однак важливо враховувати, що впровадження таких технологій потребує значних ресурсів і високої кваліфікації фахівців. Отже, сучасні засоби протидії шкідливому програмному забезпеченню є надзвичайно важливими для захисту інформаційних систем[4]. Вони базуються на використанні комплексних рішень, що поєднують антивірусні програми, міжмережеві екрани, системи IDS/IPS та сучасні методи шифрування. Розвиток штучного інтелекту і машинного навчання відкриває нові можливості для боротьби з кіберзагрозами, однак також ставить нові виклики, з якими необхідно справлятися. Завдяки цьому ми можемо бачити, як кібербезпека стає все більш важливою галуззю, яка постійно розвивається, щоб відповідати на нові виклики і загрози.

1.2 Антивірусне програмне забезпечення

Антивірусне програмне забезпечення (АПЗ) є ключовим елементом у забезпеченні кібербезпеки сучасних інформаційних систем[5]. Його основна функція полягає у виявленні, нейтралізації та запобіганні поширенню шкідливого програмного забезпечення (ШПЗ), такого як віруси, трояни, черв'яки та інші зловмисні програми (Рисунок 1.4). Еволюція АПЗ відображає постійне протистояння між розробниками захисних рішень та творцями шкідливого ПЗ, що стимулює розвиток нових технологій та методів захисту.

Шкідливе програмне забезпечення, відоме як «malware», охоплює широкий спектр програм, створених для завдання шкоди комп'ютерам, мережам або користувачам. Це можуть бути програми, які працюють непомітно, викрадаючи конфіденційну інформацію, або ж такі, що безпосередньо порушують роботу системи. Наприклад, деякі з них впроваджуються у вигляді вірусів, які самовідтворюються і заражають інші файли, створюючи ланцюгову реакцію вразливостей. Інші види діють як троянські коні, маскуючись під корисні програми, але після встановлення починають виконувати зловмисні дії, наприклад, надають доступ до пристрою стороннім особам[6].

Існують також програми, які спеціалізуються на вимаганні коштів, шифруючи дані користувача і обіцяючи їх відновлення лише після сплати викупу. Інший тип шкідливого ПЗ орієнтований на шпигунство, збір паролів, банківських даних чи особистої інформації для подальшого продажу. Хоча технології безпеки постійно вдосконалюються, кіберзловмисники також знаходять нові способи обману користувачів, використовуючи соціальну інженерію чи вразливості в програмах. Боротися з такими загрозами можливо лише шляхом підвищення обізнаності та використання сучасних інструментів кіберзахисту.

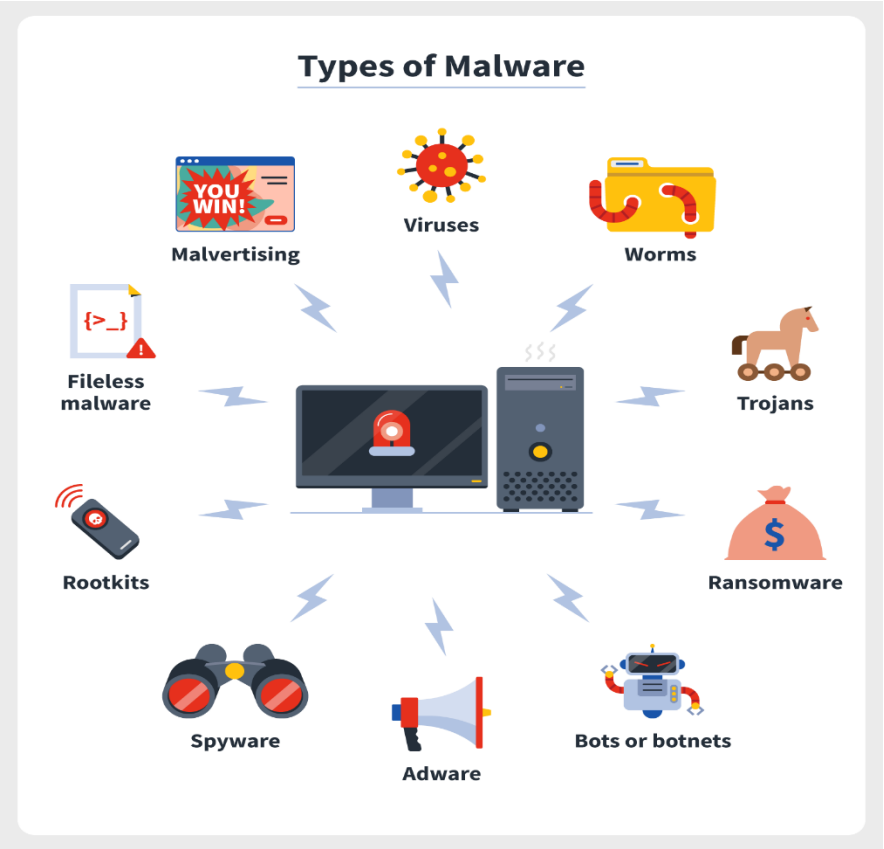

Рисунок 1.4 - Різні типи шкідливого програмного забезпечення, наприклад, віруси, трояни, черв'яки

Перші антивірусні програми з'явилися наприкінці 1980-х років у відповідь на зростання кількості комп'ютерних вірусів. Ці ранні рішення базувалися на сигнатурному аналізі, тобто порівнянні коду файлів із базою даних відомих вірусів[7]. Такий підхід був ефективним для виявлення вже відомих загроз, але не міг забезпечити захист від нових або модифікованих вірусів. З розвитком технологій та збільшенням складності шкідливого ПЗ виникла потреба у вдосконаленні методів виявлення.

У 1990-х роках з'явилися поліморфні віруси, які змінювали свій код при кожному зараженні, ускладнюючи їх виявлення за допомогою традиційних сигнатурних методів[8]. У відповідь на це розробники антивірусного ПЗ почали впроваджувати евристичний аналіз, що дозволяв виявляти потенційно шкідливі програми на основі аналізу їхньої поведінки та структури. Цей підхід значно підвищив ефективність виявлення нових загроз, але також призвів до збільшення кількості хибних спрацьовувань.

З початку 2000-х років антивірусні рішення почали інтегруватися з іншими засобами забезпечення безпеки, такими як міжмережеві екрани, системи виявлення та запобігання вторгненням (IDS/IPS) та інші. Це дозволило створити комплексні рішення, що забезпечують багаторівневий захист інформаційних систем. Крім того, з розвитком хмарних технологій антивірусні програми отримали можливість оперативно оновлювати свої бази даних та аналізувати загрози в реальному часі, що значно підвищило їх ефективність.

Сучасні антивірусні рішення використовують поєднання різних методів виявлення шкідливого ПЗ, включаючи сигнатурний аналіз, евристичний аналіз, поведінковий аналіз та методи машинного навчання. Сигнатурний аналіз залишається основним методом для виявлення відомих загроз, тоді як евристичний та поведінковий аналізи дозволяють виявляти нові та модифіковані загрози на основі аналізу їхньої поведінки та характеристик[9]. Методи машинного навчання, у свою чергу, дозволяють антивірусним програмам самостійно навчатися та вдосконалювати свої алгоритми виявлення, що підвищує їх ефективність та адаптивність до нових загроз.

Важливим аспектом сучасних антивірусних рішень є їхня здатність працювати в реальному часі, забезпечуючи постійний моніторинг системи та оперативне реагування на потенційні загрози я. Це досягається завдяки використанню технологій хмарних обчислень, що дозволяють антивірусним програмам отримувати актуальну інформацію про нові загрози та оновлювати свої бази даних у режимі реального часу (рисунок 1.5). Крім того, сучасні антивірусні рішення часто включають додаткові функції, такі як захист від фішингу, спаму[10], а також засоби для забезпечення конфіденційності та безпеки персональних даних користувачів.

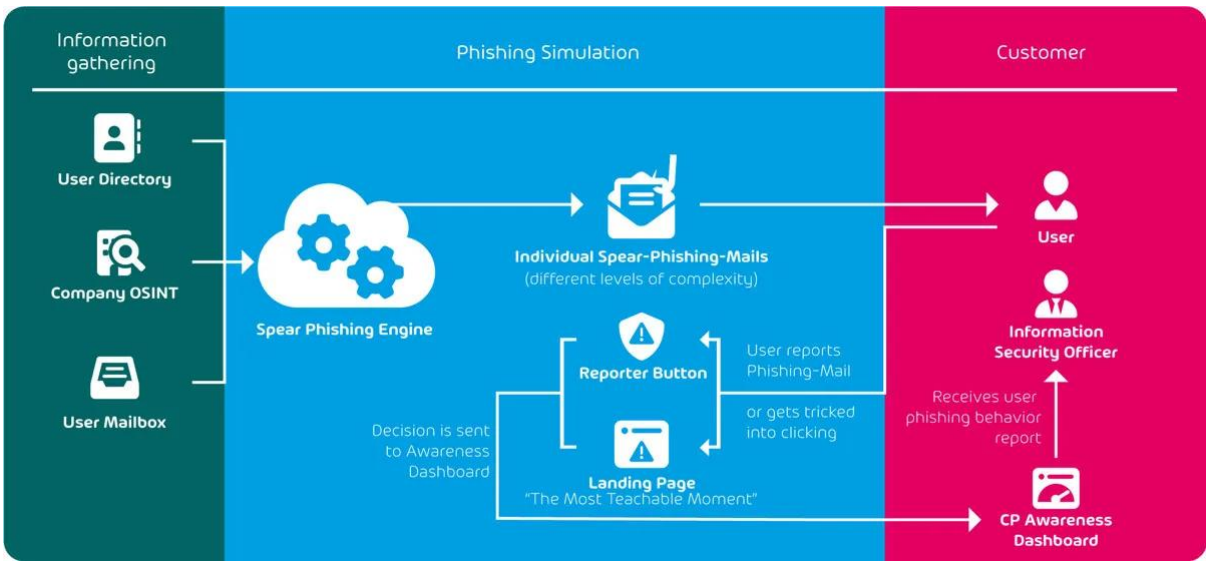

Рисунок 1.5 - Процес оновлення антивірусних баз у хмарі та приклади захисту від фішингу і спаму

Отже, еволюція антивірусного програмного забезпечення відображає постійне вдосконалення методів та технологій захисту інформаційних систем від шкідливого ПЗ. Сучасні антивірусні рішення поєднують у собі різноманітні методи виявлення загроз, забезпечують роботу в реальному часі та інтегруються з іншими засобами забезпечення безпеки, що дозволяє ефективно протидіяти сучасним кіберзагрозам та забезпечувати надійний захист інформаційних систем.

1.3 Міжмережеві екрани як важливий елемент безпеки

Міжмережеві екрани (рисунок1.6), відомі також як брандмауери або фаєрволи, є невід'ємною складовою сучасних систем кібербезпеки. Вони виконують функцію бар'єра між внутрішньою мережею організації та зовнішніми мережами, такими як Інтернет[11], контролюючи та фільтруючи мережевий трафік відповідно до встановлених правил безпеки. Це дозволяє запобігати несанкціонованому доступу, захищати від шкідливого програмного забезпечення та забезпечувати цілісність і конфіденційність даних.

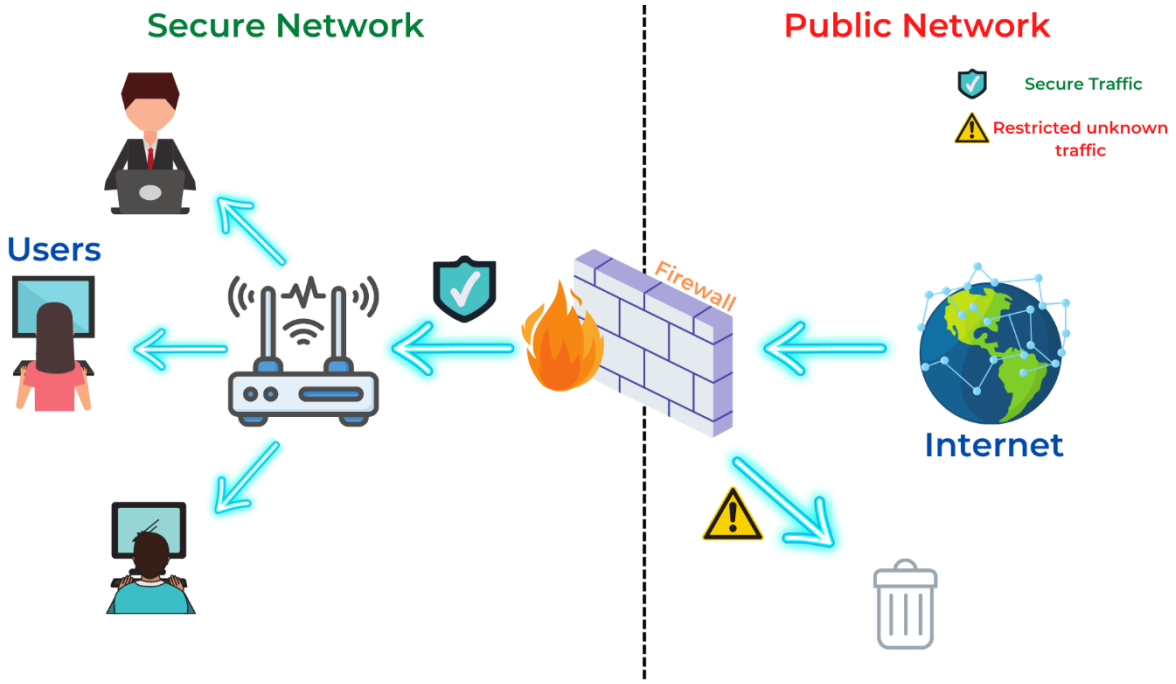

Рисунок 1.6 - Функціонування міжмережевого екрану

Історично міжмережеві екрани з'явилися у відповідь на зростання кількості мережевих загроз та необхідність захисту інформаційних систем. Перші фаєрволи були простими фільтрами пакетів, які перевіряли заголовки мережевих пакетів і приймали рішення про їх пропуск або блокування на основі IP-адрес та номерів портів. Такий підхід був ефективним на початкових етапах розвитку мереж, але з часом, зі збільшенням складності атак та появою нових загроз, виникла потреба у вдосконаленні методів захисту[12].

Сучасні міжмережеві екрани поділяються на кілька типів залежно від їх функціональних можливостей та рівня роботи в моделі OSI (рисунок 1.7). Пакетні фільтри працюють на мережевому рівні та аналізують заголовки пакетів, приймаючи рішення на основі IP-адрес, портів та протоколів. Шлюзи сеансового рівня (stateful inspection firewalls) відстежують стан з'єднань і дозволяють або блокують пакети на основі інформації про активні сесії. Шлюзи прикладного рівня (proxy firewalls) працюють на рівні додатків, аналізуючи вміст трафіку та забезпечуючи

більш глибоку перевірку даних[13]. Крім того, існують міжмережеві екрани нового покоління (Next-Generation Firewalls, NGFW), які поєднують функції традиційних фаєрволів з можливостями виявлення та запобігання вторгненням, аналізу додатків та контролю користувачів.

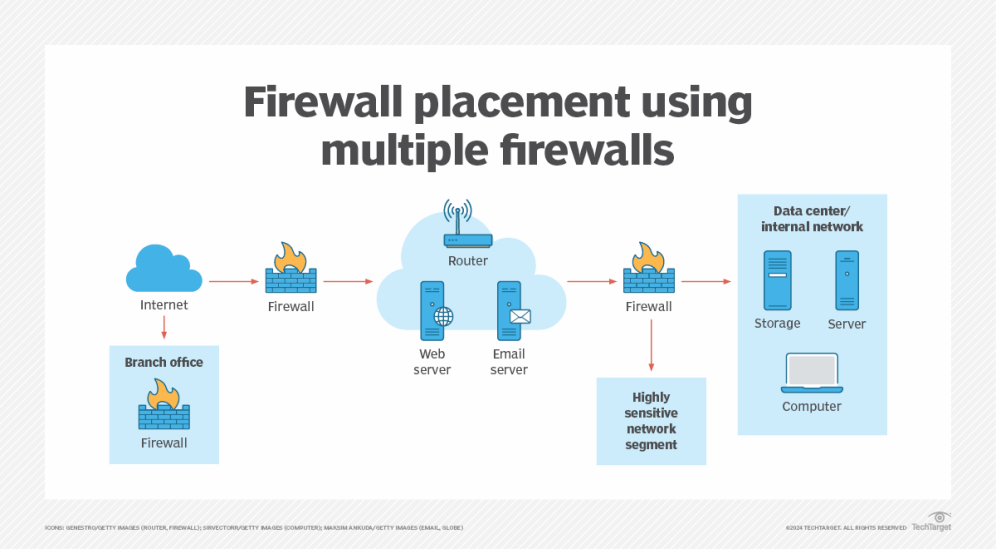

Рисунок 1.7 - Різні рівні роботи фаєрволів

Використання міжмережевих екранів забезпечує кілька ключових переваг. По-перше, вони контролюють вхідний та вихідний трафік, запобігаючи несанкціонованому доступу до внутрішніх ресурсів мережі. По-друге, фаєрволи можуть блокувати шкідливий трафік, такий як віруси, черв'яки та інші загрози, захищаючи систему від компрометації. По-третє, міжмережеві екрани дозволяють реалізувати політики безпеки, обмежуючи доступ до певних ресурсів або сервісів на основі ролей користувачів або інших критеріїв.

У сучасних умовах, коли атаки стають все більш складними та цілеспрямованими, традиційні методи захисту можуть бути недостатніми. Тому міжмережеві екрани нового покоління (NGFW) набувають все більшого поширення (рисунок 1.8). Вони поєднують функції традиційних фаєрволів з можливостями глибокого аналізу трафіку, виявлення та запобігання вторгненням, а також контролю додатків та користувачів.

NGFW здатні аналізувати трафік на рівні додатків, виявляти шкідливі дії та автоматично блокувати загрози в реальному часі. Крім того, вони можуть інтегруватися з іншими системами безпеки, такими як антивірусні програми та системи виявлення вторгнень, забезпечуючи комплексний підхід до захисту інформаційних систем.

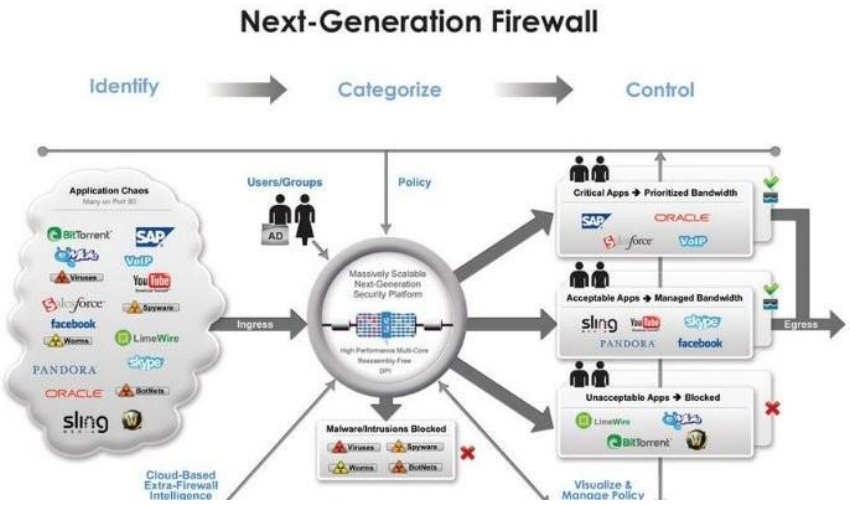

Рисунок 1.8 - Принцип роботи NGFW

Важливим аспектом використання міжмережевих екранів є їх правильне налаштування та управління. Неправильна конфігурація фаєрвола може призвести до пропуску загроз або, навпаки, до блокування легітимного трафіку, що негативно вплине на роботу організації. Тому адміністратори безпеки повинні ретельно розробляти та впроваджувати політики безпеки, регулярно оновлювати правила та моніторити роботу міжмережевих екранів. Крім того, важливо проводити регулярні аудити та тестування системи безпеки, щоб виявляти та усувати потенційні вразливості[14].

Отже, міжмережеві екрани є важливим елементом сучасних систем кібербезпеки, забезпечуючи захист від несанкціонованого доступу, шкідливого трафіку та інших загроз. Еволюція фаєрволів від простих пакетних фільтрів до міжмережевих екранів нового покоління відображає

постійне вдосконалення методів захисту та адаптацію до нових викликів у сфері кібербезпеки. Правильне впровадження та управління міжмережевими екранами дозволяє організаціям ефективно захищати свої інформаційні ресурси та забезпечувати безперебійну роботу бізнес-процесів.

1.4 Постановка завдання

Магістерська робота на тему «Алгоритми виявлення та захисту від зловмисного програмного забезпечення» має на меті дослідження актуальних методів і підходів до протидії шкідливому програмному забезпеченню (malware) у сучасних комп'ютерних системах[15]. Сьогодні зловмисне програмне забезпечення стає дедалі витонченішим, що створює значні виклики для забезпечення безпеки інформаційних технологій та захисту користувацьких даних. Постійне зростання кількості кіберзагроз зумовлене поширенням інтернет-зв’язків, розвитком Інтернету речей (IoT) та збільшенням кількості мобільних пристроїв. Це вимагає застосування новітніх підходів для своєчасного виявлення і нейтралізації шкідливого ПЗ. Однією з найважливіших причин актуальності даного дослідження є зростаюча залежність сучасного суспільства від цифрових технологій та зростаючий вплив кібератак на державні й приватні організації. Успішні атаки можуть призводити до значних фінансових втрат, витоку конфіденційної інформації, порушення функціонування критичних інфраструктур та інших небажаних наслідків. Традиційні методи захисту, такі як антивірусні програми та сигнатурний аналіз, вже не можуть забезпечити необхідного рівня безпеки, оскільки сучасні зловмисники використовують техніки обходу, як-от обфускація коду та поліморфізм. У цьому контексті впровадження інноваційних алгоритмів виявлення стає вкрай необхідним[16].

Метою дослідження є розробка та оцінка ефективності алгоритмів, які можуть забезпечити надійний захист систем від зловмисного програмного забезпечення. Основна увага буде зосереджена на підходах, які використовують машинне навчання для виявлення аномалій та аналізу поведінки програм. Такий підхід дозволяє підвищити швидкість і точність виявлення шкідливих дій навіть у разі, коли зловмисне ПЗ намагається маскуватися під легітимні процеси або адаптуватися до середовища. Для досягнення цієї мети необхідно виконати кілька завдань. Перш за все, буде проведено огляд сучасних методів виявлення шкідливого програмного забезпечення, що базуються на аналізі сигнатур, поведінковому аналізі та методах машинного навчання[17]. Наступним кроком буде аналіз сильних і слабких сторін існуючих підходів, щоб визначити потенційні напрями вдосконалення. У межах дослідження передбачається створення прототипу системи, яка використовуватиме алгоритми глибокого навчання для виявлення зловмисного ПЗ, а також її тестування на реальних та синтетичних даних для оцінки продуктивності та стійкості до різних методів обходу[18].

Методологія роботи включає аналіз наукової літератури, вивчення існуючих технологій і програмних рішень, а також розробку власних алгоритмічних підходів із застосуванням інструментів машинного навчання. Для тестування та валідації запропонованих алгоритмів будуть використовуватися спеціальні датасети, зокрема ті, що містять зразки шкідливого ПЗ і його поведінкові особливості. Велику увагу буде приділено вибору оптимальних параметрів моделей, а також їх здатності швидко адаптуватися до нових видів атак[19]. Очікуваними результатами є створення ефективного механізму виявлення шкідливого програмного забезпечення з високим рівнем точності і швидкості роботи. Отримані результати можуть бути використані для вдосконалення існуючих систем кібербезпеки та надання рекомендацій щодо впровадження захисних механізмів у корпоративних та державних структурах. У перспективі

можливе розширення запропонованих алгоритмів для роботи з великими обсягами даних у реальному часі, що підвищить їхню практичну цінність.

2 ТЕХНІКИ ДОСЛІДЖЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Статичний аналіз програмного забезпечення

У сучасному світі інформаційних технологій шкідливе програмне забезпечення (ШПЗ) становить серйозну загрозу для безпеки комп'ютерних систем та мереж. Для ефективної протидії цим загрозам розроблено різноманітні методи та інструменти, які можна класифікувати за кількома напрямками[20].

Основним інструментом для статичного аналізу є спеціалізовані аналізатори коду (Рисунок 2.1), які працюють із вихідним кодом, або ж із байт-кодом, якщо вихідний код недоступний. Такі аналізатори перевіряють, чи відповідає код встановленим правилам та стандартам. Зазвичай, результатом є звіт про знайдені помилки та підозрілі ділянки коду. Наприклад, якщо в програмі є рядки, що викликають підозру на шкідливу активність, такі як незахищені функції доступу до пам’яті чи необроблені винятки, це може свідчити про потенційну загрозу.

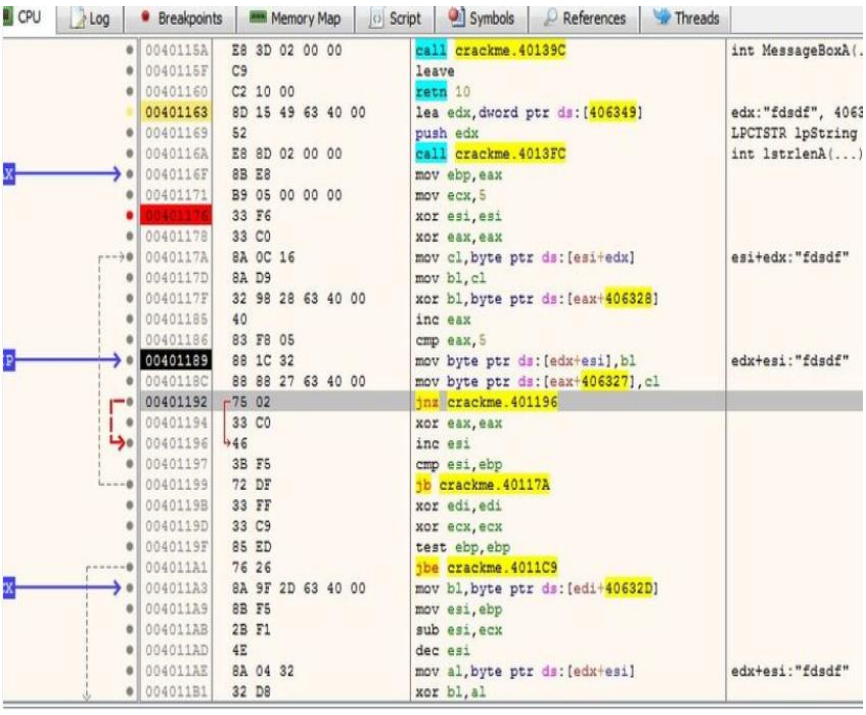

Рисунок 2.1 - Типова програма статичного аналізу

Статичний аналіз програмного забезпечення є одним з найважливіших методів дослідження шкідливих програм і відіграє ключову роль у сфері кібербезпеки[21]. Його основна суть полягає в аналізі вихідного або скомпільованого коду програми без її фактичного виконання. За допомогою цього підходу можна виявити шкідливі елементи ще на етапі розробки, що дозволяє завчасно запобігти їх розповсюдженню.

Одним з найвідоміших інструментів статичного аналізу є IDA Pro (Рисунок 2.2), який широко використовується у сфері реверс-інжинірингу. Цей інструмент дозволяє проводити детальний аналіз коду, навіть якщо він вже скомпільований у машинні інструкції. Завдяки своєму потужному дизасемблеру IDA Pro перетворює низькорівневий код у зрозумілий формат, що допомагає фахівцям краще зрозуміти структуру та логіку шкідливих програм[22]. Наприклад, за допомогою цього інструменту можна дослідити, які функції викликає програма, чи є у ній приховані API-запити, які можуть свідчити про збір даних або підключення до командних серверів.

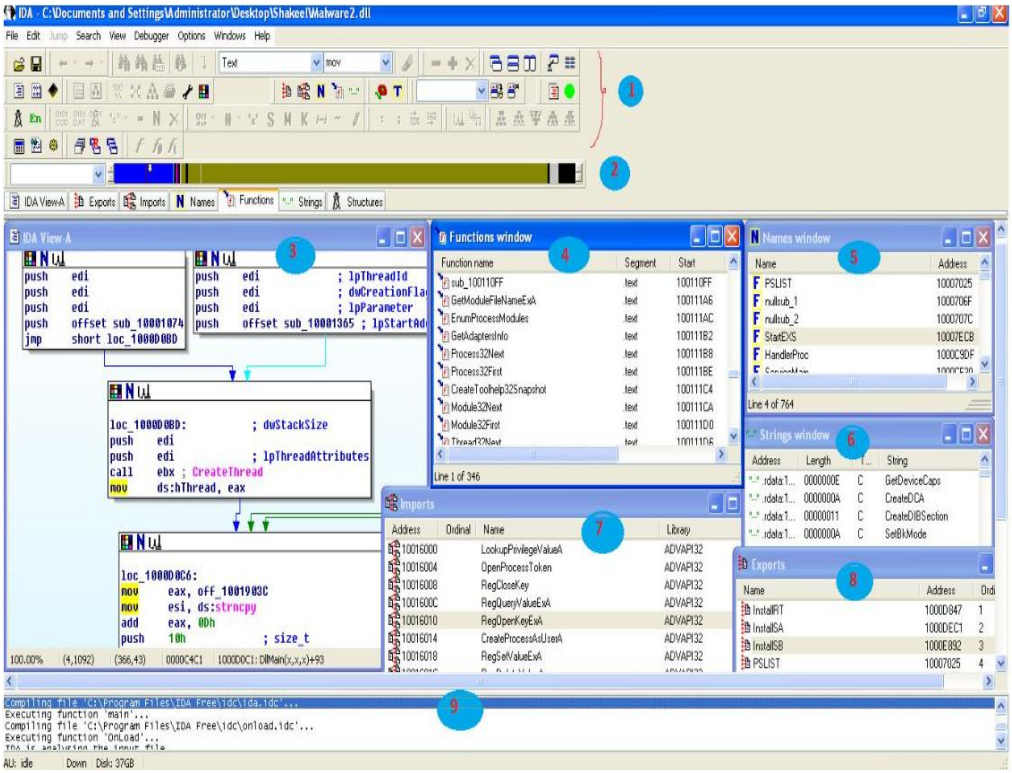

Рисунок 2.2 - Програма IDA Pro

Ще одним популярним інструментом є Ghidra, розроблений Агентством національної безпеки США (NSA). Ghidra є відкритим програмним забезпеченням і має потужні можливості для аналізу. Як і IDA Pro, Ghidra дозволяє дизасемблювати код, а також має функціонал для автоматичного визначення структур даних, що використовується в коді, і побудови схеми взаємодії функцій[23]. Це допомагає краще розуміти, які саме операції виконуються і як вони пов’язані між собою, що може бути корисним для виявлення прихованої шкідливої активності. У випадку, коли програма містить алгоритми шифрування для приховування своїх дій, Ghidra дозволяє ідентифікувати і розшифрувати ці алгоритми для подальшого детального аналізу.

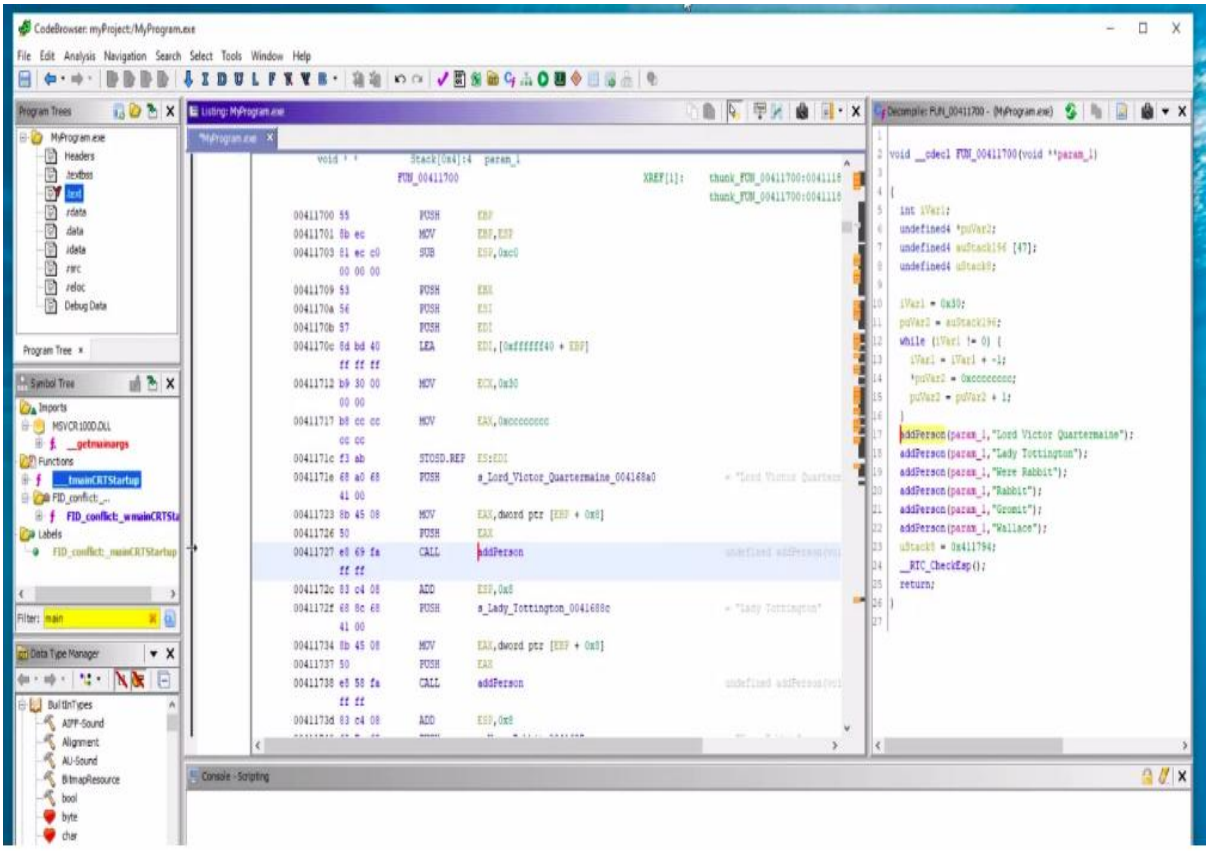

Рисунок 2.3 - Програма Ghidra

Крім того, у випадках, коли є доступ до вихідного коду, статичний аналіз може включати більш детальне дослідження самого коду. Наприклад, з використанням Checkmarx або SonarQube[24], можна

проаналізувати код на наявність специфічних вразливостей, таких як SQL-ін’єкції або XSS-атаки (Таблиця 2.1). Це особливо корисно для розробників, які можуть одразу виправити виявлені недоліки. Звіт з інструменту може містити список потенційних загроз з рекомендаціями щодо їх виправлення.

Таблиця 2.1 – Таблиця вразливостей

№	Файл	Рядок	Опис вразливості	Рівень серйозності	Рекомендації щодо виправлення
1	src/main/App.java	45	Використання незахищеного методу getRuntime(). exec() для виконання команд	Критичний	Використовуйте безпечніші альтернативи, такі як ProcessBuilder, та перевіряйте вхідні дані.
2	src/main/Database.java	102	SQL-ін'єкція через некоректну обробку користувацького вводу в SQL-запиті	Високий	Використовуйте підготовлені вирази (prepared statements) для запобігання SQL-ін'єкціям.
3	src/main/Utils.java	78	Потенційна втрата пам'яті через незакритий ресурс FileInputStream	Середній	Закривайте всі ресурси після використання, наприклад, за допомогою блоку try-with-resources.
4	src/main/Security.java	150	Використання застарілого алгоритму хешування MD5 для зберігання паролів	Високий	Перейдіть на сучасні алгоритми хешування, такі як SHA-256 або bcrypt.
5	src/main/Config.java	30	Жорстко закодовані облікові дані в коді	Середній	Зберігайте облікові дані у зовнішніх конфігураційних файлах або змінних середовища.

Таким чином, статичний аналіз є надзвичайно важливим інструментом у виявленні шкідливого ПЗ[25], оскільки дозволяє виявляти потенційні загрози без ризику виконання шкідливого коду. Цей метод зменшує ймовірність розповсюдження шкідливого програмного забезпечення, оскільки вразливі ділянки можуть бути ідентифіковані та виправлені на ранніх етапах.

2.2 Практики динамічного аналізу

Динамічний аналіз шкідливого програмного забезпечення (ШПЗ) є критично важливим етапом у дослідженні та розумінні поведінки зловмисних програм. На відміну від статичного аналізу, який зосереджується на вивченні коду без його виконання, динамічний аналіз передбачає запуск підозрілого програмного забезпечення в контрольованому середовищі з метою спостереження за його реальними діями. Такий підхід дозволяє виявити приховані функції, які можуть не бути очевидними при статичному аналізі, та зрозуміти, як саме ШПЗ взаємодіє з системою та мережею[26].

Одним із ключових інструментів для проведення динамічного аналізу є пісочниці (sandbox). Пісочниця — це ізольоване середовище, яке імітує реальну операційну систему, але не має зв'язку з основною системою чи мережею. Це дозволяє безпечно запускати та досліджувати шкідливі програми без ризику зараження реальної системи[27]. Наприклад, інструмент Cuckoo Sandbox(Рисунок 2.5) є популярним вибором серед дослідників безпеки. Він автоматично аналізує підозрілі файли та надає детальні звіти про їхню поведінку, включаючи створення нових процесів, зміну реєстру, мережеву активність та інші дії.

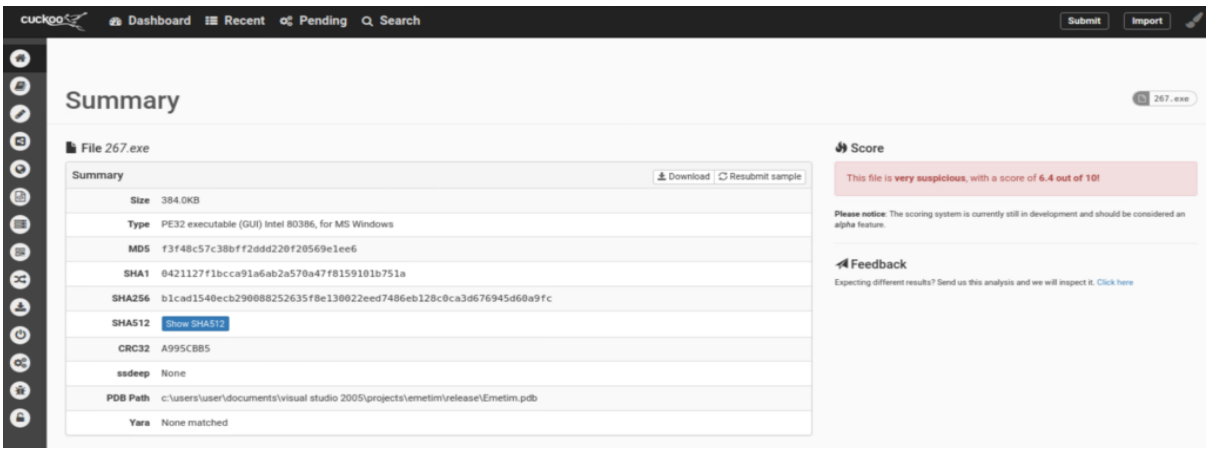

Рисунок 2.5 - Інтерфейс Cuckoo Sandbox із прикладом аналізу шкідливого файлу.

Під час динамічного аналізу важливо відстежувати мережеву активність ШПЗ[28]. Багато зловмисних програм намагаються встановити з'єднання з командно-контрольними серверами (C&C) для отримання інструкцій або передачі зібраних даних. Інструменти, такі як Wireshark(Рисунок 2.6), дозволяють аналізувати мережевий трафік та виявляти підозрілі з'єднання. Наприклад, якщо під час аналізу виявлено, що програма намагається підключитися до невідомого зовнішнього сервера, це може свідчити про спробу передачі конфіденційної інформації.

Cuckoo Sandbox — це популярна система для аналізу шкідливого програмного забезпечення в ізольованому середовищі. Її основна функція полягає у створенні безпечного простору, де можна виконати підозрілий файл або програму, не ризикуючи порушенням роботи основної системи. Платформа ретельно відстежує всі дії програми: зміни в реєстрі, файловій системі, поведінку в мережі та взаємодію з іншими процесами[29]. Завдяки цьому дослідники отримують повну картину того, як шкідливе ПЗ діє, які методи обходу захисту застосовує і які можливі наслідки його роботи.

Гнучкість Cuckoo Sandbox дозволяє аналізувати різні типи файлів, зокрема виконувані файли, документи, скрипти, а також URL-адреси. Її можна інтегрувати з іншими системами для поглибленого аналізу та автоматизації процесів кіберзахисту. Важливою перевагою є відкритий

вихідний код, що дозволяє адаптувати платформу до конкретних потреб організації чи дослідника[30]. Завдяки цьому Cuckoo Sandbox широко використовується як у професійній кібербезпеці, так і в навчальних цілях для глибшого розуміння механізмів роботи шкідливого програмного забезпечення.

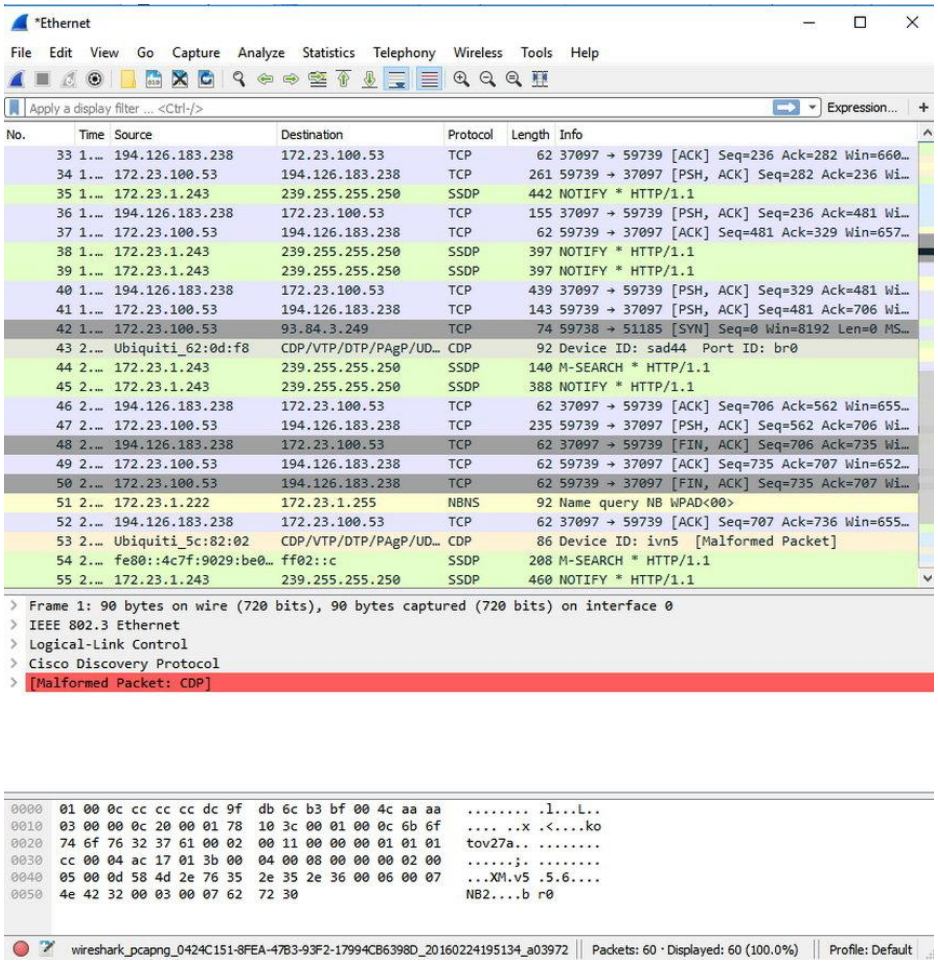

Рисунок 2.6 - Wireshark підозрілий мережевий трафік

Ще одним важливим аспектом динамічного аналізу є моніторинг змін у файловій системі та реєстрі. Шкідливі програми часто створюють або модифікують файли, а також вносять зміни до реєстру для забезпечення своєї стійкості в системі[31]. Інструменти, такі як Process Monitor від Sysinternals (рисунок 2.7), дозволяють відстежувати всі операції з файлами та реєстром у реальному часі. Наприклад, якщо під час аналізу виявлено, що програма створює файл у папці автозавантаження або

додає ключ до реєстру для автоматичного запуску при старті системи, це є явною ознакою шкідливої активності.

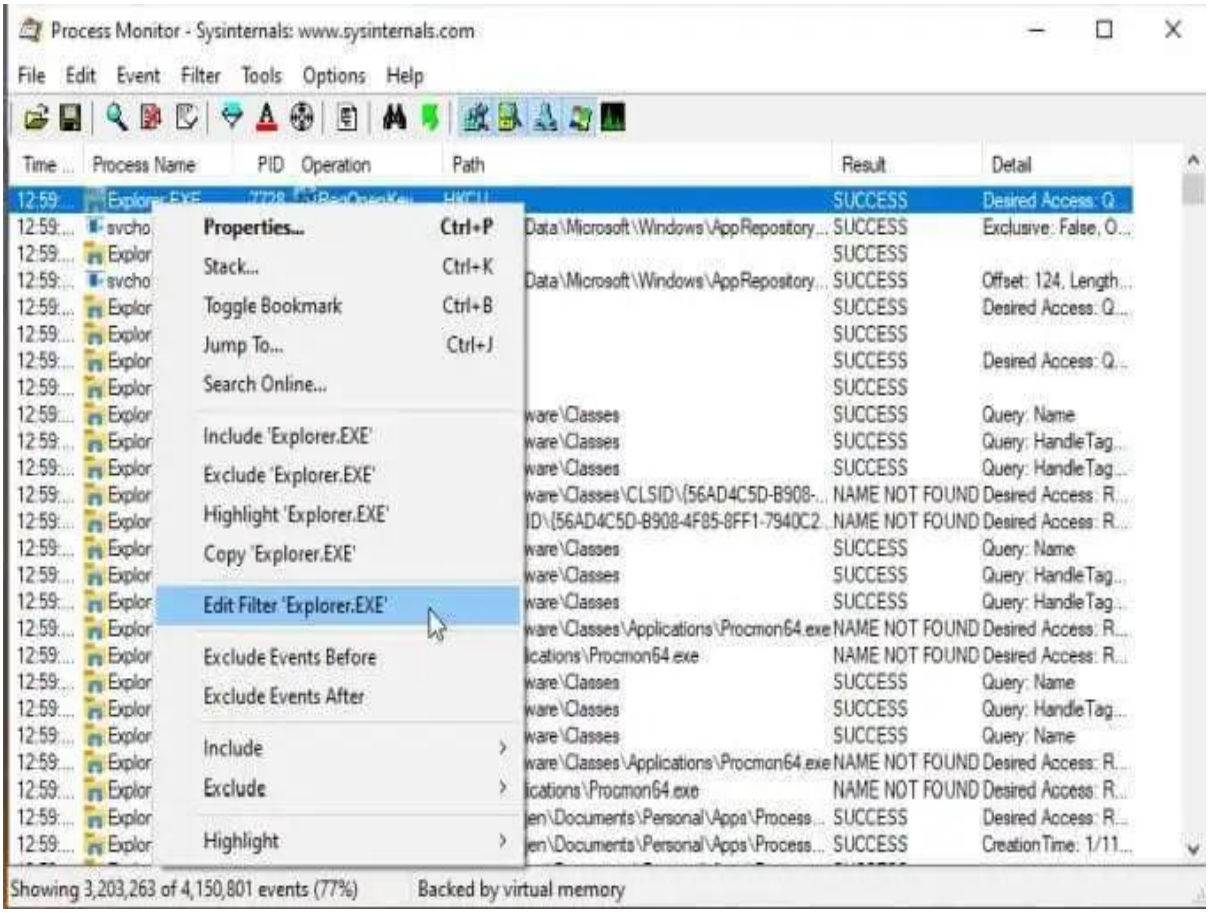

Рисунок 2.7 - Аналіз діяльності ШПЗ з допомогою Process Monitor

Динамічний аналіз також може включати відстеження поведінки процесів. Шкідливі програми можуть створювати нові процеси, інжектувати свій код у легітимні процеси або завершувати роботу антивірусних програм. Інструменти, такі як Process Explorer, дозволяють детально досліджувати всі активні процеси в системі, їхні взаємозв'язки та ресурси, які вони використовують[32]. Наприклад, якщо під час аналізу виявлено, що підозріла програма інжектує свій код у процес браузера (рисунок 2.8), це може свідчити про спробу перехоплення даних користувача.

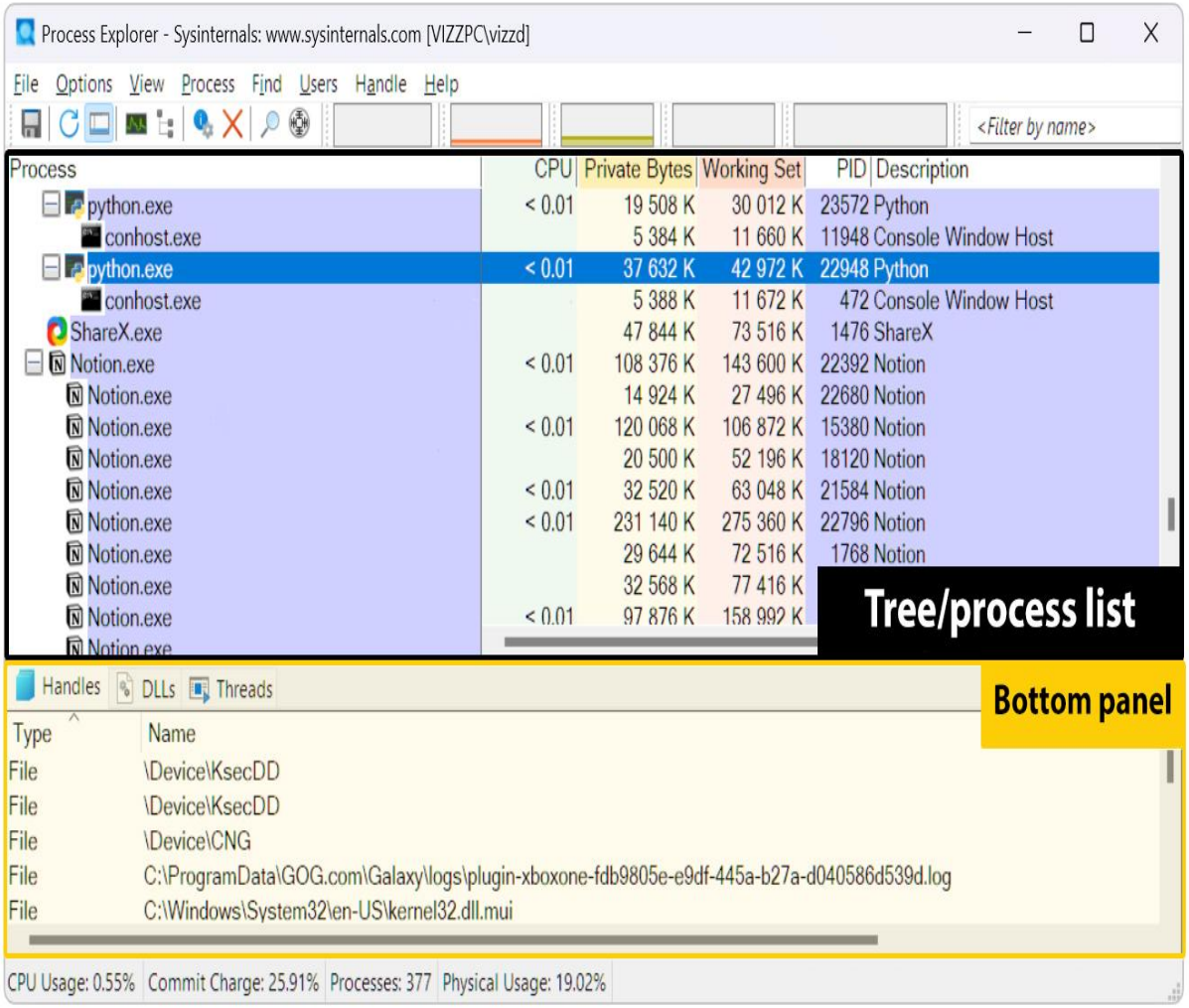

Рисунок 2.8 - Дослідження експлуатації XSS з допомогою Process Explorer

Важливо зазначити, що шкідливі програми можуть намагатися виявити, що вони виконуються в середовищі пісочниці або віртуальної машини(Рисунок 2.9), та змінювати свою поведінку, щоб уникнути детекції. Тому дослідники безпеки використовують різні техніки для маскування середовища аналізу, щоб отримати максимально точні результати. Наприклад, можна змінювати параметри віртуальної машини, щоб вона виглядала як реальна система, або використовувати інструменти для приховування ознак пісочниці.

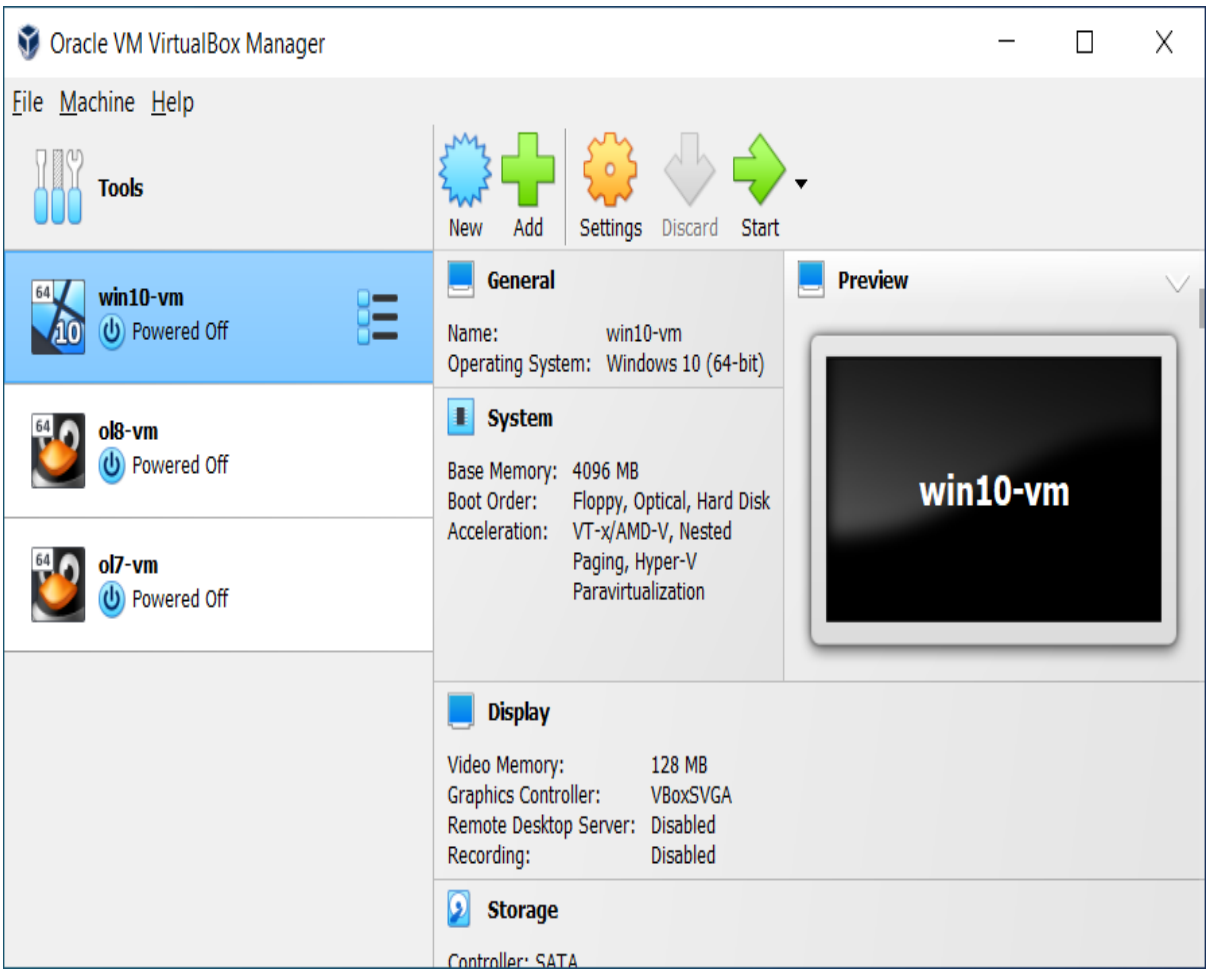

Рисунок 2.9 - Віртуальна машина virtual box

Динамічний аналіз є потужним інструментом для виявлення та розуміння шкідливого програмного забезпечення. Він дозволяє дослідникам спостерігати за реальною поведінкою програм, виявляти приховані функції та розробляти ефективні методи захисту. Однак цей підхід також має свої обмеження та ризики, тому його слід використовувати в поєднанні з іншими методами аналізу та дотримуватися всіх заходів безпеки під час проведення досліджень.

2.3 Сигнатурний аналіз

Сигнатурний аналіз є одним із фундаментальних методів виявлення шкідливого програмного забезпечення (ШПЗ) і широко використовується в антивірусних системах. Його суть полягає у порівнянні коду або поведінки

підозрілої програми з базою відомих сигнатур — унікальних послідовностей байтів або характеристик, притаманних конкретним зразкам ШПЗ. Якщо в коді програми виявляється відповідність з однією з сигнатур, програма вважається шкідливою.

Основною перевагою сигнатурного аналізу є його висока точність у виявленні відомих загроз. Однак цей метод має обмеження, оскільки не здатен ефективно виявляти нові або модифіковані шкідливі програми, які не мають відповідних записів у базі сигнатур. Тому сучасні антивірусні рішення часто поєднують сигнатурний аналіз з іншими методами, такими як евристичний аналіз та поведінковий моніторинг. Одним із потужних інструментів для створення та використання сигнатур є YARA (рисунок 2.10). YARA дозволяє дослідникам безпеки визначати правила для ідентифікації та класифікації шкідливих файлів на основі їхніх характеристик. Правила YARA складаються з набору умов, які можуть включати текстові або бінарні рядки, регулярні вирази та інші критерії. Наприклад, для виявлення шкідливої програми, яка використовує специфічний рядок коду, можна створити правило, що шукає цей рядок у файлах.

Yara — це інструмент, призначений для ідентифікації та класифікації шкідливого програмного забезпечення шляхом створення та використання спеціальних правил. Вона дозволяє описувати характеристики файлів чи програм у вигляді набору умов, які можуть включати текстові, бінарні або регулярні вирази. Ці правила застосовуються до аналізованих даних, допомагаючи дослідникам знаходити схожість між зразками шкідливого ПЗ, навіть якщо вони належать до різних версій або модифікацій однієї сім’ї.

Інструмент є надзвичайно корисним у боротьбі з сучасними кіберзагрозами завдяки своїй гнучкості та масштабованості. Його використовують для аналізу як окремих файлів, так і цілих систем, зокрема у поєднанні з платформами для аналізу шкідливого ПЗ, такими як Cuckoo

Sandbox. Yara також добре інтегрується з іншими інструментами кібербезпеки, що робить її важливою складовою у створенні комплексного захисту. Завдяки відкритому вихідному коду та активній спільноті користувачів, цей інструмент постійно вдосконалюється, забезпечуючи ефективність у виявленні навіть найскладніших загроз.

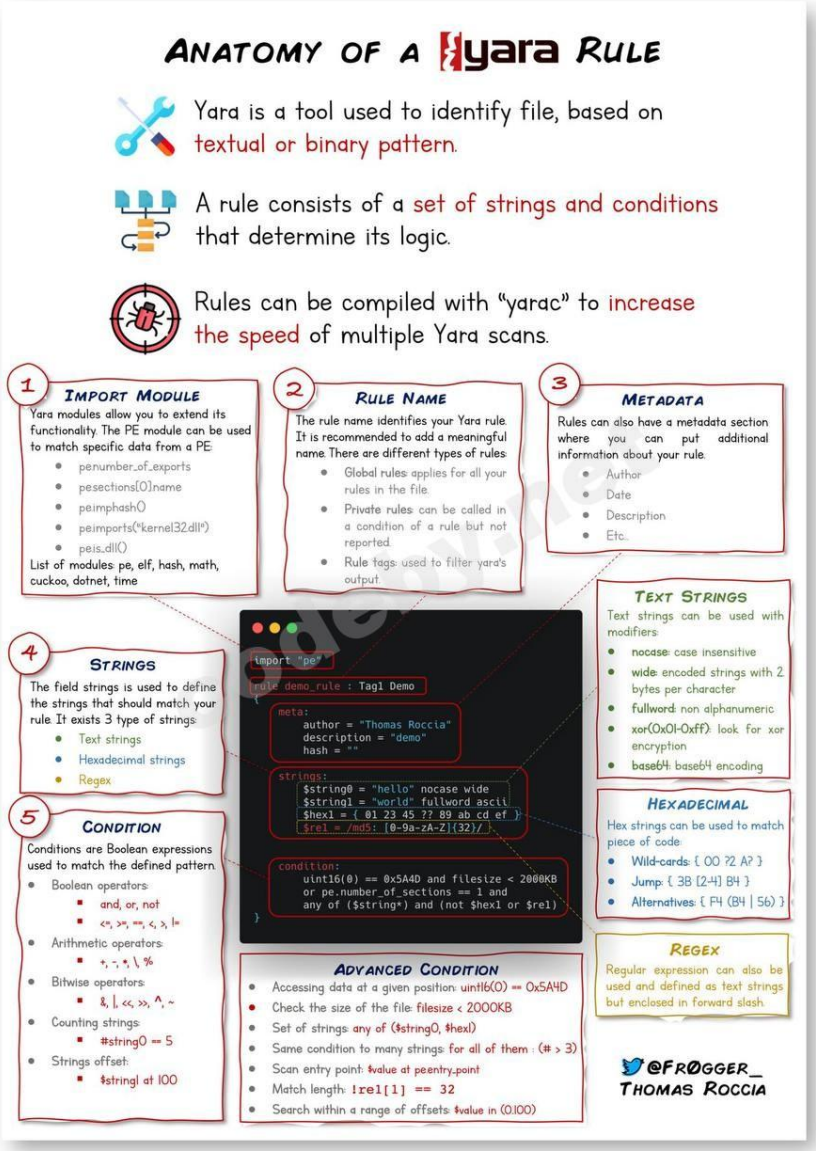

Рисунок 2.10 - Правила YARA

Розглянемо приклад використання YARA для виявлення шкідливого ПЗ. Припустимо, що відомо про існування шкідливої програми, яка

містить у своєму коді рядок "malicious_function". Для її виявлення можна створити наступне правило YARA:

Bash: rule DetectMaliciousFunction { strings: $a = "malicious_function"
condition: $a }

Це правило шукає вказаний рядок у файлах та сигналізує про його наявність.

Варто зазначити, що шкідливі програми часто використовують техніки обфускації та поліморфізму для уникнення детекції сигнатурними методами. Обфускація полягає у зміні коду таким чином, щоб ускладнити його аналіз, зберігаючи при цьому функціональність. Поліморфізм передбачає динамічну зміну коду при кожному виконанні або розповсюдженні, що робить кожен зразок унікальним. Для протидії таким технікам дослідники використовують розширені можливості YARA, зокрема регулярні вирази та умови, що враховують варіативність коду.

rule DetectObfuscatedCode {

strings:

$obfuscated_code = /eval\((unescape|atob|decodeURIComponent)\(.*\)\)/

condition:

$obfuscated_code }

Крім того, YARA дозволяє створювати правила, що базуються на метаданих файлів, таких як розмір, хеш-суми або специфічні атрибути. Це особливо корисно для виявлення шкідливих програм, які мають певні характерні ознаки, наприклад, специфічні розміри або структуру. Наприклад, можна створити правило, яке виявляє файли з певним розміром та хеш-сумою, що відповідають відомому зразку ШПЗ.

import "hash"

rule DetectSpecificMalware {

meta: description = "Виявлення файлів з відомим розміром та хеш-сумою"
author = "ім'я" date = "2024-11-18" condition: (filesize == 123456) and
(hash.md5(0, filesize) == "d41d8cd98f00b204e9800998ecf8427e") }

Сигнатурний аналіз, зокрема з використанням інструментів на кшталт YARA, залишається важливим компонентом у системах виявлення шкідливого програмного забезпечення. Однак для ефективного захисту необхідно поєднувати його з іншими методами аналізу та постійно оновлювати бази сигнатур, враховуючи швидкий розвиток та еволюцію загроз у сфері кібербезпеки.

Сигнатурний аналіз широко застосовується в антивірусних системах, таких як Kaspersky, Avast чи Norton, які постійно оновлюють свої бази даних для ефективної детекції нових загроз. Після того, як створене правило YARA виявляє шкідливу програму, антивірусний сканер може виконати подальші дії: видалити заражений файл, помістити його в карантин або повідомити користувача про загрозу. Проте навіть із цими заходами важливо розуміти, що сигнатурний аналіз є ефективним лише проти вже відомих зразків, тому нові або модифіковані загрози можуть залишатися непоміченими.

Існують також приклади, коли шкідливе програмне забезпечення використовує комбінації поліморфізму та метаморфізму, щоб значно ускладнити сигнатурний аналіз. Поліморфічне ПЗ змінює свій код кожного разу при виконанні або розповсюдженні, тоді як метаморфічне ПЗ переписує свій власний код, використовуючи різні методи шифрування або створення змінних структур. Через це стандартний сигнатурний підхід стає майже неефективним, що вимагає розробки складніших правил, здатних враховувати такі модифікації.

Наприклад, для боротьби з поліморфічними загрозами дослідники розробляють правила YARA, які аналізують не тільки рядки коду, але й послідовність функціональних викликів або структуру виконуваного файлу. У випадку зі складним ШПЗ, яке змінює свої байтові сигнатури, правила можуть включати умови, що враховують певні постійні характеристики, наприклад, використання конкретних системних функцій

або бібліотек. Це дозволяє підвищити ефективність виявлення навіть у разі обфускації.

```
rule DetectPolymorphicMalware {
    meta:
        description = "Виявлення поліморфного шкідливого ПЗ за допомогою багаторівневих умов"
        author = "ім'я"
        date = "2024-11-18"
    strings:
        $s1 = "malicious_function"
        $s2 = { E8 ?? ?? ?? ?? 83 C4 04 }
        $s3 = /[a-zA-Z0-9]{8,}/
    condition:
        (uint16(0) == 0x5A4D) and
        (filesize < 500000) and
        (
            ($s1 and $s2) or
            ($s3 and pe.imphash() == "d41d8cd98f00b204e9800998ecf8427e")
        )
```

Іншим прикладом вдосконалення сигнатурного аналізу є застосування зворотного інжинірингу для детального вивчення шкідливих програм. За допомогою таких інструментів, як Ghidra або IDA Pro, фахівці можуть розбирати виконувані файли, знаходити критичні функції та створювати спеціалізовані сигнатури для подальшого виявлення. Наприклад, під час аналізу програмного забезпечення, яке виконує специфічні API-запити для обходу систем захисту (рисунок 2.11), можна створити унікальну сигнатуру на основі цих викликів.

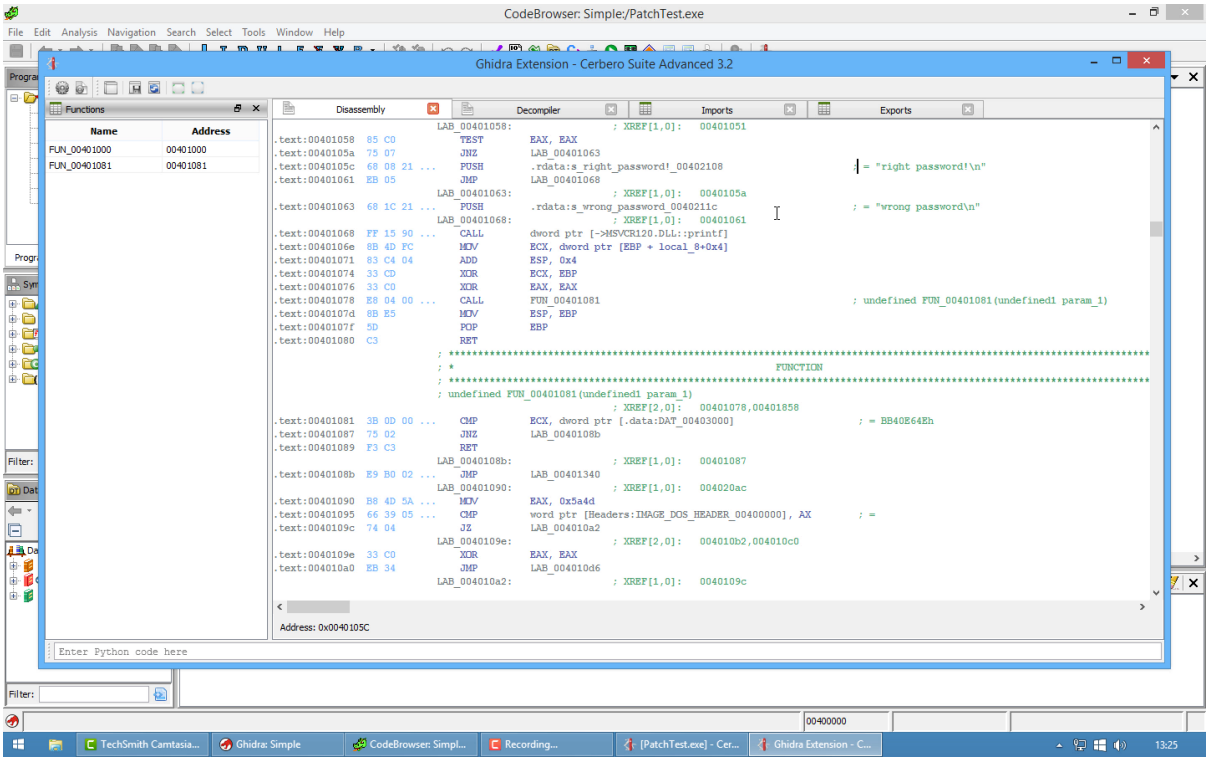

Рисунок 2.11 - Інтерфейс Ghidra в процесі дослідження

Додатково, в деяких випадках сигнатурний аналіз може бути поєднаний із контекстним аналізом. Це дозволяє враховувати не тільки вміст файлів, але й їхнє розташування, взаємодію з іншими компонентами системи та загальний контекст виконання. Наприклад, файл, який на перший погляд виглядає безпечним, але знаходиться у підозрілій директорії або взаємодіє з відомими шкідливими процесами, може бути ідентифікований як загроза. Такі комбіновані підходи значно підвищують ефективність захисту інформаційних систем.

У сучасних антивірусних рішеннях сигнатурний аналіз також тісно пов’язаний із системами машинного навчання. Алгоритми аналізують великі обсяги даних та допомагають автоматично створювати нові сигнатури або вдосконалювати існуючі. Ці алгоритми здатні ідентифікувати нетипову поведінку програм, що може свідчити про потенційну загрозу. Таким чином, антивірусні системи стають більш гнучкими та швидко адаптуються до нових типів шкідливих програм.

Незважаючи на розвиток технологій, сигнатурний аналіз залишається одним із ключових методів виявлення загроз у сфері кібербезпеки. Він надає швидкі та точні результати у разі виявлення відомих зразків, але для забезпечення повного захисту його завжди слід використовувати у поєднанні з іншими методами аналізу. Саме тому сучасні системи захисту використовують багаторівневий підхід, що поєднує сигнатурний, евристичний, поведінковий аналіз та новітні технології штучного інтелекту, забезпечуючи комплексний захист від постійно зростаючих загроз у цифровому середовищі.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ЗАСОБУ ВИЯВЛЕННЯ ШПЗ

3.1 Проектування програмного засобу

Проектування програмного засобу для виявлення шкідливого програмного забезпечення є ключовим етапом розробки, який визначає функціональність, архітектуру та основні принципи роботи системи(Рисунок 3.1). Основною метою цього етапу було створення ефективного та гнучкого інструменту, який здатний адаптуватися до постійно зростаючих кіберзагроз. У процесі проектування я враховував як технічні, так і користувацькі вимоги, оскільки система має бути не лише функціональною, а й зручною для кінцевого користувача.

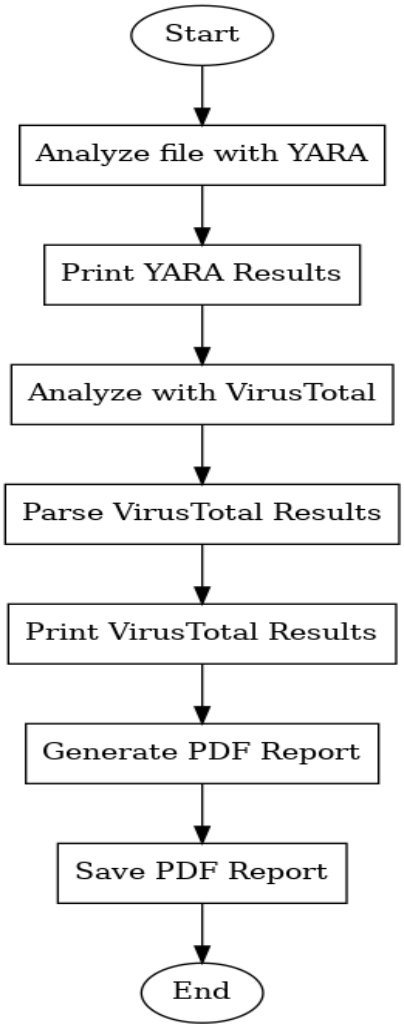

Рисунок 3.1 - Алгоритм роботи програми

Розробка почалася з аналізу можливостей сучасних інструментів виявлення шкідливого ПЗ. Одним із таких інструментів є YARA, який дозволяє створювати правила для визначення специфічних характеристик файлів. Важливо було забезпечити підтримку роботи з правилами YARA, щоб користувач міг створювати або редагувати їх без додаткових складнощів. Крім того, планувалося інтегрувати програму з сервісом VirusTotal, який може виконувати багатопоточний аналіз файлів за допомогою різних антивірусних двигунів. Це забезпечило б можливість перевірки підозрілих файлів у реальному часі. На основі цього аналізу була визначена архітектура системи, яка складається з кількох основних модулів. Першим модулем є модуль управління файлами, який відповідає за обробку та аналіз файлів або директорій, наданих користувачем. Цей модуль має підтримувати різні формати файлів, включаючи виконувані файли, документи, архіви та інші типи, що можуть містити шкідливий код. Другий модуль відповідає за генерацію та управління правилами YARA. Він дозволяє створювати правила, перевіряти їх на коректність і застосовувати до вибраних файлів. Третій модуль — це інтеграція із зовнішніми сервісами, зокрема VirusTotal, що забезпечує можливість розширеного аналізу підозрілих файлів.

На етапі проектування були створені схеми взаємодії між цими модулями. Основна увага приділялася забезпеченню безперервного потоку даних між модулями, щоб користувач міг виконувати аналіз файлів із мінімальними затримками. Для цього було вирішено використовувати багатопоточну обробку, яка дозволяє аналізувати кілька файлів одночасно. Це особливо важливо при роботі з великими обсягами даних.

Також було вирішено реалізувати графічний інтерфейс користувача для спрощення роботи з програмою(Рисунок 3.2). Інтерфейс повинен бути інтуїтивно зрозумілим, щоб користувач міг легко завантажувати файли, переглядати результати аналізу та керувати правилами YARA. Наприклад, вікно управління файлами дозволяє вибирати окремі файли або цілі

директорії для аналізу, а також відображає статус кожного файлу після завершення перевірки.

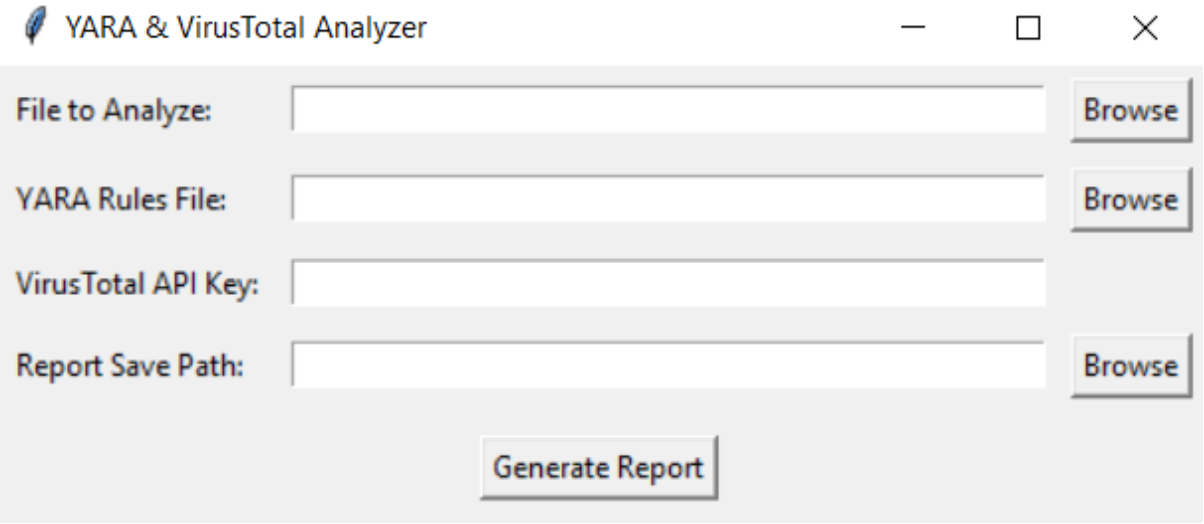

Рисунок 3.2 - Інтерфейс програми

Особлива увага приділялася проектуванню бази даних для збереження інформації про перевірені файли, застосовані правила та отримані результати. Це забезпечує можливість перегляду історії аналізу, що є важливим для подальшого дослідження та розслідування кіберінцидентів. Наприклад, база даних містить інформацію про кожен файл, його хеш-суму, результати перевірки за допомогою YARA та дані з VirusTotal. Це дозволяє швидко отримати доступ до раніше перевірених файлів і уникнути повторного аналізу.

Під час проектування також було враховано необхідність захисту системи від можливих атак. Наприклад, у програмі реалізована функція перевірки вхідних файлів на розмір і формат, щоб уникнути перевантаження системи або її виходу з ладу через обробку некоректних файлів. Крім того, модуль інтеграції з VirusTotal має обмеження на кількість запитів, що дозволяє уникнути перевищення лімітів API. Це було важливо для забезпечення стабільної роботи програми навіть у разі інтенсивного використання. Проектування програмного засобу для виявлення ШПЗ було складним, але цікавим процесом, який вимагав

врахування багатьох аспектів, починаючи від функціональності та ефективності до зручності користувачів і безпеки. Результатом цього етапу стала чітка архітектура системи, яка об'єднує модулі для управління файлами, роботи з YARA та інтеграції з VirusTotal, що забезпечує високу гнучкість і ефективність у виявленні шкідливих програм.

3.2 Розробка основних функцій

Розробка основних функцій програмного засобу для виявлення шкідливого програмного забезпечення є важливим етапом, який визначає його ефективність, продуктивність та зручність у використанні. На цьому етапі основну увагу було приділено створенню функціоналу для аналізу файлів, управління правилами YARA, інтеграції із зовнішніми сервісами та обробці результатів. Кожна з цих функцій була реалізована з урахуванням сучасних вимог до програм безпеки.

Однією з перших розроблених функцій був модуль аналізу файлів. Цей модуль відповідає за сканування наданих користувачем файлів за допомогою правил YARA. Я реалізував обробку як окремих файлів, так і цілих директорій, що значно розширило можливості системи. Наприклад, якщо користувач завантажує директорію, програма автоматично аналізує всі файли в ній, використовуючи наявні правила. У процесі розробки було вирішено забезпечити підтримку різних типів файлів, включаючи виконувані файли, текстові документи, архіви та навіть зображення. Це досягалося шляхом створення універсального механізму читання та аналізу файлів, що дозволяло застосовувати правила YARA(Рисунок 3.3) до будь-якого формату.

```python
import yara

def analyze_file(file_path, rule_path):

    try:
        rules = yara.compile(filepath=rule_path)
        matches = rules.match(file_path)
        return matches
    except Exception as e:
        return f"Помилка: {e}"
```

Рисунок 3.3 - Основна функція аналізу файлів

Для реалізації аналізу я використовував бібліотеку YARA-Python, яка надає можливості для роботи з правилами YARA безпосередньо з Python-коду. Одним із важливих аспектів цієї реалізації була підтримка динамічного завантаження правил. Це означає, що користувач може додавати нові правила або оновлювати існуючі без необхідності перезапуску програми. Наприклад, якщо з'являється нова загроза, дослідник безпеки може створити нове правило YARA(Рисунок 3.4) та одразу застосувати його до наявних файлів. Така гнучкість є важливою перевагою програми.

```
rule AdvancedMalwareDetection {
    meta:
        description = "Правило для виявлення підозрілих функцій і патернів у шкідливому ПЗ"
        author = "Student"
        date = "2024-11-05"

    strings:
        $suspicious_string1 = "malicious_function"
        $suspicious_string2 = "critical_call"
        $regex_pattern = /dangerous_[a-zA-Z0-9_]+/ nocase
        $encoded_data = { 68 65 6C 6C 6F }  // Хекс-послідовність для "hello"

    condition:
        // Розмір файлу менше 1 МБ і виконуються хоча б дві умови
        filesize < 1MB and
        (
            all of ($suspicious_string*) or
            $regex_pattern or
            $encoded_data
        )
}
```

Рисунок 3.4 - Прикладом створення правила YARA

Для обчислення хеш значень файлів створена наступна функція calculate_file_hash.

```
def calculate_file_hash(file_path, hash_algorithm="sha256"):
    try:
        hash_func = hashlib.new(hash_algorithm)
    except ValueError:
        print(f"Алгоритм '{hash_algorithm}' не підтримується.")
        return None
    try:
        with open(file_path, "rb") as f:
            while chunk := f.read(8192):
                hash_func.update(chunk)
        return hash_func.hexdigest()
    except FileNotFoundError:
        print(f"Файл '{file_path}' не знайдено.")
        return None
    except Exception as e:
        print(f"Помилка: {e}")
        return None
```

Інтеграція з сервісом VirusTotal була ще одним важливим компонентом. Цей модуль дозволяє завантажувати підозрілі файли на сервер VirusTotal для додаткової перевірки. У процесі розробки було важливо забезпечити надійну обробку результатів, отриманих від сервісу. Наприклад, якщо файл виявляється шкідливим, програма зберігає деталі в базі даних, включаючи кількість виявлень та опис знайдених загроз. Для реалізації цього модуля я використав офіційний API VirusTotal, що дозволяє надсилати файли на сервер та отримувати результати аналізу у форматі JSON (рисунок 3.5). Ці дані потім обробляються і відображаються користувачу у зрозумілому вигляді.

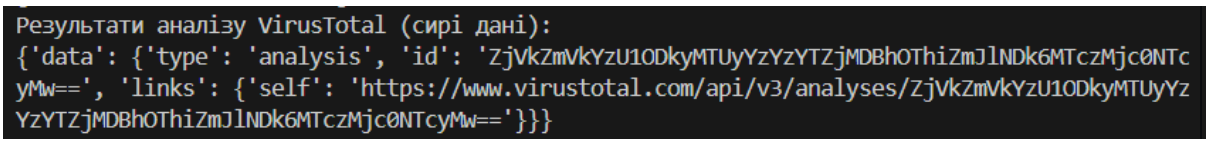

Рисунок 3.5 - Сирі дані

Однією з ключових задач під час розробки було забезпечення високої продуктивності програми. Для цього я реалізував багатопоточний механізм обробки файлів, що дозволило аналізувати кілька файлів одночасно. Наприклад, якщо користувач завантажує директорію з сотнею файлів, програма розподіляє їх між потоками, виконуючи аналіз паралельно. Це значно зменшує час сканування, особливо для великих наборів даних. У процесі реалізації багатопоточності я врахував можливі конфлікти доступу до спільних ресурсів, таких як база даних, і забезпечив коректну синхронізацію потоків.

Додатково було реалізовано функцію генерації звітів про результати аналізу. Звіти містять інформацію про кожен проаналізований файл, включаючи його хеш-суму, статус перевірки, знайдені загрози та використані правила.

Також для пошуку файлів за хешем реалізовано необхідну функцію find_file_by_hash.

```
def find_file_by_hash(directory, target_hash, hash_algorithm="sha256"):
    try:
        for root, _, files in os.walk(directory):
            for file in files:
                file_path = os.path.join(root, file)
                if calculate_file_hash(file_path, hash_algorithm) == target_hash:
                    return file_path
    except Exception as e:
        print(f"Помилка під час пошуку: {e}")
    return None
```

Ці звіти можуть експортуватися у форматі PDF або CSV, що зручно для подальшого аналізу або передачі іншим фахівцям. Наприклад, якщо виявлено кілька підозрілих файлів у директорії, користувач може згенерувати звіт і надіслати його команді кібербезпеки для додаткового дослідження

Розробка основних функцій також включала створення механізму журналювання подій та створення бази даних хеш значень для спрощення пошуку. Коб відповідної функції наведемо нижче.

```
def create_hash_database(directory, output_file, hash_algorithm="sha256"):
    hash_database = {}
    try:
        for root, _, files in os.walk(directory):
            for file in files:
                file_path = os.path.join(root, file)
                file_hash = calculate_file_hash(file_path, hash_algorithm)
                if file_hash:
                    hash_database[file_hash] = file_path
        with open(output_file, "w", encoding="utf-8") as f:
            json.dump(hash_database, f, indent=4, ensure_ascii=False)
        print(f"База даних хешів збережена в '{output_file}'")
    except Exception as e:
        print(f"Помилка під час створення бази даних: {e}")
```

Усі дії програми, включаючи завантаження файлів, застосування правил, результати перевірок та інтеграцію з VirusTotal, зберігаються у вигляді журналу.

```
def check_hash_on_virustotal(file_hash, api_key):
    url = f"https://www.virustotal.com/api/v3/files/{file_hash}"
    headers = {
```

```
        "x-apikey": api_key
    }
    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            return response.json()
        elif response.status_code == 404:
            print("Хеш не знайдено на VirusTotal.")
            return None
        else:
            print(f"Помилка: {response.status_code} - {response.text}")
            return None
    except Exception as e:
        print(f"Помилка під час запиту до VirusTotal: {e}")
        return None
```

Це забезпечує прозорість роботи програми та дозволяє відстежувати її поведінку у разі виникнення помилок. Наприклад, якщо файл не вдалося проаналізувати через пошкодження або інший технічний збій, ця інформація відображається у журналі разом із описом проблеми. Такий підхід допомагає ідентифікувати та вирішувати проблеми під час експлуатації програми. Усі ці функції були розроблені з урахуванням потреб користувачів, які працюють у сфері кібербезпеки. Завдяки гнучкості, продуктивності та зручності програма дозволяє ефективно виявляти та досліджувати шкідливе програмне забезпечення, відповідаючи сучасним викликам у галузі інформаційної безпеки. Основні функції що були використані в проекті приведені в додатку Б.

3.3 Тестування та відлагодження

Тестування та відлагодження є завершальним етапом розробки програмного засобу, який забезпечує його стабільну та коректну роботу. Після завершення розробки основних функцій програма потребує ретельної перевірки, щоб гарантувати відповідність функціональним вимогам, виявити можливі помилки та оптимізувати продуктивність. На цьому етапі я провів комплексне тестування програмного забезпечення, включаючи перевірку коректності аналізу файлів, роботи з правилами YARA, інтеграції з VirusTotal та генерації звітів. Першим кроком було функціональне тестування кожного з модулів окремо. Для модуля аналізу файлів я створив тестову вибірку, яка включала різні типи файлів: виконувані програми, текстові документи, архіви та зображення. У цій вибірці також були файли, що містили відомі шкідливі сигнатури, і звичайні безпечні файли. Програма коректно ідентифікувала всі шкідливі файли відповідно до правил YARA (рисунок 3.6), що підтвердило правильність роботи модуля аналізу. Однак у процесі тестування були виявлені незначні проблеми з обробкою великих файлів, що призводило до перевищення часу аналізу. Для вирішення цієї проблеми я реалізував оптимізацію коду, що значно підвищило продуктивність модуля.

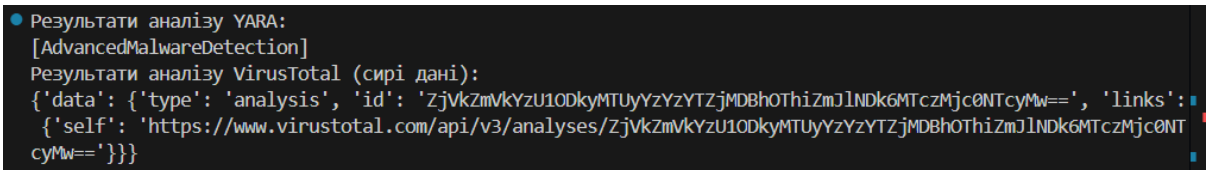

Рисунок 3.6 - Дані Yara і VirusTotal

Наступним етапом стало інтеграційне тестування, яке перевіряло взаємодію між модулями програми. Зокрема, я протестував, як модуль аналізу файлів передає дані модулю генерації звітів і як обробляються результати від сервісу VirusTotal. Наприклад, під час тестування було виявлено, що деякі відповіді від VirusTotal містили нестандартні дані, які

призводили до помилок при їх обробці. Щоб усунути цю проблему, я додав механізм перевірки коректності отриманих даних, який дозволив уникнути збоїв. Результати інтеграційного тестування показали, що всі модулі коректно взаємодіють між собою, забезпечуючи цілісність роботи системи (рисунок 3.7).

Рисунок 3.7 - Програмою було перевірено декілька файлів

Під час тестування багатопотокової обробки файлів я звернув увагу на можливі конфлікти доступу до бази даних, які могли виникати при одночасній роботі кількох потоків. Для вирішення цієї проблеми я реалізував механізм блокування ресурсів, що гарантував послідовний доступ до бази даних. Наприклад, якщо кілька потоків намагалися одночасно записати результати аналізу, програма автоматично розподіляла доступ, забезпечуючи коректність запису. Після цього багатопоточна обробка працювала стабільно навіть із великими обсягами даних.

Особливу увагу я приділив тестуванню функції генерації звітів. Звіти містять інформацію про всі проаналізовані файли, знайдені загрози, використані правила та результати перевірки з VirusTotal. Я перевірив, як програма формує звіти у форматах PDF і CSV (рисунок 3.8), і переконався, що всі дані відображаються коректно. Наприклад, під час тестування було виявлено, що звіти у форматі PDF некоректно обробляли символи Unicode, що могло призводити до помилок у відображенні тексту. Після внесення змін до механізму генерації звітів ця проблема була усунена.

YARA Analysis Report
File: ../test_samples/malicious_sample.txt

YARA Results:
- AdvancedMalwareDetection
VirusTotal Results: https://www.virustotal.com/api/v3/analyses/ZjVkZmVkYzU1ODkyMTUyYzYzYTZjMDBhOThiZmJlNDk6MTczMjc0NTcyMw==
Detected Threats: 2
Threat Details:

Рисунок 3.8 - Звіт Pdf

Після завершення всіх етапів тестування програмний засіб був готовий до використання. В результаті тестування вдалося виявити та виправити всі критичні помилки, оптимізувати продуктивність і забезпечити стабільну роботу всіх модулів. Програма повністю відповідає функціональним вимогам і демонструє високу ефективність у виявленні шкідливого програмного забезпечення. У майбутньому її можна розширити, додавши нові функції, такі як автоматичне оновлення правил YARA або інтеграція з іншими сервісами для аналізу загроз.

ВИСНОВКИ

Сучасний розвиток цифрових технологій супроводжується зростанням кіберзагроз, особливо шкідливого програмного забезпечення (malware), яке завдає значної шкоди як приватним особам, так і організаціям. Традиційні методи захисту, такі як сигнатурний аналіз, вже не можуть забезпечити ефективний рівень безпеки, що обумовлює необхідність впровадження інноваційних підходів, зокрема машинного навчання, поведінкового аналізу та інтеграції сучасних інструментів, таких як YARA та VirusTotal. Метою роботи є розробка ефективних алгоритмів для виявлення та захисту від шкідливого ПЗ, що дозволяють забезпечити високий рівень безпеки інформаційних систем.

У ході дослідження було розглянуто сучасні методи протидії кіберзагрозам, включаючи антивірусні системи, міжмережеві екрани, системи IDS/IPS та застосування штучного інтелекту. На основі проведеного аналізу розроблено програмний засіб, який поєднує правила YARA для аналізу файлів та інтеграцію з платформою VirusTotal. Програма підтримує багатопотокову обробку, що дозволяє аналізувати великі обсяги даних, створювати звіти у зручних форматах (PDF, CSV) та забезпечувати ефективну взаємодію модулів.

Проведено комплексне тестування системи, виявлено та усунуто проблеми, пов’язані з обробкою великих файлів та нестандартних даних. Оптимізовано продуктивність за рахунок використання багатопотоковості та механізмів блокування ресурсів. Розроблений засіб підтвердив свою ефективність у виявленні кіберзагроз, демонструючи стабільну роботу. У перспективі можливе впровадження нових функцій, таких як автоматичне оновлення правил YARA, розширення підтримуваних форматів файлів та інтеграція з іншими платформами кібербезпеки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чубей О, Романів О., Івасьєв С. Алгоритми виявлення та захисту від зловмисного програмного забезпечення. Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп’ютерно-інтегровані технології» (АКІТ - 2024), Тернопіль, 2024. -С. 73-76.

2. Романів О., Кобиця В., Лизун Я., Автоматизована перевірка програмних засобів для виявлення шкідливого впливу на ОС. Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024.С. 146-150.

3. Астахов В.Ю., Гладун О.М., Мельничук О.О. "Методи захисту інформаційних систем від кіберзагроз." Збірник матеріалів науково-практичної конференції "Інформаційна безпека та кіберзахист", Київ, 2023, с. 45-52.

4. Іваненко І.О., Петров С.В. "Впровадження алгоритмів машинного навчання для виявлення шкідливого програмного забезпечення." Науковий вісник Черкаського національного університету. Серія "Інформатика", 2022, т. 5, №3, с. 33-38.

5. Сорока Т.І., Глушенко Л.М. "Використання YARA для виявлення шкідливих файлів у корпоративних мережах." Вісник ТНТУ. Комп’ютерні технології та системи управління, 2021, т. 15, №4, с. 78-84.

6. Бабенко М.С., Шевченко П.В. "Міжмережеві екрани нового покоління: ефективність та можливості." Збірник наукових праць НТУУ "КПІ", 2023, т. 20, №2, с. 90-96.

7. Alici, E., Keleş, A., & Ertaş, B. "Malware Detection with Machine Learning and Deep Learning Models: A Comparative Study." Journal of Computer Science and Technology, 2021, pp. 45-60.

8. Guo, H., Xu, Z., & Li, C. "Enhancing Threat Detection Using YARA Rules: A Practical Perspective." International Conference on Cybersecurity Advances, 2020, pp. 112-117.

9. Alazab, M., Vemuri, S., & Reddy, T. "Dynamic Analysis of Polymorphic Malware Using Sandboxing Techniques." IEEE Transactions on Information Forensics and Security, 2019, vol. 14, pp. 1821-1832.

10. Іванова Н.О. "Захист інформації у віртуальних середовищах." Матеріали міжнародної конференції "Кібербезпека у сучасному світі", Львів, 2022, с. 112-118.

11. Kim, D., Kim, K., & Park, H. "The Integration of VirusTotal API in Automated Malware Detection Systems." Proceedings of the IEEE Symposium on Security and Privacy, 2020, pp. 85-90.

12. Ahmed, M., Mahmood, A., & Hu, J. "A Survey of Network Anomaly Detection Techniques." Journal of Network and Computer Applications, 2020, vol. 100, pp. 35-54.

13. Гуренко А.П., Романюк Ю.М. "Системи IDS/IPS як засіб захисту інформаційних мереж." Вісник Київського національного університету імені Тараса Шевченка. Серія: Комп’ютерні науки, 2022, №8, с. 25-30.

14. Neves, J., Teixeira, A., & Morais, M. "A Study on the Effectiveness of Modern Anti-Malware Tools." Cybersecurity and Privacy Journal, 2019, vol. 8, pp. 68-74.

15. Bishop, M., & Bailey, D. "Computer Security: Art and Science." Pearson Education, 2019, pp. 234-256.

16. Diffie, W., & Hellman, M. "New Directions in Cryptography." IEEE Transactions on Information Theory, 1976, vol. 22, pp. 644-654.

17. Hutchins, E. M., Cloppert, M. J., & Amin, R. M. "The Cyber Kill Chain: Linking Malware and Attacks." Lockheed Martin Cyber Security Research, 2019, pp. 89-96.

18. Глушко П.В. "Використання хмарних технологій у кібербезпеці." Наукові праці ОНПУ. Серія "Комп’ютерні науки", 2023, т. 19, №3, с. 102-108.

19. Trend Micro. "Deep Security Techniques for Modern Threats." Trend Micro Research Center, 2021, pp. 78-89.

20. Garfinkel, T., & Rosenblum, M. "A Virtual Machine Introspection Based Architecture for Intrusion Detection." Proceedings of the Network and Distributed Systems Security Symposium, 2019, pp. 67-72.

21. Clarke, E., & Peled, D. "Advanced Model Checking Techniques in Cybersecurity." The MIT Press, 2019, pp. 45-50.

22. Johnson, L. "Comprehensive Analysis of IDS/IPS Systems in Corporate Environments." IEEE Transactions on Network and Service Management, 2021, vol. 17, pp. 120-132.

23. Hashimoto, T., & Yamada, S. "Cyber Threats and Security Measures in the Era of Machine Learning." Journal of Artificial Intelligence Research, 2020, vol. 67, pp. 78-89.

24. Patel, H., & Jain, A. "Behavioral Analysis of Malware Using Process Monitoring and Memory Analysis." Proceedings of the International Conference on Software Security and Reliability, 2021, pp. 90-96.

25. Virvilis, N., & Gritzalis, D. "The Big Four—What We Did Wrong in Advanced Persistent Threat Detection?" Computers & Security, 2021, vol. 77, pp. 45-55.

26. Duan, Q., Wang, Y., & Zhao, R. "Static vs Dynamic Analysis in Malware Research: Challenges and Opportunities." ACM Computing Surveys, 2021, vol. 53, no. 6, pp. 22-28.

27. R. H. Mahdi and H. Trabelsi, "Detection of Malware by Using YARA Rules," 2024 21st International Multi-Conference on Systems, Signals & Devices (SSD), Erbil, Iraq, 2024, pp. 1-8, doi: 10.1109/SSD61670.2024.10549308.

28. N. Naik, P. Jenkins, R. Cooke, J. Gillett and Y. Jin, "Evaluating Automatically Generated YARA Rules and Enhancing Their Effectiveness," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 2020, pp. 1146-1153, doi: 10.1109/SSCI47803.2020.9308179.

29. N. Naik *et al.*, "Fuzzy Hashing Aided Enhanced YARA Rules for Malware Triaging," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 2020, pp. 1138-1145, doi: 10.1109/SSCI47803.2020.9308189.

30. C. N. Vanitha, S. Malathy, M. M. Musthafa, T. C. Kalaiselvi, S. A. Krishna and K. Harishankar, "An Effective Method to Detect Malware Files with Yara Using RaspberryPi," 2023 5th International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2023, pp. 1358-1362, doi: 10.1109/ICIRCA57980.2023.10220643.

31. H. T. Zaw, T. Aung, A. H. Maw and M. Thida Mon, "Comparative Analysis of YARA and VirusTotal Integrations with Wazuh for Enhanced Malware Detection," 2024 5th International Conference on Advanced Information Technologies (ICAIT), Yangon, Myanmar, 2024, pp. 1-6, doi: 10.1109/ICAIT65209.2024.10754931.

32. M. Khalid, M. Ismail, M. Hussain and M. Hanif Durad, "Automatic YARA Rule Generation," 2020 International Conference on Cyber Warfare and Security (ICCWS), Islamabad, Pakistan, 2020, pp. 1-5, doi: 10.1109/ICCWS48432.2020.9292390.

ДОДАТОК А

КОПІЇ ПУБЛІКАЦІЙ

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАДВІРНЯНСЬКИЙ КОЛЕДЖ НАЦІОНАЛЬНОГО ТРАНСПОРТНОГО УНІВЕРСИТЕТУ
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Проблемно-наукова міжгалузева конференція

АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

(АКІТ – 2024)

22-24 лютого 2024 року
м. Тернопіль

Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно-інтегровані технології» (АКІТ - 2024), Тернопіль, 2024. -86 с.

Адреса організаторів:

вул. Олени Теліги 8, м. Тернопіль 46003,
кафедра спеціалізованих комп'ютерних систем,
Західноукраїнський національний університет.

Контакти: тел.: (0352) 50-17-87, e-mail: scs.kafedra@gmail.com.

УДК 681.32

***Олександра ЧУБЕЙ*[1], *Олег РОМАНІВ*[1], *Степан ІВАСЬЄВ*[2]**
[1]*Галицький фаховий коледж імені В'ячеслава Чорновола*
[2]*Західноукраїнський національний університет*

АЛГОРИТМИ ВИЯВЛЕННЯ ТА ЗАХИСТУ ВІД ЗЛОВМИСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Актуальність теми виявлення та захисту від зловмисного програмного забезпечення, безумовно, важлива в сучасному цифровому світі. Зловмисне програмне забезпечення, таке як віруси, трояни, шпигунське програмне забезпечення та інші види атак [1], може завдати серйозної шкоди комп'ютерним системам, персональним даним та комп'ютерним мережам. Постійний розвиток цих загроз вимагає постійного вдосконалення методів виявлення та запобігання їм. Тому вивчення алгоритмів захисту та їх застосування у практиці є надзвичайно важливим для забезпечення кібербезпеки.

Метою роботи є дослідження підходів та засобів дослідження шкідливого програмного забезпечення.

1. Статистичний аналіз шкідливого програмного забезпечення

Статистичний аналіз шкідливого програмного забезпечення (ШПЗ) є важливим етапом у розумінні характеристик та властивостей цього виду загрози. Для статистичного аналізу ШПЗ можна виділити наступну послідовність кроків.

Збір даних про загрози: включає в себе збір і аналіз даних про виявлені випадки ШПЗ. Це може включати зразки ШПЗ, логи подій, дані з обліку вразливостей та інші джерела інформації. Опис зразків ШПЗ: Отримані зразки ШПЗ аналізуються для визначення їх характеристик, таких як тип, функціональність, методи розповсюдження тощо. Вивчення вразливостей і експлойтів. Досліджуються вразливості в системах, які використовуються ШПЗ, та методи їх експлуатації. Це допоможе зрозуміти, як ШПЗ потрапляє в систему та які уразливості вона використовує.

Аналіз розподілу і поширення ШПЗ. Визначення розподілу та швидкості поширення ШПЗ в мережі або серед користувачів. Це може включати аналіз часових рядів, графіків поширення, статистику інцидентів тощо. Оцінка впливу ШПЗ: Проаналізуйте наслідки атак ШПЗ, такі як фінансові збитки, втрати даних, порушення конфіденційності тощо. Це допоможе визначити масштаби впливу і розробити стратегії мінімізації ризиків. Пошук патернів та трендів: Використовуйте статистичні методи для виявлення патернів та трендів в розвитку ШПЗ. Це може допомогти передбачити майбутні напрямки розвитку атак та вжити заходів забезпечення. Розробка протидійних заходів включає результати аналізу, розробку та впровадження стратегії протидії ШПЗ, такі як заходи безпеки, патчі вразливостей, виявлення та вилучення ШПЗ тощо.

Статичний аналіз шкідливого програмного забезпечення (ШПЗ) використовує різноманітні інструменти для аналізу вихідного коду, виконуваних файлів та інших складових, щоб виявити потенційно шкідливі дії та вразливості. Серед поширених інструментів для статичного аналізу ШПЗ:

- IDA Pro один з найбільш потужних і популярних інструментів для

аналізу бінарного коду. Він дозволяє динамічне та статичне аналізувати програми, виявляти вразливості та видаляти ШПЗ.

- Binary Ninja це інструмент що надає широкі можливості для аналізу бінарного коду, включаючи виявлення вразливостей, декомпіляцію, реверс-інжиніринг та інше.
- Розроблений Національним центром кібербезпеки (NSA), Ghidra є безкоштовним інструментом для реверс-інжинірингу і аналізу ШПЗ. Він надає широкий спектр функцій для аналізу бінарних файлів.
- Radare2 вільний та відкритий інструмент надає можливості для аналізу бінарних файлів, реверс-інжинірингу та виявлення вразливостей.
- YARA інструмент для створення правил пошуку підписів ШПЗ у файловій системі. Він дозволяє автоматизувати процес виявлення ШПЗ за їхніми характеристиками.
- PEiD / PEview спеціалізуються на аналізі Portable Executable (PE) файлів, які часто використовуються у Windows-програмах. Вони допомагають виявити підозрілі або вбудовані вирази ШПЗ.

Ці інструменти допомагають дослідникам та аналітикам виявити та аналізувати ШПЗ, щоб розробити ефективні стратегії протидії цій загрозі[2].

2. Динамічний аналіз шкідливого програмного забезпечення

Динамічний аналіз шкідливого програмного забезпечення (ШПЗ) включає в себе виконання програми-зразка в контрольованому середовищі для спостереження його поведінки та виявлення потенційно шкідливих дій. Приведемо послідовність кроків, які можна виконати для динамічного аналізу ШПЗ. Підготовка середовища: Спочатку необхідно створити безпечне середовище для виконання програми-зразка. Це може бути віртуальна машина, контейнер або ізольована система, що запобігає поширенню ШПЗ за межі тестового середовища. Запуск програми-зразка. Запуск у контрольованому середовищі. Це може бути виконано вручну або автоматизованою системою.

Спостереження за діями програми включає моніторинг мережевої активності, зміни файлової системи, спроби доступу до системних ресурсів тощо. Збір і аналіз поведінкових даних може бути зроблено за допомогою різноманітних інструментів для аналізу системних журналів, мережевого трафіку, змін файлової системи тощо.

Виявлення потенційно шкідливих дій дозволить виявити будь-які незвичайні або потенційно шкідливі дії програми-зразка. Це може включати виявлення спроб експлуатації вразливостей, співробітництво з віддаленими серверами, модифікацію системних файлів тощо. Аналіз результатів і розробка заходів протидії: На основі результатів аналізу розробляється стратегії протидії ШПЗ, такі як створення сигнатур або правил для виявлення, патчі вразливостей, блокування шкідливої активності та інші заходи безпеки.

Ці кроки дозволяють ефективно виявляти та аналізувати ШПЗ, а також розробляти стратегії для захисту від цих загроз.

Для динамічного аналізу шкідливого програмного забезпечення (ШПЗ) існують різноманітні інструменти, які дозволяють аналізувати поведінку програм у контрольованому середовищі. Серед поширених інструментів для динамічного

аналізу ШПЗ є: Cuckoo Sandbox, FireEye Malware Analysis, Joe Sandbox, VMRay Analyzer, Sandboxie, CAPE Sandbox. Cuckoo Sandbox відкрите програмне забезпечення, яке дозволяє автоматизувати процес аналізу ШПЗ у віртуальному середовищі. Воно надає можливості для спостереження за діями програми-зразка, виявлення потенційно шкідливих дій та створення звітів про аналіз. FireEye Malware Analysis: інструмент що використовується для динамічного аналізу ШПЗ та виявлення нових загроз. Він надає різноманітні можливості для відстеження активності програми-зразка, виявлення вразливостей та здатність до автоматичного аналізу. Joe Sandbox: інструмент що дозволяє автоматизовано виконувати динамічний аналіз ШПЗ та глибоке вивчення поведінки програм. Він надає засоби для виявлення навмисних дій програм, виявлення вразливостей та інших загроз безпеці. VMRay Analyzer: інструмент що використовує віртуальне середовище для динамічного аналізу ШПЗ та виявлення потенційних загроз. Він надає розширені можливості для моніторингу системних викликів, мережевої активності та іншої поведінки програм. Sandboxie: інструмент що дозволяє ізолювати програми в окремому середовищі для вивчення їх поведінки та спостереження за можливими шкідливими діями. CAPE Sandbox: відкритий інструмент надає можливості для автоматизованого динамічного аналізу ШПЗ та виявлення загроз безпеки. Він дозволяє створювати звіти про аналіз та інтегрувати з іншими системами безпеки.

Ці інструменти допомагають аналітикам та дослідникам ефективно аналізувати поведінку ШПЗ у контрольованому середовищі та виявляти потенційні загрози для безпеки [3].

Висновки

Динамічний і статистичний аналіз - це два різних підходи до аналізу даних, кожен з яких має свої переваги та обмеження. Статистичний аналіз використовує статистичні методи для вивчення взаємозв'язків та закономірностей у наборі даних. Це може включати аналіз розподілу, кореляційний аналіз, факторний аналіз тощо.

Обидва методи мають свої сильні та слабкі сторони, і їхню ефективність можна максимально використовувати шляхом комбінування їх застосування. Наприклад, динамічний аналіз може виявити нові атаки або вразливості, а статистичний аналіз може допомогти вивчити попередні інциденти та патерни. Комплексний підхід до аналізу дозволяє отримати більш повне розуміння загроз та захистити систему від шкідливого програмного забезпечення.

Перелік використаних джерел

1. Sikorski M., H. A. (2012). Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software. Szor, P. (2005). The Art of Computer Virus Research and Defense.

2. VirusTotal (2020). Daily Statistics. [Electronic resource] – Access: https://www.virustotal.com/en/statistics/

3. Nasi, E. (2014). Bypass Antivirus Dynamic Analysis. [Electronic resource] – Access: https://wikileaks.org/ciav7p1/cms/files/BypassAVDynamics.pdf



ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2024)

науково-практична конференція
молодих вчених, аспірантів та студентів

26–28 серпня 2024
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. - 160 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

УДК 004

Олег РОМАНІВ, Віталій КОБИЦЯ, Ярослав ЛИЗУН

Західноукраїнський національний університет

АВТОМАТИЗОВАНА ПЕРЕВІРКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВИЯВЛЕННЯ ШКІДЛИВОГО ВПЛИВУ НА ОС

Вступ. У сучасному світі програмного забезпечення (ПЗ) якість та надійність є ключовими факторами успішної розробки. Зважаючи на зростаючі вимоги до складності систем, швидкості їх розробки та випуску на ринок, перевірка програмного забезпечення стає невід'ємною складовою процесу розробки. Однак, через велику кількість можливих шляхів виконання програм, ручне тестування є неефективним і витратним.

Автоматизована перевірка ПЗ дозволяє зменшити ці витрати та підвищити якість кінцевого продукту. Одним із інструментів автоматизованої перевірки є пісочниці (sandbox), які створюють ізольоване середовище для безпечного виконання програм.

Пісочниці дозволяють аналізувати поведінку ПЗ без ризику пошкодження основної системи або мережі. Вони відіграють важливу роль у процесі тестування, особливо в умовах необхідності перевірки на шкідливість або перевірки функціональності ПЗ[1] в різних середовищах.

Мета: аналіз існуючих засобів автоматизованої перевірки програмного забезпечення, зокрема за допомогою пісочниць, а також визначення переваг і недоліків їх використання в умовах сучасних загроз для аналізу шкідливого ПЗ.

1. Існуючі засоби автоматизованого аналізу програмного забезпечення

На сьогодні існує велика кількість інструментів, що забезпечують автоматизовану перевірку програмного забезпечення. Серед них виділяються інструменти, орієнтовані на різні аспекти тестування: функціональність, безпека, продуктивність та інші. Найпопулярнішими засобами є:

- Selenium - відкритий інструмент для автоматизації веб-тестування.
- JUnit - фреймворк для автоматизації тестування Java-програм.
- Appium - інструмент для автоматизованого тестування мобільних додатків.

2. Пісочниці як засіб перевірки ПЗ

Пісочниці є ізольованими середовищами, що використовуються для виконання програм або тестів з метою уникнення потенційної шкоди основній системі. Вони дозволяють тестувати програмне забезпечення безпосередньо в середовищі, яке імітує реальні умови експлуатації[2]. Це особливо корисно для виявлення шкідливого ПЗ, тестування безпеки та виконання функціональних тестів у контрольованому середовищі. На рисунку 1 приведена архітектура типової автоматизованої пісочниці для виявлення шкідливого ПЗ.

Одними з основних інструментів для створення пісочниць є:

- VirtualBox - засіб для створення віртуальних машин, які можна використовувати як пісочниці для тестування.

- Docker - контейнеризаційна платформа, яка дозволяє запускати ПЗ в ізольованих контейнерах.
- Sandboxie - специфічний інструмент для ізоляції виконуваних файлів і додатків в середовищі Windows.

Більшість пісочниць мають типову архітектуру та набір компонентів.

Складові компоненти типової пісочниці приведемо на рисунку 1.

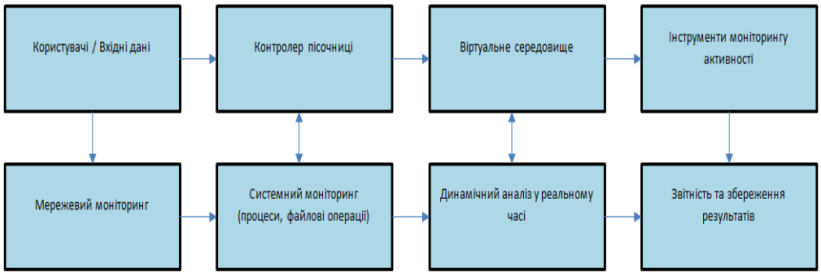

Рисунок 1 - Архітектура пісочниці для тестування ПЗ

Проаналізувавши функціональні можливості пісочниць виділимо переваги їхнього застосування:

1. Безпека: пісочниці дозволяють запускати підозріле програмне забезпечення без ризику для основної системи.
2. Гнучкість: можливість створювати різні середовища для тестування (різні ОС, конфігурації системи тощо).
3. Ефективність: автоматизація процесу тестування дозволяє скоротити час на виявлення проблем.

Серед недоліків є обмеження продуктивності: використання віртуалізації чи контейнеризації може впливати на продуктивність ПЗ, необхідність налаштування: для ефективного використання пісочниць потрібні певні знання і ресурси, неможливість повної імітації реального середовища, хоча пісочниці можуть імітувати багато аспектів системи, вони не завжди можуть точно відтворити всі умови реальної експлуатації.

3. Приклади використання пісочниць в тестуванні ПЗ

Одним з найяскравіших прикладів використання пісочниць є перевірка шкідливого програмного забезпечення в антивірусних компаніях. Компанії, такі як Microsoft та Symantec, активно використовують пісочниці для аналізу підозрілих файлів, визначаючи їхню шкідливість та поведінку в ізольованих середовищах. Часто дослідники використовують пісочниці з відкритим кодом, такі як Cuckoo Sandbox.

Архітектура Cuckoo Sandbox базується на модульному підході, який забезпечує гнучкість, масштабованість і ефективність аналізу підозрілих файлів. Основні компоненти системи розподілені між хостовою системою та віртуальними машинами (VM), що дозволяє ізолювати шкідливе програмне забезпечення для безпечного дослідження.

На рисунку 2 приведена схема послідовності дій для виконання динамічного аналізу шкідливого програмного забезпечення. Хостова система відповідає за управління всім процесом аналізу.

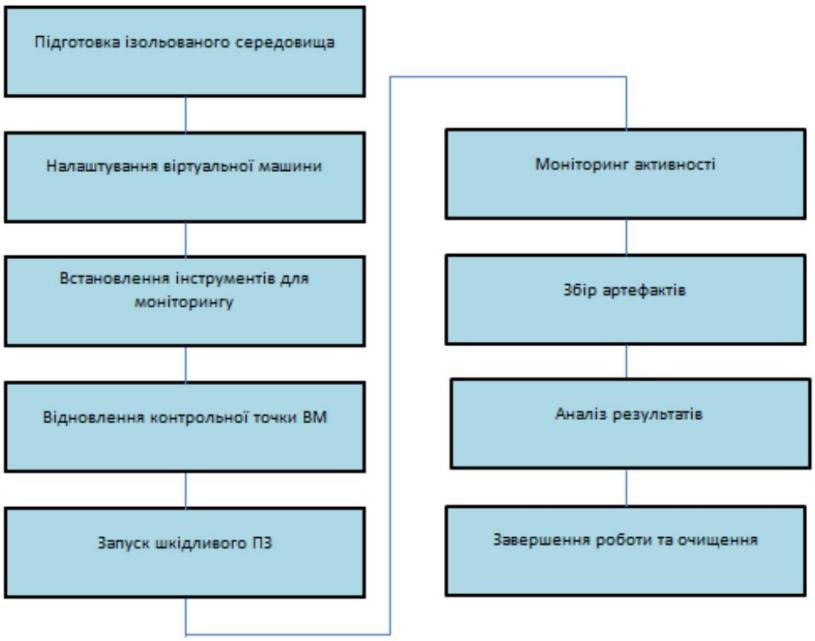

Рисунок 2 - Схема проведення динамічного аналізу шкідливого програмного забезпечення

Гостьові системи (VM) є середовищем, у якому виконується шкідливий файл.

Cuckoo Sandbox - це потужний інструмент для динамічного аналізу підозрілих файлів у безпечному віртуальному середовищі. Налаштування може здатися складним, оскільки вимагає налаштування, як хостової системи, так і віртуальних машин. Проведемо налаштування та встановлення залежностей середовища:

sudo apt update
sudo apt install python3 python3-pip python3-venv git libffi-dev libssl-dev
sudo apt install libjpeg-dev zlib1g-dev swig
sudo apt install mongodb postgresql postgresql-contrib

Наступним етапом буде встановлення hypervisor для віртуалізації. Для virtualbox потрібно виконати команду:

sudo apt install virtualbox virtualbox-ext-pack

Для використання KVM/QEMU потрібно виконати наступну команду:

sudo apt install qemu-kvm libvirt-clients libvirt-daemon-system virt-manager

Важливим етапом є створення Python-оточення. Спочатку потрібно клонувати репозиторій Cuckoo:

git clone https://github.com/cuckoosandbox/cuckoo.git
cd cuckoo

Далі створимо, активуємо та встановимо залежності для віртуального середовища Python:

python3 -m venv venv
source venv/bin/activate
pip install -U pip setuptools wheel
pip install -r requirements.txt

Для нормальної роботи середовища та збереження інформації про події потрібно налаштувати базу даних:

sudo -u postgres psql
CREATE USER cuckoo WITH PASSWORD 'cuckoo';
CREATE DATABASE cuckoo OWNER cuckoo; \q

Cuckoo Sandbox - це проєкт із відкритим вихідним кодом, доступний для будь-кого безкоштовно. Користувачі можуть змінювати код та додавати власні модулі для розширення функціональності. для аналізу Android-зразків. Пісочниця підтримує дослідження виконання різних типів файлів**:** виконувані файли (EXE, DLL), документи (PDF, Word, Excel), скрипти (Python, JavaScript), мобільні програми (APK для Android), **п**ідтримка URL-адрес та мережевого аналізу: можна аналізувати активність вебсайтів і мережевих запитів.

Інструмент дозволяє автоматично виконувати файли та записувати їх дії без втручання людини. Можна використовувати кілька віртуальних машин одночасно для паралельного аналізу великої кількості зразків. Виявлення API-викликів, системних змін, створення/зміни файлів, роботи з реєстром здійснюється окремими інструментами та автоматизовується. Фіксація мережевих запитів (DNS, HTTP, FTP) відбувається за допомогою використання Wireshark. Використання правил YARA для пошуку сигнатур шкідливого ПЗ дозволить ефективніше здійснювати пошук зразків. Пісочниця підтримує ряд операційних систем: Windows, Linux, macOS і Android.

Ще один приклад - тестування нових версій програм у середовищах Docker. Це дозволяє розробникам перевірити програму на різних платформах, не встановлюючи її безпосередньо на основну систему.

Висновок. Засоби автоматизованої перевірки програмного забезпечення, зокрема пісочниці, відіграють важливу роль у сучасній практиці тестування. Пісочниці дозволяють виконувати ПЗ в ізольованих середовищах, що значно знижує ризик пошкодження основної системи та сприяє безпеці процесу перевірки. Завдяки використанню пісочниць можна проводити функціональне тестування, аналіз шкідливих програм, перевірку продуктивності та тестування в різних конфігураціях системи.

Перелік використаних джерел.

1 Michael Sikorski and Andrew Honig. 2012. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software (1st. ed.). No Starch Press, USA.

2 Safa Altaha and Khaled Riad, "Machine Learning in Malware Analysis: Current Trends and Future Directions" International Journal of Advanced Computer Science and Applications(IJACSA), 15(1), 2024. http://dx.doi.org/10.14569/IJACSA.2024.01501124

ДОДАТОК Б

Код програмного засобу

```
import hashlib

def calculate_file_hash(file_path, hash_algorithm="sha256"):
    """
    Обчислює хеш файлу за допомогою вказаного алгоритму хешування.

    :param file_path: Шлях до файлу
    :param hash_algorithm: Алгоритм хешування (sha256, sha1, md5 тощо)
    :return: Хеш файлу або None, якщо алгоритм не підтримується
    """
    try:
        # Отримання відповідного об'єкта хешування
        hash_func = hashlib.new(hash_algorithm)
    except ValueError:
        print(f"Алгоритм '{hash_algorithm}' не підтримується.")
        return None

    try:
        with open(file_path, "rb") as f:
            # Читання файлу блоками для обробки великих файлів
            while chunk := f.read(8192):
                hash_func.update(chunk)
        return hash_func.hexdigest()
    except FileNotFoundError:
        print(f"Файл '{file_path}' не знайдено.")
        return None
    except Exception as e:
        print(f"Помилка: {e}")
        return None

if __name__ == "__main__":
    # Приклад використання
    file_path = input("Введіть шлях до файлу: ")

    algorithms = ["md5", "sha1", "sha256", "sha512"]
    for algorithm in algorithms:
        hash_value = calculate_file_hash(file_path, algorithm)
        if hash_value:
            print(f"{algorithm.upper()} хеш: {hash_value}")
```

```
import hashlib
import os

def calculate_file_hash(file_path, hash_algorithm="sha256"):
    """
    Обчислює хеш файлу за допомогою вказаного алгоритму хешування.

    :param file_path: Шлях до файлу
    :param hash_algorithm: Алгоритм хешування (sha256, sha1, md5 тощо)
    :return: Хеш файлу або None, якщо алгоритм не підтримується
    """
    try:
        # Отримання відповідного об'єкта хешування
        hash_func = hashlib.new(hash_algorithm)
    except ValueError:
        print(f"Алгоритм '{hash_algorithm}' не підтримується.")
        return None

    try:
        with open(file_path, "rb") as f:
            # Читання файлу блоками для обробки великих файлів
            while chunk := f.read(8192):
                hash_func.update(chunk)
        return hash_func.hexdigest()
    except FileNotFoundError:
        print(f"Файл '{file_path}' не знайдено.")
        return None
    except Exception as e:
        print(f"Помилка: {e}")
        return None

def find_file_by_hash(directory, target_hash, hash_algorithm="sha256"):
    """
    Пошук файлу за його хешем у заданій директорії.

    :param directory: Шлях до директорії для пошуку
    :param target_hash: Цільовий хеш для пошуку
    :param hash_algorithm: Алгоритм хешування, використаний для обчислення хеша
    :return: Шлях до файлу або None, якщо файл не знайдено
    """
    try:
        for root, _, files in os.walk(directory):
```

```
            for file in files:
                file_path = os.path.join(root, file)
                if calculate_file_hash(file_path, hash_algorithm) == target_hash:
                    return file_path
    except Exception as e:
        print(f"Помилка під час пошуку: {e}")
    return None

if __name__ == "__main__":
    # Приклад використання
    file_path = input("Введіть шлях до файлу: ")

    algorithms = ["md5", "sha1", "sha256", "sha512"]
    for algorithm in algorithms:
        hash_value = calculate_file_hash(file_path, algorithm)
        if hash_value:
            print(f"{algorithm.upper()} хеш: {hash_value}")

    # Пошук файлу за хешем
    directory = input("Введіть директорію для пошуку: ")
    target_hash = input("Введіть цільовий хеш: ")
    algorithm = input("Введіть алгоритм хешування (за замовчуванням sha256): ") or "sha256"

    found_file = find_file_by_hash(directory, target_hash, algorithm)
    if found_file:
        print(f"Файл знайдено: {found_file}")
    else:
        print("Файл не знайдено.")


import hashlib
import os
import json

def calculate_file_hash(file_path, hash_algorithm="sha256"):
    """
    Обчислює хеш файлу за допомогою вказаного алгоритму хешування.

    :param file_path: Шлях до файлу
    :param hash_algorithm: Алгоритм хешування (sha256, sha1, md5 тощо)
    :return: Хеш файлу або None, якщо алгоритм не підтримується
    """
```

```
    try:
        # Отримання відповідного об'єкта хешування
        hash_func = hashlib.new(hash_algorithm)
    except ValueError:
        print(f"Алгоритм '{hash_algorithm}' не підтримується.")
        return None

    try:
        with open(file_path, "rb") as f:
            # Читання файлу блоками для обробки великих файлів
            while chunk := f.read(8192):
                hash_func.update(chunk)
        return hash_func.hexdigest()
    except FileNotFoundError:
        print(f"Файл '{file_path}' не знайдено.")
        return None
    except Exception as e:
        print(f"Помилка: {e}")
        return None

def find_file_by_hash(directory, target_hash, hash_algorithm="sha256"):
    """
    Пошук файлу за його хешем у заданій директорії.

    :param directory: Шлях до директорії для пошуку
    :param target_hash: Цільовий хеш для пошуку
    :param hash_algorithm: Алгоритм хешування, використаний для обчислення хеша
    :return: Шлях до файлу або None, якщо файл не знайдено
    """
    try:
        for root, _, files in os.walk(directory):
            for file in files:
                file_path = os.path.join(root, file)
                if calculate_file_hash(file_path, hash_algorithm) == target_hash:
                    return file_path
    except Exception as e:
        print(f"Помилка під час пошуку: {e}")
    return None

def create_hash_database(directory, output_file, hash_algorithm="sha256"):
    """
    Створює базу даних хешів і шляхів до файлів у форматі JSON.    :
```