НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

"На правах рукопису"
УДК *004.49*

До захисту допущено
Завідувач кафедри

_____ Олександр РОЛІК

“ ____ ” _____ 20 23 р.

МАГІСТЕРСЬКА ДИСЕРТАЦІЯ

на здобуття ступеня магістра

за освітньо-науковою програмою

«*Комп’ютеризовані системи управління*»

зі спеціальності 151 «*Автоматизація та комп’ютерно-інтегровані технології*»

на тему:

«Система по формуванню бази сигнатур комп’ютерних вірусів»

Виконав:
студент VI курсу, групи **ІА-61мнв**
Руденко Андрій Сергійович _____

Керівник:
Ст. викладач, доц., к.т.н.,
Шимкович Володимир Миколайович _____

Рецензент:
Доц. каф. ОТ, к.т.н.,
Волокита Артем Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-наукова програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Олександр РОЛІК

«__» _____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Руденку Андрію Сергійовичу

1. Тема дисертації «Система по формуванню бази сигнатур комп'ютерних вірусів»

науковий керівник дисертації Шимкович Володимир Миколайович, доц. каф. ІСТ, к.т.н., затверджені наказом по університету від «17» травня 2023 р. № 1792-с

2. Строк подання студентом дисертації "19" травня 2023 р.

3. Об'єкт дослідження: сигнатури комп'ютерних вірусів.

4. Предмет дослідження: методи та засоби виявлення, аналізу шкідливого програмного забезпечення та збору сигнатур комп'ютерних вірусів.

5. Перелік завдань, які потрібно розробити: огляд та аналіз існуючих методів виявлення та аналізу шкідливого програмного забезпечення; аналіз існуючих віртуальних середовищ автоматизованого аналізу зразків; формування вимог та сценаріїв використання системи; розробка структури програмно-технічного комплексу на базі обраної системи автоматизованого аналізу зразків; дослідження технологій та алгоритмів для формування ефективних сигнатур шкідливого ПЗ;

розробка та розгортання компонентів системи у відповідності до розробленої структури комплексу; дослідження результатів роботи системи.

6. Перелік графічного (ілюстративного) матеріалу

Діаграма сценаріїв використання системи; діаграма сценаріїв використання підсистеми керування сигнатурами; структурна схема системи; алгоритм роботи компонентів підсистеми автоматичного аналізу зразків; діаграма послідовності; діаграма розгортання системи; структура бази даних системи.

7. Орієнтовний перелік публікацій

8. Консультанти розділів дисертації

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

9. Дата видачі завдання “ 10 ” березня 2023 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1	Огляд та аналіз існуючих методів виявлення та аналізу шкідливого програмного забезпечення, аналіз існуючих віртуальних середовищ автоматизованого аналізу зразків обрання на підставі проведених досліджень інструменту для побудови системи	12.03.2023-29.03.2023	
2	Формування вимог та сценаріїв використання системи	30.03.2023-02.04.2023	
3	Розробка структури програмно-технічного комплексу	03.04.2023-07.04.2023	
4	Дослідження технологій та алгоритмів для формування ефективних сигнатур шкідливого ПЗ	07.04.2023-11.04.2023	
5	Розробка та розгортання компонентів системи	04.04.2023-20.04.2023	
7	Дослідження результатів роботи системи	20.04.2023-22.04.2023	
8	Оформлення документації	10.04.2023-29.04.2023	
9	Подання роботи на попередній захист	15.05.2023	
10	Подання роботи на основний захист	19.05.2023	

Студент _____ *Андрій РУДЕНКО*

Науковий керівник _____ *Володимир ШИМКОВИЧ*

РЕФЕРАТ

Магістерська дисертація: 116 с., 32 рис., 17 табл., 1 додаток, 51 джерело.

У зв’язку зі стрімким зростанням кількості шкідливого програмного забезпечення для персональних пристроїв користувачів важливим є постійно вдосконалювати та адаптувати методи захисту від цих загроз.

Метою дослідження є підвищення ефективності антивірусного захисту за рахунок створення моделі програмно-технічного комплексу для збору, автоматичного дослідження зразків комп’ютерних вірусів та формуванню бази їх сигнатур.

Об’єктом дослідження є сигнатури комп’ютерних вірусів.

Предметом дослідження є методи та засоби виявлення, аналізу шкідливого ПЗ та збору сигнатур комп’ютерних вірусів.

У ході дослідження розроблено модель системи, яка дозволяє здійснювати автоматичний збір та дослідження зразків шкідливих комп’ютерних програм та забезпечує вірусним аналітикам зручні інструменти для формування бази вірусних сигнатур. Наповнення бази сигнатур новими зразками шкідливих програм дозволяє підвищити ефективність антивірусного захисту.

Практичне значення роботи полягає в тому, що система може інтегруватися в існуючі рішення в сфері кібербезпеки для дослідження нових загроз та розширення існуючої антивірусної бази.

КІБЕРБЕЗПЕКА, КОМП’ЮТЕРНИЙ ВІРУС, ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АНТИВІРУС, СИГНАТУРА ВІРУСУ, СТАТИЧНИЙ АНАЛІЗ, АНАЛІЗ ПОВЕДІНКИ.

ABSTRACT

The master's thesis: 116 pages, 32 figures, 17 tables, 1 appendix, 51 sources.

Due to the rapid growth of malicious software on users' personal devices, it is important to constantly improve and adapt the methods of protection against these threats.

The object of research is the signatures of computer viruses.

The subject of research is methods and ways of detection, analysis of malicious software and collection of signatures of computer viruses.

A model of a system for automatic collection and analysis of malicious computer programs samples was developed during the research. It provides convenient tools for virus analysts to build a malware signature database. Filling the signature database with new samples of malicious programs allows you to increase the effectiveness of antivirus protection.

The practical significance of the work is that the system can be integrated into existing cyber security solutions to investigate new threats and expand the existing antivirus database.

CYBER SECURITY, COMPUTER VIRUS, MALWARE, ANTIVIRUS, VIRUS SIGNATURE, STATIC ANALYSIS, BEHAVIORAL ANALYSIS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 8

ВСТУП 9

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ 12

1.1 Поняття шкідливого програмного забезпечення 12

1.1 Типи шкідливого програмного забезпечення 13

1.2 Способи виявлення шкідливого програмного забезпечення 19

1.3 Методи аналізу шкідливого програмного забезпечення 25

1.3.1. Статичний аналіз 26

1.3.2. Динамічний аналіз 29

1.4 Огляд віртуальних середовищ для автоматизованого аналізу зразків 30

1.4.1 Cuckoo Sandbox 30

1.4.2 Joe Sandbox 32

1.4.3 VMRay Analyzer 34

2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ 37

2.1 Функціональні вимоги 37

2.2 Нефункціональні вимоги 38

3 РОЗРОБЛЕННЯ СЦЕНАРІЇВ ВИКОРИСТАННЯ 41

3.1 Розроблення діаграми варіантів використання вірусним аналітиком 41

3.2 Розроблення діаграми варіантів використання адміністратора 48

4 МОДЕЛЬ СИСТЕМИ ПО ФОРМУВАННЮ БАЗИ СИГНАТУР КОМП'ЮТЕРНИХ ВІРУСІВ 52

5 ВИБІР ТА ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ 58

5.1 Вибір хмарної платформи 58

5.2 Обґрунтування вибору системи аналізу шкідливого ПЗ 62

5.3 Інструмент YARA для ідентифікації шкідливого програмного забезпечення 65
5.4 Алгоритм нечіткого хешування для пошуку подібних зразків 67
5.5 Обґрунтування вибору системи віртуалізації 69
5.6 Засоби реалізації сховища для зберігання зразків шкідливого ПЗ 70
5.7 Вибір бази даних для зберігання сигнатур 73
5.8 Вибір додаткових технологій 74
6 РОЗГОРТАННЯ СИСТЕМИ ФОРМУВАННЯ БАЗИ СИГНАТУР КОМП'ЮТЕРНИХ ВІРУСІВ 77
6.1 Архітектура підсистеми зберігання зразків 77
6.2 Архітектура підсистеми автоматичного аналізу зразків 80
6.3 Розгортання віртуального середовища для автоматизованого аналізу зразків 83
6.4 Застосування алгоритму нечіткого хешування модулем генерації сигнатур 89
6.5 Розробка веб-інтерфейсу підсистеми управління сигнатурами 92
6.6 Розгортання бази даних вірусних сигнатур 94
7 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ 97
ВИСНОВКИ 101
ПЕРЕЛІК ПОСИЛАНЬ 103
ДОДАТОК А. Графічний матеріал 108
Плакат 1. Діаграма сценаріїв використання системи 109
Плакат 2. Діаграма сценаріїв використання підсистеми керування сигнатурами 110
Плакат 3. Структурна схема системи 111
Плакат 4. Алгоритм роботи модуля керування аналізом 112
Плакат 5. Алгоритм роботи модуля статичного аналізу 113

Плакат 6. Діаграма послідовності роботи системи..... 114

Плакат 7. Діаграма розгортання системи..... 115

Плакат 8. Структура бази даних сигнатур..... 116

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AWS – Amazon Web Services

CTPH – Context Triggered Piecewise Hashing

GCP – Google Cloud Platform

MVC – Model-View-Controller

PE – Portable Executable

PUA/PUP – потенційно небажане програмне забезпечення

YARA – Yet Another Recursive Algorithm

ФС – файлова система

ОС – операційна система

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

ПС – програмна система

ВСТУП

Разом зі стрімким розвитком Інтернет-технологій та збільшенням кількості різноманітних розумних пристроїв перелік потенційних небезпек, спрямованих на користувачів цих пристроїв також зростає. Різноманітні комп'ютерні віруси та програми-шпигуни можуть завдати значних збитків користувачам, спричинити збої в роботі комп'ютера, призвести до втрати конфіденційної інформації та інших негативних наслідків. Також існують програми, які можуть здійснювати різноманітні шкідливі активності без згоди користувачів, як підписування користувачів на поштові розсилки, здійснення покупок в Інтернет-магазинах, відображення небажаної реклами. Зловмисники постійно вдосконалюють свої методи нападу та розробляють нові види вірусів та шкідливих програм, що може зробити традиційні методи захисту застарілими та неефективними. Інститут AV-TEST щодня реєструє понад 450 000 нових шкідливих програм («Malware»), а також потенційно небажаних програм («PUA») [1]. На рисунку 1 видно тенденцію щорічного зростання кількості шкідливих програм і PUA.

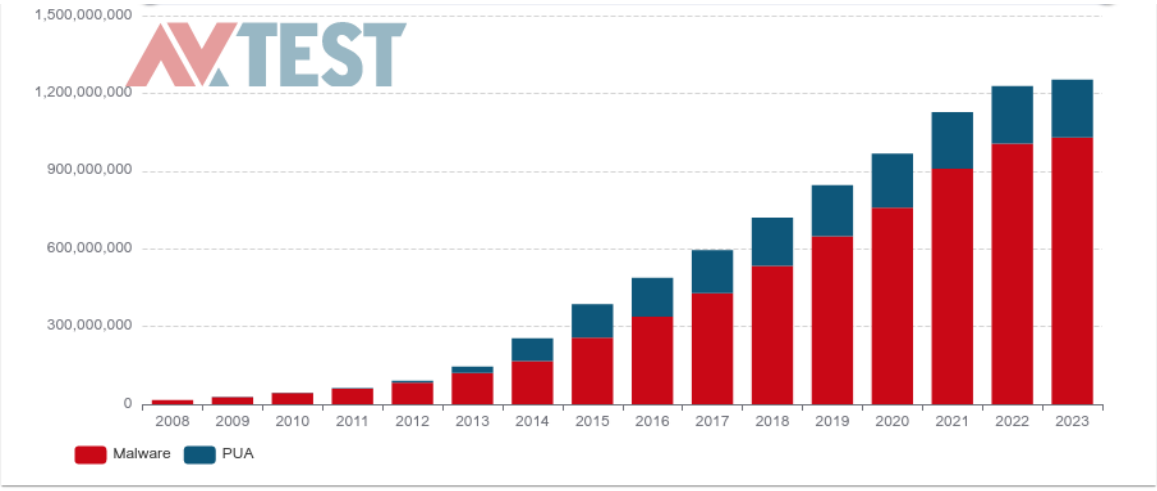

Рисунок 1 – Загальна кількість шкідливого ПЗ і PUA[1]

Тому для ефективного захисту від нових загроз необхідно постійно вдосконалювати та адаптувати методи захисту від цих загроз.

Якщо розглядати захист персональних комп’ютерів, то основним методом боротьби зі шкідливим програмним забезпеченням є використання антивірусів. Хоча існують й інші методи захисту, такі як міжмережеві екрани та інструменти аналізу поведінки, антивіруси залишаються найпоширенішим, найдешевшим та одним з найефективніших методів захисту.

Більшість сучасних антивірусних програм забезпечують захист від вірусів шляхом розпізнавання їхніх сигнатур. Сигнатура вірусу – це певний шаблон або кодова послідовність, унікальна для певного комп’ютерного вірусу чи зловмисного програмного забезпечення [2]. Антивірусне програмне забезпечення використовує ці сигнатури для ідентифікації та виявлення наявності відомих вірусів. Однак, з появою нових вірусів, існуючі сигнатури можуть виявитися недостатньо ефективними. Антивіруси на основі сигнатур мають свої недоліки, основним з яких є обмежена ефективність виявлення нових вірусів, які ще не занесені до бази даних. Крім того, такі антивіруси можуть спрацьовувати з певною затримкою, оскільки потребують періодичного оновлення бази даних сигнатур. Також деякі віруси можуть бути змінені таким чином, що їх сигнатури стають нерозпізнаваними, що також зменшує ефективність захисту. Тому перед розробниками та фахівцями в сфері кібербезпеки постає завдання в постійному вдосконаленні наявних інструментів формування сигнатур для якнайшвидшого виявлення нових загроз.

Постійна робота над вдосконаленням своїх продуктів та постійна актуалізація вірусних баз є важливою складовою для ведення успішного бізнесу розробників антивірусного програмного забезпечення. Також важливо бути в курсі останніх тенденцій та загроз у світі кібербезпеки, щоб вчасно реагувати на нові види вірусів та шкідливих програм. Це дозволяє залучити нових клієнтів та збільшити свою частку на ринку.

Метою дослідження є підвищення ефективності антивірусного захисту за рахунок створення моделі програмно-технічного комплексу для збору, автоматичного дослідження зразків комп’ютерних вірусів та формуванню бази їх сигнатур.

Під час дослідження були виконані наступні задачі:

- дослідження існуючих методів виявлення та аналізу шкідливого програмного забезпечення;
- аналіз існуючих віртуальних середовищ автоматизованого аналізу зразків і обрання на підставі проведених досліджень інструменту для побудови системи, що проектується;
- формування вимог та сценаріїв використання майбутньої системи;
- розробка структури програмно-технічного комплексу на базі обраної системи автоматизованого аналізу зразків;
- дослідження технологій та алгоритмів для формування ефективних сигнатур шкідливого ПЗ;
- розробка та розгортання компонентів системи у відповідності до розробленої структури комплексу;
- дослідження результатів роботи системи.

Об’єкт дослідження – сигнатури комп’ютерних вірусів.

Предмет дослідження – методи та засоби виявлення та аналізу шкідливого ПЗ та збору сигнатур комп’ютерних вірусів.

Наукова новизна:

У ході дослідження розроблено модель програмно-технічного комплексу, який дозволяє здійснювати автоматичний збір та дослідження зразків шкідливих комп’ютерних програм та забезпечує вірусним аналітикам зручні інструменти для формування бази вірусних сигнатур. Використання алгоритму нечіткого хешування для кластеризації зразків та методів автоматизації дозволяє зробити процес формування сигнатур більш ефективним, що в кінцевому результаті дозволяє підвищити ефективність антивірусного захисту.

Практичне значення роботи полягає в тому, що система може інтегруватися в існуючі рішення в сфері кібербезпеки для дослідження нових загроз та розширення існуючої антивірусної бази.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Поняття шкідливого програмного забезпечення

Шкідливе програмне забезпечення (англ. Malware, утворене поєднанням слів «malicious» («зловмисний») і «software» («програмне забезпечення»)) – це програмне забезпечення, призначене для перешкоджання нормальному функціонуванню комп'ютера, мережі або інших пристроїв, а також для компрометації конфіденційної інформації, що зберігається в системі [3]. Може проявлятися у вигляді файлу, коду, скрипту, байтової послідовності тощо. Зазвичай передається, використовуючи мережу Інтернет.

Комп'ютерний вірус – це тип шкідливого програмного забезпечення, яке може приєднується до інших програм, розмножуватися та поширюватися без дозволу користувача з одного комп'ютера на інший. Коли вірус заражає комп'ютер, він створює свої копії та приєднується до інших файлів чи документів. Потім він змінює ці файли та продовжує поширюватися [4].

Віруси заражають пристрої непомітно, найчастіше вони призначені для знищення особистих файлів або отримання контролю над пристроями. Створюючи копії самого себе, комп'ютерні віруси поширюються між пристроями та мережами, як біологічні віруси, що передаються від однієї людини до іншої. Як і їх біологічні версії, деякі комп'ютерні віруси просто дратують людей і не створюють реальної загрози безпеці, а інші можуть завдати значної шкоди як особистим пристроям користувачів так і задати величезних збитків [5] великим компаніям.

Часто люди ототожнюють поняття «шкідливе програмне забезпечення» і «комп'ютерний вірус». Але технічно ці два терміни відрізняються один від одного [6]. Вірус є одним з типів шкідливого ПЗ, ключовою особливістю якого є механізм приєднання та самовідтворення. Він заражає файли, а потім поширюється через пристрій під час кожного запуску файлу чи програми.

1.1 Типи шкідливого програмного забезпечення

В даному розділі розглядаються найпопулярніші типи шкідливого програмного забезпечення, пояснюються відмінності між ними та пов’язані з ними небезпеки.

Одним з найбільш поширених завдань, які виконують аналітики шкідливого програмного забезпечення, є первинне сортування або класифікація зразків шкідливого ПЗ. Часто можна скоротити час аналізу, зробивши осмислені припущення про те, що намагається зробити це програмне забезпечення, і потім підтверджуючи ці гіпотези.

Класифікація може бути простою, наприклад, виявлення типу файлу, або складнішою, як виявлення процентної подібності з іншими зразками та визначення спільних рис поведінки між варіантами того ж самого шкідливого програмного забезпечення.

Далі описано основні типи шкідливого програмного забезпечення, пояснено відмінності між ними і описані загрози, які вони несуть.

Програми типу *«Троянський кінь»* (англ. Trojan Horse, Trojan). Це тип шкідливого програмного забезпечення, яке маскується під звичайну доброякісну програму, але насправді таємно від користувача виконує додаткову шкідливу діяльність. Ключовою особливістю, яка відрізняє даний тип шкідливого ПЗ від інших, є спосіб його поширення. Трояни не можуть копіювати себе в інші файли і не поширюються самостійно через мережу, як віруси і черви. Трояна, наприклад, можуть містити деякі безкоштовні програми для роботи з медіа, піратське програмне забезпечення на недобросовісному веб-сайті тощо.

Вірус – шкідливе програмне забезпечення, яке заражає комп’ютери та інші файли шляхом саморозмноження. Він не може існувати самостійно, тому приєднується до інших виконуваних файлів і програм. Інфікований код в свою чергу намагається заразити новий код. Самовідтворення є головною особливістю таких шкідливих програм.

«Мережеві черв'яки» (англ. Worms). Черв'яки мають спільні особливості з вірусами, але найважливіша відмінність полягає в їх методах поширення та впливі на систему. Віруси зазвичай потребують наявності хост-файлу для інфікування, тоді як черв'яки можуть самостійно розповсюджуватися між комп'ютерами по мережі без участі користувача, використовуючи вразливості в ОС. Черв'яки також можуть поширюватися іншими способами — наприклад, масово надсилаючи себе електронною поштою на кожну електронну адресу в адресній книзі інфікованого користувача. Вони зазвичай завдають шкоди своїм хост-мережам, зменшуючи пропускну здатність і перевантажуючи веб-сервери. Часто черв'яки можуть містити в собі корисне навантаження, поширюючи так звані «бекдори». Можливість самовідтворення та велика швидкість поширення дозволяє зловмисникам створити цілі мережі підконтрольних машин, які називають ботнетами. Такі мережі можуть використовуватися для проведення DDoS-атак (розподілена атака з відмовою в обслуговуванні) та поширення спаму [7].

«Бекдори» (англ. Backdoor) – це програмний код, який дозволяє зловмисникам отримувати несанкціонований доступ до комп'ютерної системи, програми або мережі шляхом обходу аутентифікації або інших стандартних методів та технологій безпеки. Після встановлення бекдор отримує доступ до керування пристроєм, може викрасти конфіденційну інформацію та використовувати комп'ютер для спаму або запуску DDoS-атак [8]. Даний тип шкідливого ПЗ найчастіше потрапляє на пристрій жертви під час завантаження файлів користувачем. Зокрема, одна із наймасштабніших кібератак в Україні була здійснена бекдором NotPetya, який був поширений через оновлення популярного програмного забезпечення «M.E.Doc». Загалом сума збитків, спричинених даною вірусною атакою сягнула 10 млрд доларів [9].

Програми-шпигуни (англ. Spyware). Це тип шкідливого програмного забезпечення, яке приховано збирає конфіденційну інформацію про користувачів без їхньої згоди. Spyware може відстежувати активність користувача в Інтернеті, збирати інформацію про веб-історію, паролі, банківські дані, адреси електронної пошти, а також контролювати дії користувачів, наприклад, записувати натискання

клавіш і копіювати знімки екрану. Інформація, зібрана шпигунською програмою, відправляються сторонньому суб'єкту чи організації та може бути використана для реклами, злочинної діяльності, шахрайства, викрадення особистої інформації і т.д. Такі програми зазвичай не здатні до само розмноження та поширюються у вигляді безкоштовного або умовно-безкоштовного програмного забезпечення з мінімальними ліцензійними обмеженнями, щоб охопити якомога більше користувачів.

Рекламне програмне забезпечення (англ. Adware) – це програмне забезпечення, яке показує користувачам рекламу з метою заробітку для розробників. Часто це може бути безкоштовним програмним забезпеченням, яке фінансується через показ реклами. Adware може використовувати різні способи показу реклами, включаючи вспливаючі вікна, банери на веб-сторінках, рекламу в програмах і т.д. Хоча такі програми самі по собі не є шкідливим, вони можуть бути надокучливим та нав'язливим, переривати роботу користувача та сповільнювати пристрій. Крім того, деякі види adware можуть бути пов'язані зі збором особистої інформації користувача, яка може використовуватися для персоналізованої реклами або продажу третім сторонам [10].

Ransomware – це програма, яка блокує доступ до файлів на комп'ютері користувача шляхом їх шифрування та вимагає від користувача викуп для повернення доступу до своїх даних. Розшифрування заблокованих даних можливе лише з допомогою ключа для розшифрування, який знаходиться у розпорядженні зловмисників. Поява криптовалют призвела до значного поштовху у розвитку програм-вимагачів. Адже платежі в такій валюті здійснюються анонімно і їх майже не можливо відслідкувати [11].

Потенційно небажані програми (англ. Potentially Unwanted Programs, PUP) – це програми, які можуть бути встановлені на комп'ютер без згоди користувача або бути встановленими разом з іншими програмами, що завантажуються з Інтернету. Хоча вони і не є шкідливими програмами в повному значенні цього слова, але можуть містити рекламу, панелі інструментів і спливаючі вікна, не пов'язані із завантаженим програмним забезпеченням.

За способом зараження комп’ютерні віруси можна розділити на *резидентні* і *нерезидентні*. Резидентний вірус після проникнення в комп’ютер тримає свій код в оперативній пам’яті. Небезпека полягає в тому, що він перехоплює усі звернення операційної системи до файлів та програм та може інфікувати будь-яку з них. Резидентні віруси можуть заблокувати деякі функції системи, внести зміни у реєстрі або внести інші зміни, що дозволяють їм залишатися прихованими та уникати виявлення антивірусним програмним забезпеченням. Залишаючись в пам’яті, дані віруси продовжують свою діяльність на пристрої до того моменту, поки користувач не вимкне або перезавантажить систему. Нерезидентні віруси не зберігаються в пам’яті комп’ютера постійно, вони можуть заражати файли безпосередньо при їх відкритті або виконанні.

Для підсумування особливостей поширення кожного з типі шкідливих програм наведемо в таблиці 1.1 основні характеристики, присутні шкідливому програмному забезпеченню [12]:

- розмноження або самореплікація – здатність шкідливо ПО швидко поширюватись шляхом створення копій самого себе;

- паразитизм – це можливість існування частини коду вірусу в іншому середовищі; це може бути boot-запис диска, двійковий код програми, вихідний код, файл, скрипт, макрос.

Таблиця 1.1 – Характеристики різних типів шкідливого ПЗ

Тип шкідливого ПЗ	**Розмноження**	**Паразитизм**
Троянський кінь	ні	так
Черв’як	так	ні
Вірус	так	так
Бекдор	ні	можливо
Програми-шпигуни	ні	ні
Рекламне програмне забезпечення	ні	ні
Програма-вимагач	ні	ні

Існує класифікація комп’ютерних вірусів в залежності від того, який об’єкт вони заражають [12]. Серед них можна виділити наступні.

Файлові віруси прикріплюються до виконуваних файлів, документів або інших програм і поширюються через них. При запуску зараженого файлу вірус поширюється на інші файли на комп’ютері та може спричинити їх пошкодження або видалення.

Завантажувальні віруси заражають boot-сектор завантажувального диска. Дані віруси підміняють код програми, яка контролює запуск операційної системи, тому після завантаження системи контроль передається вірусу. Цей тип вірусів в даний час зустрічається дуже рідко.

Макро-віруси впливають на програми обробки текстів, такі як Microsoft Word або Excel. Вони заражають макроси в документах та можуть поширюватися через електронну пошту або файлові хостинги.

Комп'ютерні черви поширюються через мережу Інтернет і не потребують наявності файлу для інфікування.

Розглянемо основні ознаки прояву шкідливих програм:

- підвищене використання ЦП;
- повільна робота пристрою;
- затримки при запуску програм;
- зависання або збої;
- програми запускаються автоматично;
- зміна або видалення файлів;
- неможливість збереження файлів;
- сповільнення роботи мережі;
- неспроможність завантажити комп’ютер;
- поява незрозумілих системних повідомлень, звукових ефектів тощо;
- зміна домашньої сторінки або пошукової системи за замовчуванням у веб-браузері без дозволу користувача;
- нові панелі інструментів, розширення або плагіни у веб-браузері;

- розсилка сумнівних повідомлені в соціальних мережах та мессенджерах від імені користувача.

За ступенем небезпеки, яку несуть комп’ютерні віруси їх можна розділити на наступні категорії:

- *нешкідливі*: ці віруси зазвичай не завдають значної шкоди комп’ютеру і можуть бути легко видалені без втрати даних;

- *безпечні*: вплив може обмежитись появою різних рекламних банерів, звукових ефектів, зменшенням вільної пам’яті тощо;

- *небезпечні*: здатні призвести до зниження продуктивності та збоїв в роботі пристрою користувача, але не призводять до порушення цілісності та конфіденційності даних;

- *дуже небезпечні*: є найбільш шкідливими та небезпечними, можуть призвести до серйозної втрати даних та навіть до повної недоступності комп’ютера; вони можуть поширюватися дуже швидко та ефективно, а видалення їх може бути дуже складним та вимагати значних зусиль та досвіду.

На рисунку 1.1 схематично зображено вищеописану класифікацію шкідливого ПЗ.

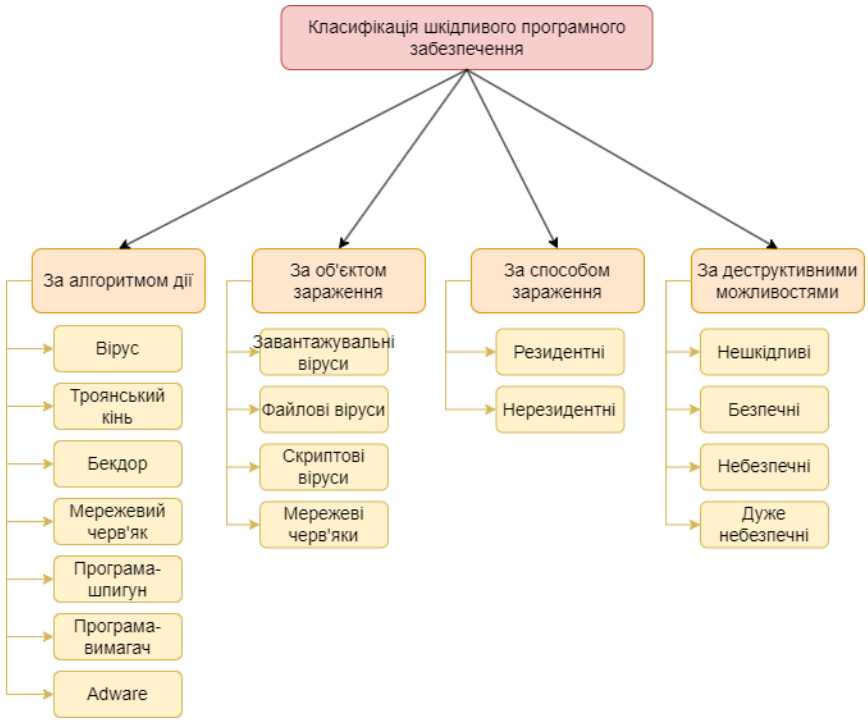

Рисунок 1.1 – Узагальнена класифікація шкідливого ПЗ

1.2 Способи виявлення шкідливого програмного забезпечення

Виявлення зловмисного програмного забезпечення передбачає використання методів і інструментів для виявлення, блокування, попередження та реагування на загрози. Серед основних методів виявлення шкідливого програмного забезпечення можна виділити три основні категорії: на основі сигнатур, на основі аналізу поведінки та на основі евристичного аналізу. Розглянемо детальніше кожну з цих категорій.

Аналіз на основі сигнатур

Наразі більшість антивірусних систем для детектування шкідливого ПЗ використовують сигнатурний аналіз. Сигнатура – це унікальний фрагмент коду, який завжди зустрічається тільки у певному шкідливому файлі або вірусі і ніколи в інших програмах. Цей код може містити певні характеристики, які відрізняють вірус від інших файлів, а також інформацію про те, як вірус поводиться в системі. Серед основних характеристик, що складають сигнатуру, можна виділити [13]:

- унікальна послідовність байтів, характерна тільки для цієї програми;
- хеш окремого файлу чи програми цілком;
- унікальний рядок в коді програми;
- м’ютекси – унікальні для сімейства зловмисних програм маркери, які використовуються для того, щоб перевірити, чи скомпрометовано вже даний пристрій;
- ключі реєстру – часто створюються шкідливим програмним забезпеченням як механізм збереження власних даних;
- шлях до системної бібліотеки до якої звертається програма;
- шляхи до програмної бази даних, яка містить різноманітну службову інформацію для налагодження програм;
- зашифровані рядки конфігурації – зловмисне програмне забезпечення часто шифрує свою конфігурацію, яка може містити такі важливі ідентифікатори, як протокол, домен, порт, на який програма виконує запити.

На рисунку 1.2 описана загальна процедура виявлення шкідливого ПЗ на основі сигнатурного аналізу.

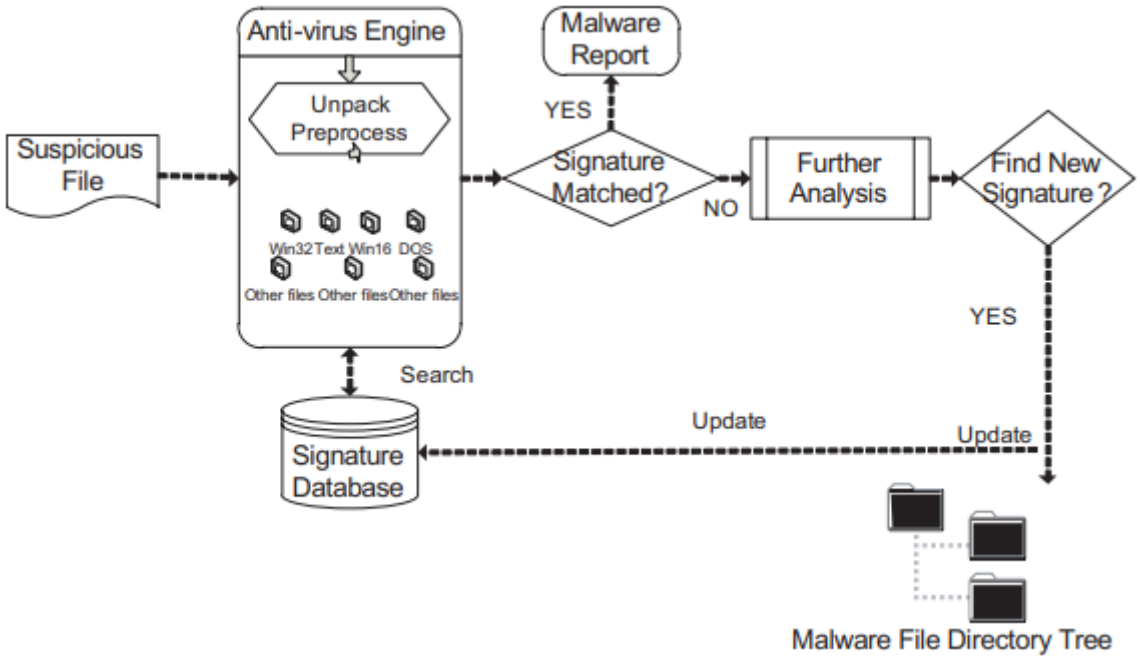

Рисунок 1.2 - Процедура виявлення шкідливого ПЗ на основі сигнатур [14]

Після отримання підозрілого виконуваного файлу від відправляється на аналіз антивірусному механізму. Антивірус спочатку перевіряє чи упакований зразок чи ні. Якщо він упакований, то механізм розпаковує його і передає розпаковану версію на відповідний сканер антивірусного ядра. Сканер – це окрема частина механізму, яка працює з певним типом файлу. Розпакування виконуваного файлу – це функція, вбудована в антивірусну систему, що служить для видобутку вихідних кодів виконуваної програми і подальшої передачі їх на сканування. Далі антивірусний механізм оцінює асемблер-код файлу, порівнюючи конкретні байти коду з інформацією в своїй базі даних сигнатур. Якщо антивірусний продукт виявляє файл з сигнатурою, яка відповідає відомому вірусу, то він вважає цей файл шкідливим та в залежності від налаштувань захисту може виконати один з наступних кроків:

- видалити інфікований файл;

- відправити файл у карантин (тимчасове сховище файлів, в якому відбувається шифрування шкідливого файлу для запобігання його подальшого запуску та розповсюдження);

- антивірус може спробувати “вилікувати файл”, видаляючи з нього шкідливий код, що дозволяє залишити корисний вміст файлу.

Якщо в цьому файлі не знайдено відповідної сигнатури, тоді підозрілий файл передається антивірусним інженерам для подальшого аналізу і оновлення бази даних сигнатур зловмисного програмного забезпечення [14].

На рисунку 1.3 зображено приклад частини сигнатури вірусу Casper [15], описаний з використанням правил YARA («Yet Another Recursive Algorithm») [16].

```
rule Casper_Included_Strings
{
    meta:
        description = "Casper French Espionage Malware - String Match in File - http://goo.gl/VRJNLo"
        author = "Florian Roth"
        reference = "http://goo.gl/VRJNLo"
        date = "2015/03/06"

    strings:
        $a0 = "cmd.exe /C FOR /L %%i IN (1,1,%d) DO IF EXIST"
        $a1 = "& SYSTEMINFO) ELSE EXIT"
        $mz = { 4d 5a }
        $c1 = "domcommon.exe" wide fullword                         // File Name
        $c2 = "jpic.gov.sy" fullword                                // C2 Server
        $c3 = "aiomgr.exe" wide fullword                            // File Name
        $c4 = "perfaudio.dat" fullword                              // Temp File Name
        $c5 = "Casper_DLL.dll" fullword                             // Name
        $c6 = { 7B 4B 59 DE 37 4A 42 26 59 98 63 C6 2D 0F 57 40 }   // Decryption Key
        $c7 = "{4216567A-4512-9825-7745F856}" fullword              // Mutex
    condition:
        all of ($a*) or ( $mz at 0 ) and ( 1 of ($c*) )
}
```

Рисунок 1.3 – Сигнатура для виявлення вірусу Casper [17]

На сьогодні сканування на основі сигнатур є одним з основних методів роботи антивірусного ПЗ. Оновлення сигнатур є важливою частиною захисту від шкідливих програм, оскільки це дозволяє антивірусному програмному забезпеченню підтримувати базу даних сигнатур вірусів у відповідності з новими загрозами, які з’являються в мережі Інтернет.

Недоліком системи виявлення на основі сигнатур є те, що для отримання сигнатури необхідно мати зразок вірусу, тому вона може виявити тільки існуюче

шкідливе ПЗ, яке вже є в базі сигнатур. Іншим недоліком є розмір і обслуговування бази даних сигнатур. З появою нових зразків зловмисного програмного забезпечення база даних сигнатур зростає лавиноподібним чином, оскільки для кожного зразка потрібна окрема сигнатура. Частково ця проблема вирішується написанням універсальних сигнатур, які описують ціле сімейство схожих за поведінкою вірусів. Однак при цьому існує ймовірність помилкових спрацьовувань. Необхідність частого та регулярного оновлення бази даних сигнатур є також одним з недоліків, тому вірусні аналітики потребують інструменти, які дозволять спростити та автоматизувати процес аналізу зразків шкідливого ПЗ.

Серед основних переваг даного методу виявлення шкідливого ПЗ є те, що для аналізу зразка немає необхідності запускати виконуваний файл. Це важливо, оскільки більшість вірусів розпочинають свій цикл життя після запуску виконуваного файлу, до якого вони приєднані. Цей метод забезпечує високу точність виявлення вже відомих зразків при невеликих витратах ресурсів. Також цей метод забезпечує найбільшу швидкодію в порівнянні з іншими.

Аналіз поведінки

На відміну від попереднього методу, процес детектування небезпечного програмного забезпечення відбувається шляхом аналізу поведінки об’єкта. Для цього в режимі реального фіксуються всі дії, які виконує програма або процес. Це може бути, наприклад:

- спроби взаємодії з системними файлами або процесами;
- спроби форматування розділів накопичувача або інші незворотні операції з диском;
- спроби відкрити або видалити файли;
- намагання модифікувати виконувані файли, скрипти або макроси;
- модифікація конфігураційних файлів або файлів реєстру ОС;
- мережева активність.

Якщо детектор виявляє аномалії в поведінці досліджуваної програми, то він помічає її як потенційно небезпечну, сповіщає про це та вживає заходів для запобігання пошкодження системи. За алгоритмом роботи аналізатори поведінки дуже схожі на системи виявлення вторгнень на базі хоста і можуть бути їх складовою [18].

Аналіз поведінки може використовуватися як самостійний метод детекції шкідливого ПЗ, але найбільшу ефективність він має у поєднанні з іншими методами. Основною перевагою є те, що даний метод дозволяє виявляти нові зразки шкідливого програмного забезпечення, яких ще не має в базі антивірусних сигнатур. Тому цей метод підходить для виявлення невідомих і поліморфних вірусів.

Серед недоліків варто зазначити те, що аналізатори даного типу часто мають значний процент хибних спрацювань. Тому важливо враховувати, що програми або антивірусні модулі, які ґрунтуються на методі поведінкового аналізу, можуть видавати значну кількість попереджень. Це може призвести до того, що користувачі перестануть реагувати на всі попередження, що зменшить ефективність виявлення нових загроз. Використання технології поведінкового аналізу не передбачають “лікування” модифікованих вірусом файлів і програм. Тривалий час сканування та більше, в порівнянні з сигнатурним методом, споживання ресурсів є ще одним з недоліків систем, побудованих на основі аналізу поведінки.

Евристичний метод

Як вже зазначалось, методи виявлення шкідливих програм на основі сигнатур і поведінки мають деякі недоліки. Тому щоб подолати ці недоліки запропоновано евристичні методи аналізу. Евристичний аналіз – це метод виявлення шкідливого ПЗ шляхом перевірки коду на наявність підозрілих ознак з використання правил прийняття рішень або методів зважування [19]. Технологія заснована на гіпотетичному припущенні, що нові віруси частково схожі з раніше вивченими зразками. Процес виявлення включає три незалежних етапи: динамічний аналіз, сканування файлів та багатокритеріальний аналіз. Як правило,

спочатку виконується статичний аналіз коду зразка і виділяються ознаки, які притаманні для шкідливих програм. Також додатково виконується динамічний аналіз для виявлення підозрілих активностей. Кожна така ознака або активність має свій коефіцієнт шкідливості (вага). Якщо загальний коефіцієнт шкідливості зразка перевищує певний встановлений ліміт, то даний об’єкт вважається шкідливим.

Евристичні методи дозволяють виявити раніше не відомі зразки шкідливого ПЗ, але як і методи поведінкового аналізу все ще мають високий показник хибних спрацювань. Використання даного методу виявлення шкідливого програмного забезпечення може вимагати значних обчислювальних ресурсів для аналізу поведінки програм та виявлення можливих загроз. Це може призвести до зменшення продуктивності системи, на якій працює антивірусна програма. “Вилікувати” інфіковані вірусом файли стане можливо тільки після вивчення зразка, внесення інформації в базу сигнатур і розробки способу лікування. Евристичне сканування безсильне проти передових новаторських вірусних програм, написаних з чистого аркушу і не схожих на інші комп’ютерні віруси (вразливості нульового дня) [20].

Виділимо сильні і слабі сторони кожного методу детектування шкідливого ПЗ в підсумкову таблицю 1.2.

Таблиця 1.2 – Переваги і недоліки методів виявлення шкідливого ПЗ

Метод	Переваги	Недоліки
Сигнатурний аналіз	- легке виявлення відомих зразків шкідливого ПЗ - мінімальне використання ресурсів	- не може виявити невідомі зразки - потребує регулярного оновлення вірусних баз
Аналіз поведінки	- виявлення невідомих зразків	- значне споживання ресурсів - високий рівень хибних спрацювань
Евристичний аналіз	- виявлення відомих і невідомих зразків	- потребує регулярного оновлення вірусних баз - потребує більше часу і ресурсів для сканування - високий рівень хибних спрацювань

1.3 Методи аналізу шкідливого програмного забезпечення

Що таке аналіз шкідливих програм? Уявімо, що сталося найгірше: ви або ваша організація стали жертвою вірусної атаки. Щоб виявити вектор зараження та усунути заподіяну шкоду з найменшими втратами вам потрібно буде скористатися інструментами та методами аналізу зловмисного програмного забезпечення. Аналіз нових зразків шкідливого програмного забезпечення – це процес визначення призначення та функціональності зловмисного програмного забезпечення. Цей процес дозволяє визначити який тип шкідливої програми заразив вашу систему, шкоду, яку вона здатна спричинити, і, найголовніше, як її видалити. Розуміння поведінки і структури нещодавно виявлених загроз полегшує впровадження захисних функцій розробникам детекторів. Аналіз шкідливих програм здійснюють не тільки антивірусні компанії, але й адміністратори безпеки в різних організаціях. Системні адміністратори повинні аналізувати зловмисне програмне забезпечення, щоб визначити ступінь заподіяної шкоди для системи. Мережеві адміністратори повинні аналізувати його, щоб визначити, наскільки мережа була пошкоджена, а потім намагатися виправити її. Раніше аналіз зразків ПЗ виконувався експертами вручну, цей процес був громіздким і займав багато часу. Сьогодні існує низка інструментів аналізу зловмисного програмного забезпечення з відкритим кодом, які можуть виконувати цей процес автоматично [21].

Аналіз зловмисного програмного забезпечення є вирішальним як для відновлення після кібератак, так і для запобігання таким атакам у майбутньому. Це може допомогти розробникам антивірусного програмного забезпечення виявляти та блокувати нові види шкідливого ПЗ та розробляти методи захисту від атак.

На рисунку 1.4 наведена схема типового процесу аналізу шкідливого ПЗ.

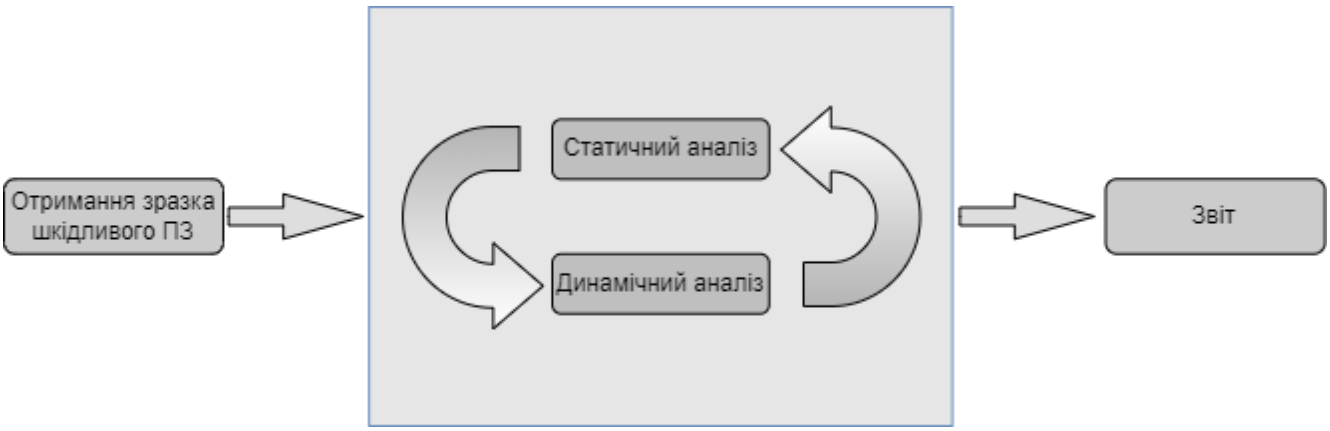

Рисунок 1.4 – Діаграма процесу аналізу шкідливих програм

Далі розглянемо два основних методи аналізу шкідливих програм: статичний і динамічний.

1.3.1. Статичний аналіз

При виконанні статичного аналізу відбувається перевірка зразків шкідливих файлів або програм без їх запуску. Це найбезпечніший спосіб аналізу, оскільки виконання коду може призвести до інфікування системи. Базовий варіант статичного аналізу передбачає аналіз виконуваного файлу без перегляду його коду. При цьому аналізуються такі метадані: тип і розмір файлу, бібліотеки і функції, ресурси, рядки. Також обчислюється контрольна сума (хеш) підозрілого файлу щоб визначити чи досліджувався він раніше антивірусними лабораторіями [22].

Розглянемо основні методи, які використовуються при проведенні статичного аналізу шкідливого ПЗ [23].

Ідентифікація файлу. Зазвичай це перший крок при проведенні статичного аналізу. Зловмисники використовують різні методи (наприклад, використання подвійних розширень), щоб обманом змусити користувачів завантажити й відкрити виконуваний файл, тому дуже важливо мати можливість ідентифікувати шкідливі файли в системі. Даний процес може включає в себе наступні кроки.

Визначення типу файлу, використовуючи його сигнатуру – це унікальний ідентифікаційний номер, розташований на початку файлу. Цей номер визначає тип файлу, надаючи інформацію про дані, які містяться у файлі. Комп'ютер використовує його, щоб визначити, як читати файл або яку програму використовувати для його відкриття[24]. Наприклад, перші два байти виконуваних PE (Portable Executable) файлів сімейства Windows файлу завжди 4D 5A, які є ASCII-кодами літер MZ (ініціали розробника, який винайшов перший виконуваний формат).

Визначення контрольної суми. Контрольна сума (хеш) файлу – деяке значення, яке обчислюється з використанням математичних алгоритмів на основі вмісту файлу. Файли з однаковим вмістом завжди матимуть однакову контрольну суму. Тому хеш використовується для унікальної ідентифікації зловмисного програмного забезпечення. Хеш можна використовувати замість файлу, а також досліджувати у відкритих джерелах, як VirusTotal [25], щоб визначити, чи було дане зловмисне програмне забезпечення попередньо проаналізовано чи ідентифіковано антивірусними механізмами. Популярні хеші: MD5, SHA-1 і SHA-256.

Аналіз вбудованих рядків. Рядки – це символи ASCII і Unicode, вбудовані у файл. Виявлення рядків може дати підказки про функціональні можливості та індикатори, пов'язані зі шкідливим файлом. Дана методологія дозволяє вилучити такі важливі дані як назви файлів, URL-адреси, ключі шифрування ,доменні імена, IP-адреси, ключі реєстру тощо.

Аналіз заголовка PE. Формат Portable Executable – це формат файлів для виконуваних файлів, об'єктного коду, DLL та інших файлів, які використовуються в 32-розрядних і 64-розрядних версіях операційних систем Windows. Заголовок файлу PE складається з заглушки Microsoft MS-DOS, підпису PE, заголовка файлу COFF і додаткового заголовка. Інформація, отримана з заголовка PE, може мати велике значення для аналітика зловмисного програмного забезпечення. Заголовок включає в себе інформацію про час компіляції, ресурси,

статичні й динамічні бібліотеки, функції, які викликаються виконуваним файлом та інші корисну для аналітика інформацію [23].

Більш складний варіант статичного аналізу передбачає зворотну розробку шкідливого програмного забезпечення (англ. Reverse engineering) шляхом дизасемблювання або декомпіляції виконуваного файлу. Дивлячись на асемблер-код зловмисної програми, аналітик може зрозуміти, для чого призначена програма. На жаль, досвідчені хакери вправно уникають цієї техніки. Вбудовуючи певні синтаксичні помилки у свій код, вони можуть неправильно скеровувати дизасемблери, але шкідливий код при цьому продовжує справно працювати [21].

З недоліків можна виділити те, що даний метод вразливий до технік обходу захисту від детекції і статичного аналізу. Серед популярних методів ухилення можна виділити наступні [18].

Шифрування. Віруси можуть використовувати алгоритми шифрування для приховування власного коду. Проте, зашифрований код перед запуском вірусу потрібно розшифрувати і власне код для алгоритму дешифрування сам по собі не може бути зашифрований. Його потрібно буде замаскувати за допомогою іншого методу, як поліморфізм. Яка б техніка шифрування не використовувалась, ключ розшифрування має бути десь усередині вірусу, тому ретельний аналіз алгоритму дешифровки покаже де він.

Поліморфізм. Це властивість вірусів, яка полягає в існуванні великої кількості варіацій власного коду з однаковою базовою функціональністю. Це забезпечується шляхом використання шифрування, перестановок, включення різних комбінацій інструкцій, які не мають жодного результату, наприклад:

- операції з регістрами, які насправді не використовуються алгоритмом;
- нульові операцій NOP або подібні;
- нейтральні групи інструкцій, як інкремент, що слідує за декрементом або здвиг вліво після здвигу вправо.

Такі віруси складно виявити антивірусами, оскільки вони змінюють свій код, а відповідно змінюється і відбиток. Тому для поліпшення ефективності необхідно також застосовувати і динамічний аналіз.

1.3.2. Динамічний аналіз

При виконанні динамічного аналізу (аналізу поведінки) виконується запуск шкідливого програмного забезпечення для перевірки його поведінки. Звичайно, запуск такого ПЗ завжди несе в собі певний ризик, тому динамічний аналіз потрібно виконувати в безпечному середовищі. Емуляція зазвичай відбувається в так званій «пісочниці». Середовище «пісочниця» – це віртуальна система, яка ізольована від решти мережі та може запускати зловмисне програмне забезпечення без ризику для робочих систем. Щоб дослідження було максимально об’єктивним середовище моделює цілком весь хост (включно з процесором, системною пам’яттю та всіма пристроями). Пісочниці можуть робити знімки свого стану, тому після завершення аналізу пісочницю можна легко повернути до початкового непошкодженого стану.

Після запуску шкідливого ПЗ у віртуальному оточенні фіксуються такі важливі дані, як мережевий трафік, системні виклики, зміни файлів, нові ключі реєстру, IP-адреси, доменні імена тощо. Все це використовується для визначення поведінки зловмисної програми, індикаторів компрометації та потенційного впливу на реальну систему [23].

Метод аналізу з використанням ізольованого середовища є особливо ефективним, оскільки він є проактивним і забезпечує високий рівень виявлення загроз без ризику для хост-машини. Деякі зловмисні програми можуть виявити, що вони працюють в пісочниці та змінити свою поведінку, приховавши шкідливі дії. Це може ускладнити аналіз [26]. Серед недоліків також можна виділити те, що аналіз може бути повільнішим, оскільки потрібен час на підготовку віртуального середовища.

1.4 Огляд віртуальних середовищ для автоматизованого аналізу зразків

Кожен з методів аналізу шкідливого ПЗ має свої недоліки, тому для побудови нашої системи варто обрати гібридне рішення, що поєднує обидва методи. Програмні комплекси на основі пісочниці реалізують найкращі особливості кожного з методів і дають можливості їх розширити. Крім того, пісочниці дозволяють класифікувати знайдені типи загроз, що значно спрощує розробку сигнатур вірусним аналітикам. Нижче ми розглянемо найпопулярніші рішення в даній сфері: Cuckoo Sandbox, Joe Sandbox, VMRay Analyzer.

1.4.1 Cuckoo Sandbox

Cuckoo Sandbox – це модульна автоматизована система з відкритим вихідним кодом для аналізу шкідливих програм. Проєкт розроблений в 2010 році в рамках ініціативної програми від Google “Google Summer of Code” і продовжує розвиватися до цих пір [27].

Цей інструмент використовується для автоматизації аналізу шкідливого коду, дозволяючи аналітикам запускати тести на кілька підозрілих файлів одночасно та в різних віртуалізованих середовищах. Пісочниця записує всю активність, спровоковану досліджуваним файлом, а потім створює звіт про те, що зловмисне програмне забезпечення намагалося зробити в цьому безпечному середовищі [28].

Cuckoo можна встановити на ОС сімейства Windows, Linux і Mac Os і використовувати для аналізу вкладену віртуалізацію або навіть фізичні машини. Перевага використання віртуалізаації полягає в тому, що аналітик може швидко створювати або видаляти віртуальні машини, а Cuckoo автоматизує процес запуску та повернення машин до попередньо визначених станів (знімків системи).

Короткий перелік можливостей Cuckoo Sandbox:

- моніторинг викликів win32 API функцій;

- дамп мережевої активності;
- створення скріншотів в ході виконання аналізу;
- збереження копій всіх створених файлів у процесі перевірки;
- трасування інструкцій, що виконуються шкідливим процесом;
- створення зручного звіту у txt, JSON та HTML-форматах;
- абсолютна ізольованість середовища, в якому проводиться запуск шкідливих програм.

Архітектуру Cuckoo зображено на а рисунку 1.5.

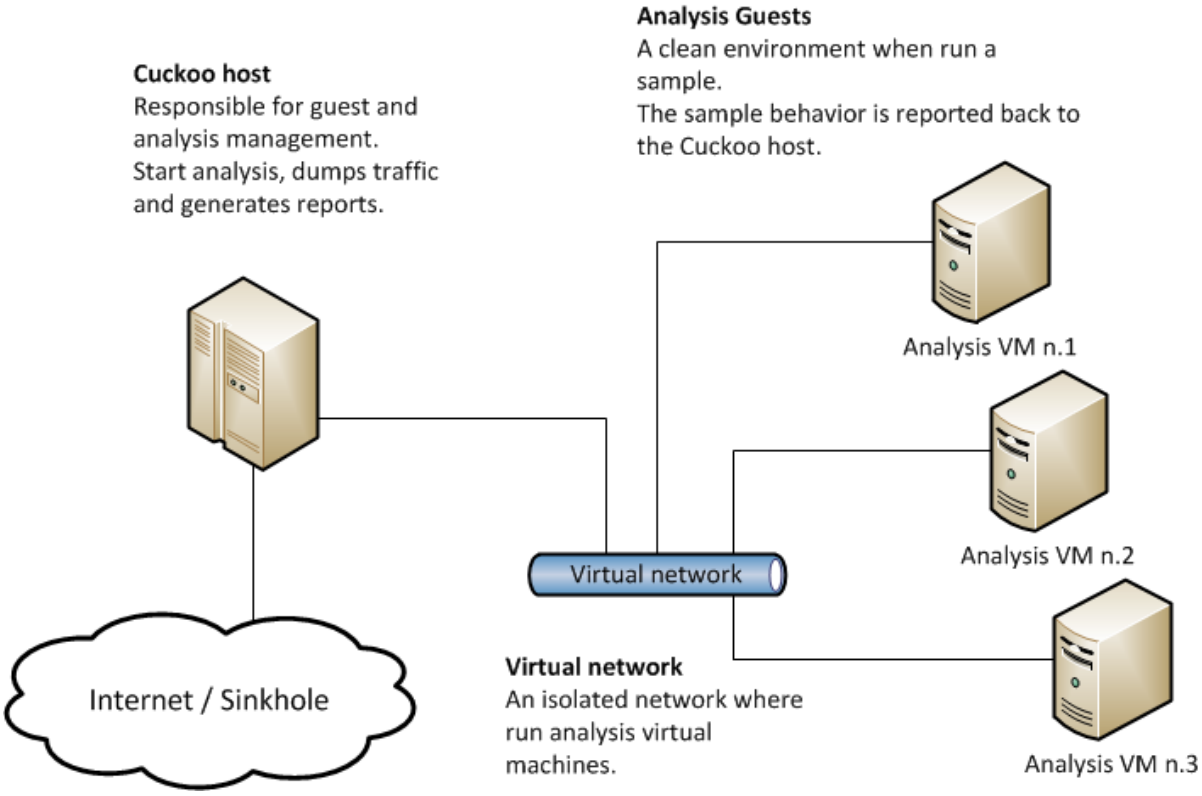

Рисунок 1.5 – Архітектура Cuckoo Sandbox[27]

Кожен аналіз запускається на новій ізольованій віртуальній або фізичній машині. Основними компонентами інфраструктури Cuckoo є хост-машина (програмне забезпечення для керування) і декілька гостьових машин (віртуальних або фізичних машин для аналізу).

Хост запускає основний компонент пісочниці, який керує всім процесом аналізу, в той час як гостьові машини є ізольованими середовищами, де фактично виконуються та аналізуються зразки зловмисного програмного забезпечення.

Завдяки модульній конструкції Cuckoo можна використовувати як у вигляді окремої програми, так і для інтеграції у великі інфраструктури.

Систему можна використовувати для аналізу наступних об’єктів:

- виконувані файли Windows;
- DLL файли;
- PDF документи;
- документи Microsoft Office;
- URL-адреси та файли HTML;
- PHP скрипти;
- файли CPL;
- скрипти Visual Basic;
- ZIP файли;
- Java JAR;
- файли Python;
- майже все інше.

Як заявляють розробники, завдяки модульності та потужним можливостям використання додаткових скриптів немає обмежень у тому, що можна реалізувати завдяки Cuckoo [27].

1.4.2 Joe Sandbox

Joe Sandbox – це система динамічного аналізу зловмисного програмного забезпечення, яка виявляє шкідливе ПЗ на основі його поведінки. Проєкт належить швейцарській компанії Joe Security. Ізольоване програмне середовища здатне виявляти й глибоко аналізувати загрози в найбільш використовуваних операційних системах для ПК, серверів і мобільних пристроїв: Windows, macOS, Linux і Android. Існує хмарний варіант системи Joe Sandbox Cloud та серверний варіант Joe Sandbox Ultimate. Розробник пропонує два варіанти підписки: базовий (безкоштовний, пропонує обмежений функціонал) і розширений (платна версія без обмежень функціональності).

Архітектура Joe Sandbox Ultimate є модульною (рисунок 1.6). Система складається принаймні з одного контролера, що працює під керуванням ОС Linux, і кількох підключених машин для аналізу (зі встановленими Windows і Android), розміщених на платформах віртуалізації, таких як VMware або VirtualBox.

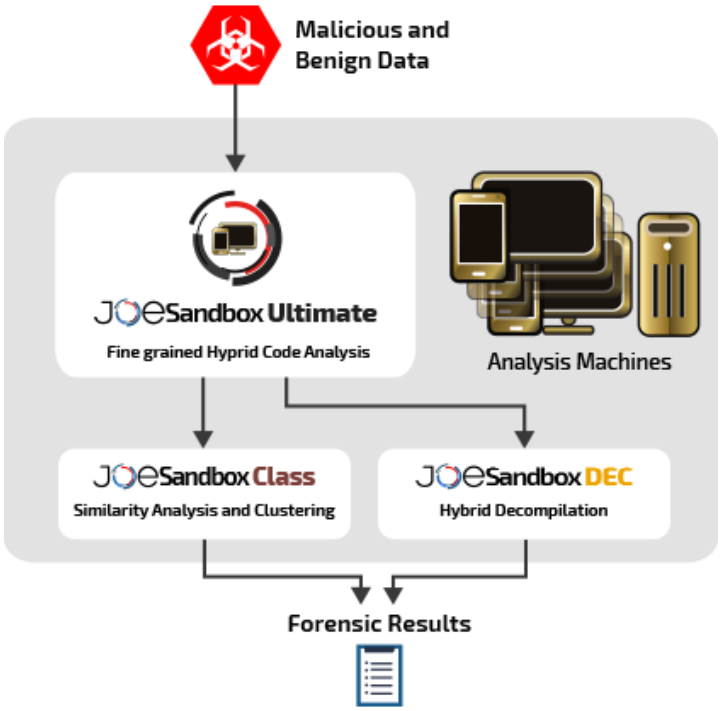

Рисунок 1.6 – Архітектура пісочниці Joe Sandbox [29]

Файли та URL-адреси завантажуються для аналізу в Joe Sandbox Ultimate вручну, через RESTful Web API або через Joe Sandbox Mail Monitor. Joe Sandbox Ultimate спочатку аналізує файл статично. Після цього виконується поглиблений аналіз за допомогою Joe Sandbox Desktop. Після динамічного аналізу Joe Sandbox Class створює звіт про схожість зловмисного програмного забезпечення. Для розуміння зловмисної поведінки Joe Sandbox Ultimate використовує Joe Sandbox DEC, який виконує декомпіляцію коду зразка шкідливого ПЗ.

Joe Sandbox базується на ідеї, що об’єднання найкращих частин кількох різних методів аналізу дасть найкращі результати. З цією метою використовується моніторинг на основі гіпервізора агентом в режимі ядра, статичний і динамічний аналіз коду, декомпіляція, аналіз графів виконання на основі потоку керування, загальні статичні інструменти, які модифікують програми перед виконанням, щоб

краще відстежувати їх внутрішні функції і уникнути виявлення та багато інших функцій. Усі ці різні типи аналізу об’єднуються в розширений звіт щоб дати аналітикам повне розуміння дій аналізованих зразків [29].

1.4.3 VMRay Analyzer

VMRay Analyzer – комплексне рішення на основі пісочниці для автоматизованого виявлення та аналізу складних загроз, яке вирізняється з-поміж інших тим, що використовує безагентний підхід до пісочниці. Якщо в інших пісочницях є агент, який працює в цільовій системі для запису активності зловмисного програмного забезпечення та зв’язку з сервером пісочниці, VMRay реалізується на рівні гіпервізора поза системою, що контролюється. Оскільки всередині віртуальної машини немає агентів або перехоплювачів, саму пісочницю не можна виявити зловмисним програмним забезпеченням. Крім того, VMRay містить низку функцій, призначених для протидії поширеним методам виявлення та ухилення від віртуальних машин.

VMRay можна розгорнути локально або в хмарі. Продукт постачається за річною підпискою. Ліцензування базується на кількості динамічних аналізів, що виконуються на день [30].

На основі отриманих даних можна побудувати порівняльну таблицю систем для аналізу шкідливого програмного забезпечення (таблиця 1.3).

Таблиця 1.3 – Порівняльна характеристика систем аналізу шкідливого ПЗ

	Cuckoo Sandbox	Joe Sandbox	VMRay Analyzer
Локальне розгортання	+	+	+
Розгортання у хмарі	-	+	+

Продовження таблиці 1.3

Підписка	Безкоштовна	Платна/безкоштовна з обмеженнями	Платна
Веб-інтерфейс	+	+	+
Web API	+	+	+
Звіт у різних форматах	+	+	+
Дамп пам’яті	+	+	+
Знімки екрана	+	+	+
Аналіз фізичних серверів	+	+	-
Аналіз Windows	+	+	+
Аналіз Mac OS	+	+	+
Аналіз Linux	+	+	-
Аналіз Android	+	+	-
Аналіз iOS	-	+	-
Підтримка YARA	+	+	+
Документація	+	-	-
Захист від виявлення пісочниці	+-	+	+

Висновки до розділу

У розділі розглянуто основні поняття, що стосуються шкідливого програмного забезпечення, наведено особливості основних видів зловмисних програм, пояснено загрози, які вони несуть. Далі розглянуто список основних методів, які дозволять виявити шкідливе ПО. Важливо було виокремити

особливості основних методів аналізу зразків, адже розуміння поведінки і структури виявлених загроз є важливим аспектом для побудови ефективної системи для боротьби з цими загрозами.

Головною структурною частиною проектованої системи є пісочниця для аналізу структури і поведінки зразків шкідливого ПЗ. Проведений аналіз існуючих рішень в сфері дає можливість зробити висновок, що для побудови нашої системи слід обрати пісочницю Cuckoo Sandbox. Дана система має наступні переваги:

- відкритий код і модульність дозволяють реалізувати додаткові можливості щодо аналізу зразків;
- підтримка всіх сучасних ОС і великої кількості файлових форматів;
- наявність добре документованого REST API;
- можливість одночасного аналізу зразків у різних віртуальних оточеннях;
- вільне поширення продукту дає змогу скоротити витрати на розгортання системи, що важливо для малого і середнього бізнесу.

2 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

Вимоги до системи – це опис того, що система повинна робити, які функції вона повинна виконувати, а також які характеристики та вимоги повинна відповідати. Цей процес допомагає розробникам і проектним командам зрозуміти потреби користувачів, бізнес-вимоги та інші фактори, які повинні бути враховані при проектуванні системи.

Формування чітких вимог до системи дозволяють розбити процес її створення на логічні етапи, встановити пріоритетність того, чи іншого функціоналу та розподілити зони відповідальності.

Аналіз вимог можна розділити на три етапи.

Етап 1. Аналіз предметної області. Даний етап передбачає вивчення предметної області, в рамках якої буде експлуатуватися система, здійснюється огляд існуючих рішень, аналіз конкурентів.

Етап 2. Аналіз вимог. Цей етап передбачає аналіз та вивчення потреб та очікувань користувачів системи. На цьому етапі необхідно зрозуміти, як система повинна працювати та які функції повинна виконувати.

Етап 3. Фіксація вимог. Це формалізований опис функціональних, нефункціональних і технічних вимог, вимог до характеристик якості, до структури системи і принципів взаємодії з компонентами.

2.1 Функціональні вимоги

Функціональні вимоги – це перелік вимог до системи, які описують що система повинна робити, які функції повинна виконувати та які операції повинна підтримувати.

Основними функціональними вимогами системи, що проектується є:

- система повинна мати можливість збирати зразки шкідливих або потенційно небезпечних програм з пристроїв користувачів у разі їх згоди;

- крім зразків має бути можливість збору різноманітної системної інформації для розуміння оточення, в якому було запущене шкідливе ПЗ;

- повинна бути можливість інтеграції додаткових джерел для отримання зразків шкідливого ПЗ;

- панель керування сигнатурами має підтримувати роботу з різними користувачами;

- має бути можливість аутентифікації користувачів для доступу до панелі адміністратора;

- аналіз зразків має здійснювати в автоматичному режимі в окремому ізольованому середовищі;

- має збиратися статистика щодо роботи компонентів системи;

- звіт з отриманими результатами аналізу має надходити в систему управління проєктами.

2.2 Нефункціональні вимоги

Нефункціональні вимоги – це вимоги до програмного комплексу, які не пов’язані з його безпосереднім функціональним змістом. Вони описують такі якісні характеристики системи, такі як надійність, продуктивність, безпека, конфіденційність, зручність користування, сумісність, підтримуваність, масштабованість тощо.

Для проектованої системи можна сформувати наступні нефункціональні вимоги:

- продуктивність аналізу має досягати 500 зразків на добу;

- система має бути доступною 99.9% часу;

- резервне копіювання даних системи;

- система повинна бути масштабованою, щоб забезпечувати потрібну продуктивність при збільшенні кількості зразків;

- веб-інтерфейс системи має бути сумісним з усіма сучасними браузерами;
- логи роботи системи мають зберігатися як мінімум 3 роки;
- зразки шкідливого ПЗ мають видалятися зі сховища через 1 рік.

Одним з видів нефункціональних вимог є апаратні та програмні вимоги. Для нашої системи висунуто такі апаратно-програмні вимоги:

- система має бути розвернута в хмарному середовищі;
- в разі збільшення навантаження система повинна масштабуватися;
- програмне забезпечення має бути розвернуто на ОС сімейства Linux останніх стабільних версій;
- зразки шкідливого ПЗ мають зберігатися в захищеному сховищі з підтримкою шифрування.

Також сформовані вимоги щодо підтримки аналізу різних типів шкідливого ПЗ. Система повинна підтримувати збір і аналіз шкідливого ПО для таких операційних систем і їх версій:

- macOS Ventura;
- macOS Monterey;
- macOS Big Sur;
- macOS Catalina;
- macOS Mojave.

Формати зразків, які мають підтримуватися:

- пакети ПЗ .app;
- архіви zip, 7z, gzip та інші;
- пакети .pkg;
- образи дисків dmg;
- pdf, doc, docx, xls, csv, xml та інші текстові формати;
- perl, bash-скрипти;
- виконувані файли Mach-O;
- розширення для браузерів Chrome, Safari, Firefox.

Висновки до розділу

В даному розділі було сформульовано загальні функціональні та нефункціонавльні вимоги до системи, що проектується. Висунуті вимоги були сформовані у відповідності до бізнес-потреб та з врахуванням ситуації на ринку. Аналіз переваг і недоліків схожих систем, описаних у попередньому розділі, допоміг у формуванні вимог до нашої системи.

В першу чергу були розглянуті функціональних вимоги, а вже на основі них сформовані нефункціональні. Серед нефункціональних в першу чергу можна виділити апаратно-програмні вимоги до системи. Також сформовані вимоги до того, збір зразків з яких ОС має підтримувати система та які файли повинна вміти аналізувати.

3 РОЗРОБЛЕННЯ СЦЕНАРІЇВ ВИКОРИСТАННЯ

Сценарій використання – це опис послідовності кроків або взаємодії між користувачем та системою. Він може включати опис сценаріїв взаємодії користувача з інтерфейсом, взаємодії з даними, функціональність та обмеження системи.

Невід’ємними процесами для побудови діаграм є:

- виділення діючих суб’єктів;
- розробка варіантів використання з боку акторів, які визначають функціональність ПС;
- табличний опис визначених варіантів використання.

Було визначено, що при розробці сценаріїв використання системи, необхідно врахувати наявність двох акторів у системі: вірусний аналітик та адміністратор.

Вірусний аналітик – основний користувач системи, здійснює аналіз знайдених зразків шкідливого або потенційно небажаного ПЗ, формує і вносить нові сигнатури, аналізує статистику роботи ключових підсистем.

Адміністратор – актор, основна роль якого полягає в налаштуванні і обслуговуванні системи. Основна функція – виконання сервісних задач, як налаштування окремих компонентів системи, налаштування доступів, моніторинг системи, усунення неполадок, що виникли під час експлуатації системи.

Розглянемо сценарії використання для кожного з акторів окремо.

3.1 Розроблення діаграми варіантів використання вірусним аналітиком

Вірусний аналітик є основним користувачем і замовником даної системи, для якого важливо подавати дані для аналізу в зручному вигляді в автоматизованому режимі. Спираючи на визначені у попередньому розділі вимоги, створено та детально описані сценарії взаємодії з компонентами системи.

Загальні можливості вірусного аналітика при роботі з системою зображено на діаграмі прецедентів (рисунок 3.1).

Рисунок 3.1 – Діаграма прецедентів для ролі "Вірусний аналітик"

Детальний опис прецедентів описано в таблицях 3.1 - 3.4, а повну діаграму представлено на плакаті 1 додатку А.

Прецедент роботи з системою аналізу статистики детально описано в таблиці 3.1.

Таблиця 3.1 – Опис прецеденту перегляду статистики

Назва	Аналіз та обробка статистики роботи системи
Мета або контекст використання	Перегляд і візуалізація статистики щодо зібраних зразків, статистика щодо їх аналізу
Область видимості та рівень	Адміністратор, вірусний аналітик
Триггер	Перехід на веб-ресурс для аналізу статистики

Продовження таблиці 3.1

Передумова	Користувач успішно аутентифікований	
Умова успішного виконання	Перегляд логів системи, формування звітів, графіків, аналіз роботи підсистеми збору зразків, аналіз підсистеми дослідження зразків	
Умова невиконання	Авторизація неуспішна, відсутність даних	
Актори	Вірусний аналітик	
Опис	Кроки	Дія
	1	Аутентифікація користувача в системі
	2	Вибір необхідного джерела для перегляду статистики
	3	Формування звітів
	4	Вийти з системи

Прецедент “Перегляд звітів” передбачає роботу зі звітами, автоматично створеними в системі управління проєктами. Звіти автоматично створюються при надходженні нових зразків потенційно небезпечного ПЗ. Існує два види звітів (заявок): звіт про зразки та звіт про робочі станції, на яких ці зразки знайдені. Заявки приєднані одна до одної. Детальніше про цей прецедент описано в таблиці 3.2.

Таблиця 3.2 - Опис прецеденту перегляду звітів

Назва	Перегляд звітів (заявок) про знайдені нові потенційно небезпечні зразки
Мета або контекст використання	Формування нових сигнатур, покращення існуючих сигнатур в разі повторної детекції
Область видимості та рівень	Вірусний аналітик

Продовження таблиці 3.2

Триггер	Створення заявки в тікет-систему про знайдення нового потенційно небезпечного зразка	
Передумова	Немає	
Умова успішного виконання	Зразок виявився шкідливим, перегляд результатів аналізу, перехід до формування сигнатури	
Умова невиконання	Зразок виявився нешкідливим	
Актори	Вірусний аналітик	
Опис	Кроки	Дія
	1	Аутентифікація користувача в тікет-системі
	2	Перегляд нової заявки
	3	Визначення ступеню небезпеки
	4	Перехід до формування нової сигнатури

Якщо результати аналізу зразка або групи схожих зразків підтверджують, що дане ПЗ є шкідливим, аналітик приступає до формування сигнатури. Більше детально цей прецедент описано в таблиці 3.3.

Таблиця 3.3 - Опис прецеденту створення сигнатур

Назва	Формування сигнатур шкідливого ПЗ
Мета або контекст використання	Створення сигнатур нових знайдених зразків шкідливого програмного забезпечення для додавання їх в базу і майбутньої детекції антивірусним ПЗ
Область видимості та рівень	Вірусний аналітик

Продовження таблиці 3.3

Триггер	Автоматичний аналіз виявив, що даний зразок є небезпечним або потенційно небажаним ПЗ	
Передумова	Немає	
Умова успішного виконання	Формування нової сигнатури у форматі хешу або набору YARA-правил	
Умова невиконання	Зразок виявився нешкідливим	
Актори	Вірусний аналітик	
Опис	Кроки	Дія
	1	Перегляд заявки
	2	Перегляд звіту про динамічний і статичний аналіз
	3	Формування сигнатури
	4	Збереження нової сигнатури в базу

Створені сигнатури потрібно зберегти в базі даних. Цей процес описано в таблиці 3.4.

Таблиця 3.4 - Опис прецеденту роботи з підсистемою управління сигнатурами.

Назва	Робота з підсистемою управління сигнатурами
Мета або контекст використання	Перегляд, редагування і внесення в базу нових сигнатур шкідливого програмного забезпечення
Область видимості та рівень	Вірусний аналітик
Триггер	Сформовано нову сигнатуру шкідливого ПЗ

Продовження таблиці 3.4

Передумова	Користувач успішно аутентифікований	
Умова успішного виконання	Перегляд існуючих сигнатур, збереження нових сигнатур в базу даних, редагування існуючих сигнатур	
Умова невиконання	Авторизація неуспішна, недоступність бази даних	
Актори	Вірусний аналітик	
Опис	Кроки	Дія
	1	Аутентифікація користувача в системі
	2	Вибір необхідного вкладки: перегляд, створення, редагування сигнатур
	3	Внесення необхідних даних
	4	Збереження даних в базу

Підсистема управління сигнатурами являє собою веб-інтерфейс для взаємодії вірусного аналітика з системою збереження даних (сигнатур). На рисунку 3.2 зображено діаграму прецедентів для вищезазначеної системи.

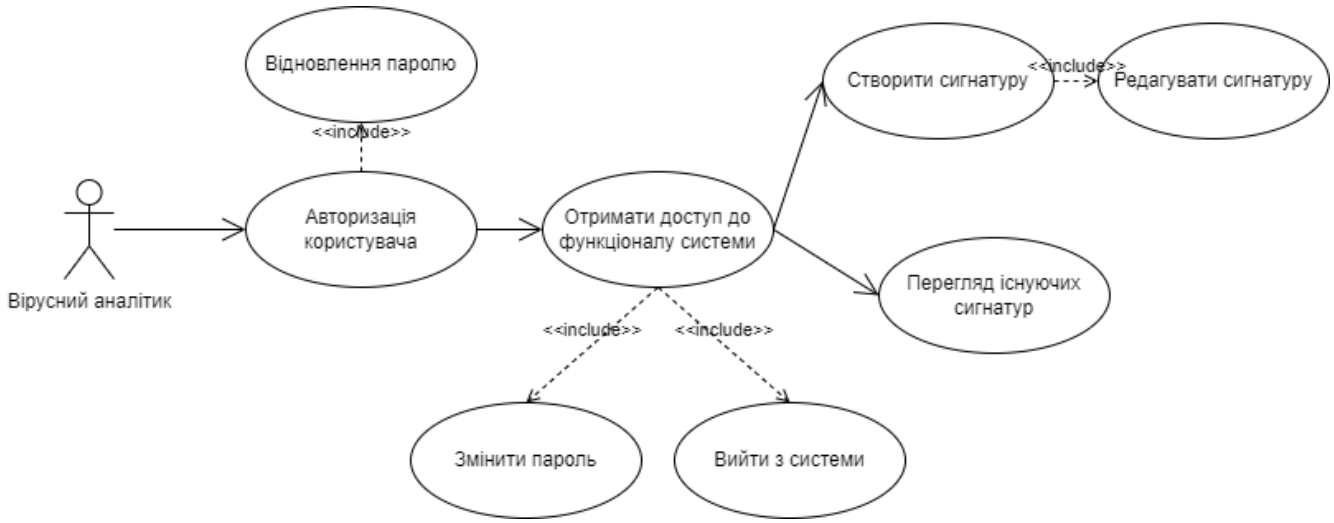

Рисунок 3.2 – Діаграма прецедентів підсистеми управління сигнатурами

Опишемо детальніше основні сценарії при роботі з даною підсистемою в таблицях 3.5-3.7.

Таблиця 3.5 - Опис прецеденту аутентифікації користувача

Назва	Аутентифікація користувача	
Мета або контекст використання	Отримання доступу до веб-інтерфейсу управління сигнатурами	
Область видимості та рівень	Вірусний аналітик, адміністратор	
Триггер	Перехід на сайт підсистеми	
Передумова	Немає	
Умова успішного виконання	Користувач успішно авторизований	
Умова невиконання	Авторизація неуспішна, невірний пароль, користувача не існує	
Актори	Вірусний аналітик, адміністратор	
Опис	Кроки	Дія
	1	Перехід на сторінку системи
	2	Заповнення даних: логін, пароль
	3	Перевірка системою наявності введених даних в локальній базі користувачів
	4	Успішний вхід до системи

Основним функціональним елементом підсистеми управління сигнатурами є блок, що дозволяє створювати нові та редагувати вже створені сигнатури. Розглянемо цей прецедент в таблиці 3.6.

Таблиця 3.6 - Опис прецеденту створення і редагування сигнатур

Назва	Створення і редагування сигнатур	
Мета або контекст використання	Внесення нових сигнатур шкідливого ПЗ в базу даних або редагування вже існуючих сигнатур	
Область видимості та рівень	Вірусний аналітик	
Триггер	Немає	
Передумова	Сформована нова сигнатура, яку потрібно додати в базу	
Умова успішного виконання	Зміни успішно внесені в базу даних	
Умова невиконання	Дані введені в неправильному форматі, не вдалось зберегти зміни до бази	
Актори	Вірусний аналітик	
Опис	Кроки	Дія
	1	Перехід на вкладку створення/ редагування сигнатур
	2	Заповнення полів, що описують сигнатуру
	3	Збереження введених даних

3.2 Розроблення діаграми варіантів використання адміністратора

Адміністратор є допоміжним користувачем системи, в обов’язки якого може входити налаштування окремих компонентів системи, формування звітів, моніторинг системи, створення користувачів та інші функції, пов’язані з обслуговуванням системи.

На діаграмі 3.3 зображено схему варіантів використання системи збоку адміністратора.

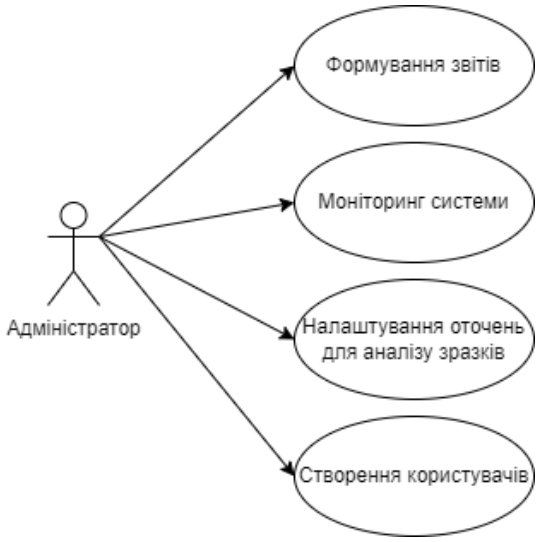

Рисунок 3.3 – Діаграма прецедентів системного адміністратора

Розглянемо більш детально основні активності, які виконує адміністратор системи. При аналізі зразків потенційно небезпечного шкідливого ПЗ важливо, щоб оточення, в якому виконується цей аналіз максимально відповідало оточенню на клієнтських машинах. Налаштуванням специфічних особливостей середовища, встановленням оновлень та створенням нових середовищ займається системний адміністратор. Цей процес описаний в таблиці 3.7.

Таблиця 3.7 - Опис прецеденту налаштування оточень для аналізу зразків

Назва	Налаштування оточень для аналізу зразків
Мета або контекст використання	Для можливості динамічного аналізу шкідливих зразків на різних версіях ОС, з різними налаштуваннями
Область видимості та рівень	Адміністратор
Триггер	Оновлення системи, вихід нової версії ОС

Продовження таблиці 3.7

Передумова	Немає	
Умова успішного виконання	Створення знімку системи з новими оновленнями або налаштуваннями	
Умова невиконання	Немає	
Актори	Адміністратор	
Опис	Кроки	Дія
	1	Отримання вимог щодо нових параметрів оточення
	2	Внесення нових налаштувань, оновлень
	3	Створення оновленого знімку системи
	4	Розгортання нового знімку

Для оперативного виявлення проблем, що виникли в процесі роботи системи важливо збирати метрики з різних підсистем. Детальний опис даного сценарію наведено в таблиці 3.8.

Таблиця 3.8 - Опис прецеденту моніторингу коректної роботи системи

Назва	Моніторингу коректної роботи системи
Мета або контекст використання	Постійна передача результатів роботи системи у зовнішнє середовище за допомогою інтеграційних модулей
Область видимості та рівень	Адміністратор
Триггер	Фіксований часовий інтервал

Продовження таблиці 3.8

Передумова	Немає	
Умова успішного виконання	Надходження метрик щодо роботи системи у сховище	
Умова невиконання	Відсутність метрик, некоректна робота модуля для збору метрик	
Актори	Адміністратор	
Опис	Кроки	Дія
	1	Збір системних метрик
	2	Збереження результатів у сховище
	3	Візуалізація отриманих метрик в зовнішній системі

Висновки до розділу

У даному розділі на основі функціональних вимог було сформовано основні сценарії використання системи, що розробляється. Основними ролями, що задіяні в системі визначено вірусного аналітика та адміністратора.

Для вірусного аналітика та адміністратора визначено контексти їх взаємодії із системою, що описано в окремих діаграмах варіантів використання. Більшість з зазначених у діаграмі сценаріїв було описано більш детально з використанням таблиць. Розгляд сценаріїв використання системи дозволяє зрозуміти, як користувачі можуть взаємодіяти з системою і які функції вони можуть виконувати. Цей процес допомагає покращити розуміння того, як система може бути використана для досягнення бізнес-цілей, а також ідентифікувати можливі проблеми та недоліки у функціональності системи.

4 МОДЕЛЬ СИСТЕМИ ПО ФОРМУВАННЮ БАЗИ СИГНАТУР КОМП'ЮТЕРНИХ ВІРУСІВ

Структурна схема розроблюваної системи зображена на плакаті 3 додатку А. Схема поділена на логічні функціональні блоки (підсистеми), що значно полегшує відлагодження системи і закладає принципи модульності. Виходячи з зазначених вимог до системи формування бази сигнатур шкідливого ПЗ було визначено наступні структурні елементи:

- підсистема збору зразків;
- сховище зразків;
- модуль керування аналізом (оркестратор);
- віртуальне середовище для автоматичного аналізу зразків (пісочниця);
- генератор сигнатур;
- підсистема управління сигнатурами;
- база даних сигнатур;
- система логування.

Розглянемо більш детально призначення кожного із елементів системи, що проектується.

Підсистема збору зразків призначена для пошуку зразків потенційно небезпечного програмного забезпечення з різних джерел. Прикладами таких зразків можуть бути виконувані програми, документи, архіви, скрипти, розширення браузера та інші.

Підсистему збору зразків можна розділити на декілька модулів (рисунок 4.1).

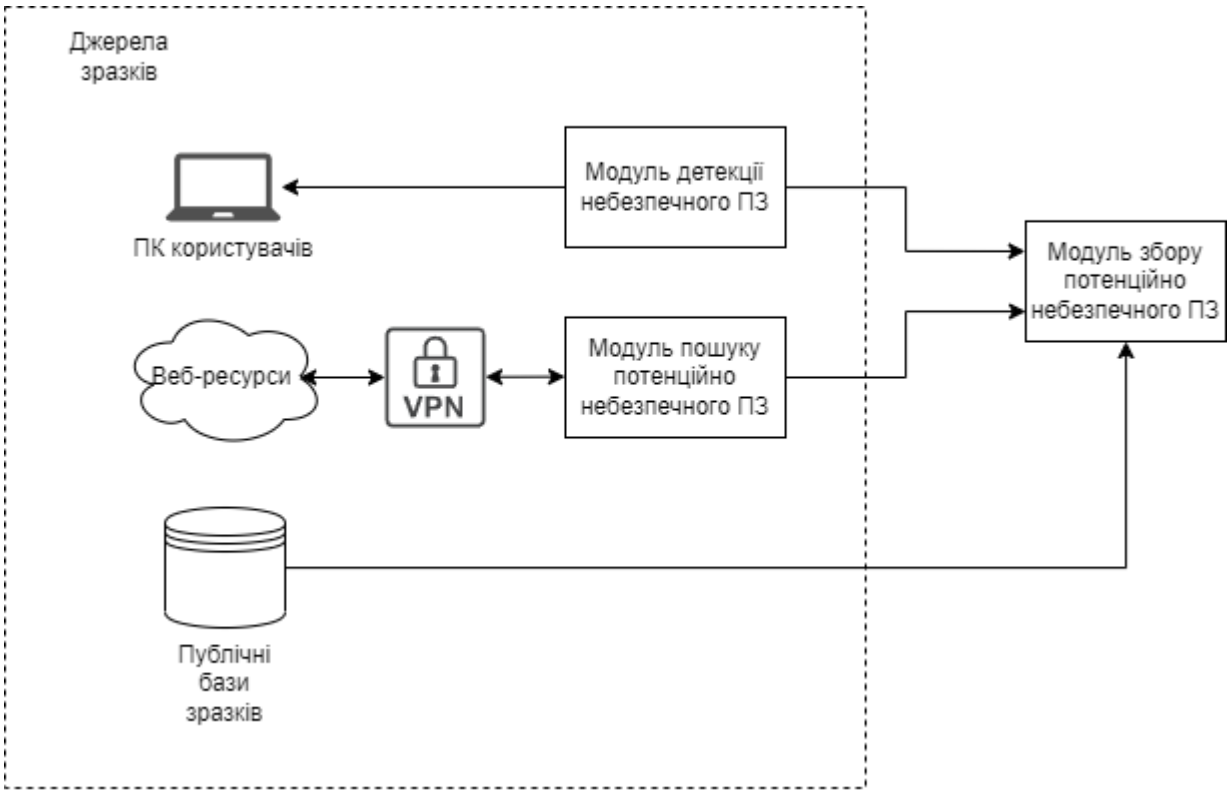

Рисунок 4.1 – Підсистема збору зразків

Перш ніж виконувати аналіз зразків і формувати сигнатури для їх детекції, необхідно ці зразки звідкись отримати. Передбачається наявність кількох джерел для отримання зразків шкідливого ПЗ.

Основним джерелом зразків виступають користувачі, на персональних машинах яких спостерігаються ознаки діяльності шкідливого ПЗ. Вони звертаються до технічної підтримки і при отриманні згоди відбувається аналіз їх пристрою модулем детекції. Модуль детекції на основі заздалегідь визначених правил формує список потенційно небезпечних файлів. Модуль збору призначений для упакування і відправки зібраних зразків в сховище зразків, звідки розпочнеться їх подальший аналіз. Разом із зразками відбувається збір важливої системної інформації для розуміння оточення системи, на якій виявлено шкідливе ПЗ. Такий спосіб дозволяє оперативно реагувати на нові загрози, які можуть бути спричинені шкідливим програмним забезпеченням.

Ще одним важливим джерелом виступають публічно доступні веб-ресурси, з яких користувачі можуть завантажувати потенційно небезпечне програмне

забезпечення. Модуль пошуку здійснює сканування таких сторінок, завантажує нові програмні продукти та відправляє їх на аналіз. Важливо постійно оновлювати список таких ресурсів, щоб мати можливість зібрати якомога більше зразків. Для приховування та захисту станції, з якої виконується таке сканування, використовується захищене мережеве з'єднання (VPN).

Щоденна поява величезної кількості нових загроз змушує виробників антивірусного ПЗ та вірусних лабораторій обмінюватися інформацією щодо знайдених зразків. Існують як платні так і безкоштовні бази, звідки можна оперативно отримати зразки небезпечного ПЗ [31]. Дослідження таких вірусних лабораторій може стати ще одним важливим джерелом для отримання зразків.

Завдяки принципу модульності розроблюваної системи можна легко інтегрувати і нові джерела зразків, що в кінцевому результаті дозволить покращити показники виявленого шкідливого ПЗ.

Модуль збору потенційно небезпечного ПЗ – це програма, що виконується на кінцевій станції користувача і збирає потенційно небезпечні зразки в сховище зразків.

Інструменту збору потенційно небезпечного ПЗ повинен мати змогу:

- отримувати інформацію про конфігурацію з критеріями прийнятності для потенційних зразків шкідливого ПЗ;
- отримувати потенційні зразки шкідливого ПЗ з необхідними метаданими відповідно до отриманих конфігурацій;
- збирати необхідну інформацію про робочу станцію;
- відправляти зібрані зразки в захищене хмарне сховище;
- завантажувати в хмарне сховище інформацію про робочу станцію;
- можливість працювати в різних середовищах: stage або production.

Даний модуль повинен збирати з робочих станцій зразки, що відповідають наступним критеріям:

- працювати з каталогами, описаними в файлі конфігурації; зразок повинен бути вкладений не глибше, ніж рівень глибини, встановлений для кожного

каталогу (якщо рівень глибини не вказано, інструмент перегляне всі вкладені каталоги); для кожного каталогу необхідно описати “Тип файлу” або “Розширення файлу”;

- збігається з одним із типів файлів, описаних у файлі конфігурації. Кожен каталог має власний список типів файлів;
- збігається з одним із розширень файлів, описаних у файлі конфігурації;
- зразок має бути розміром не більше, ніж задано у файлі конфігурації; кожен каталог має власне обмеження розміру;
- зразок не має бути завантаженим в сховище раніше.

Підсистема “*Сховище зразків*” призначена для зберігання зразків шкідливого програмного забезпечення і його метаданих. Це забезпечує вірусному аналітику легкий спосіб взаємодії із зібраними даними. Загальна структура схема зазначеної підсистеми наведена на рисунку 4.2.

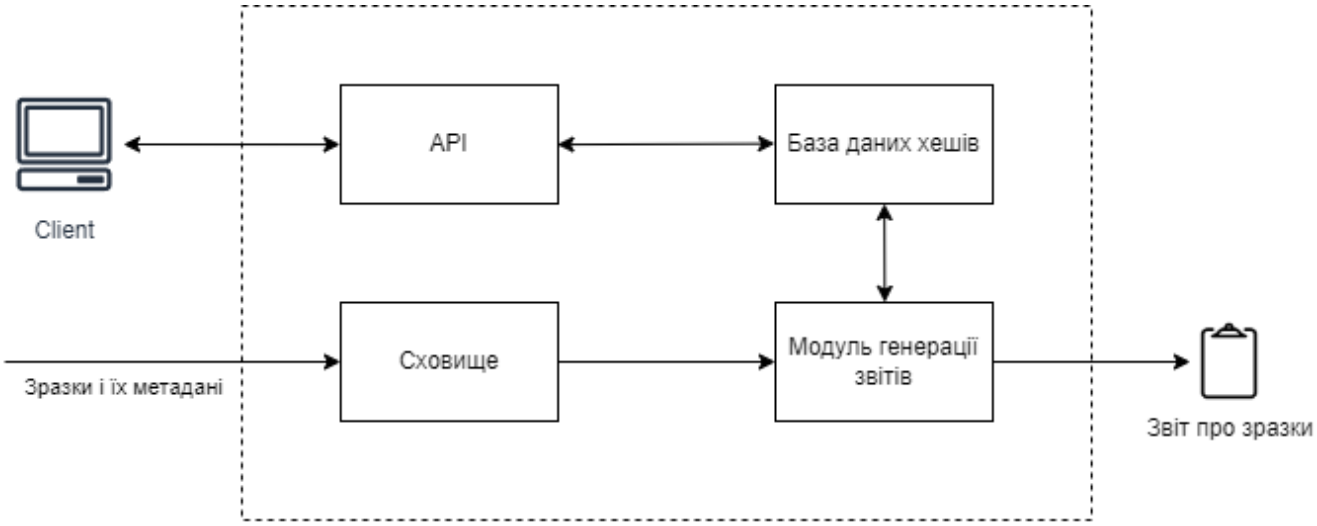

Рисунок 4.2 – Підсистема сховища зразків

Підсистема має надати аналітику зручний спосіб взаємодії із вхідними зразками, тим самим максимально підвищуючи ефективність та продуктивність його роботи.

Знайдені зразки шкідливого ПЗ можуть повторюватися, тому важливо забезпечити механізм перевірки їх унікальності, щоб не створювати зайвої роботи для аналітика і скоротити обсяг даних в системі. Для цього планується створити API, яка на основі контрольної суми зразка буде перевіряти чи не був він доданий в сховище раніше. Дані щодо контрольних сум будуть зберігатися в окремій базі даних.

В разі появи нових зразків в системі модуль генерації звітів буде створювати новий звіт до тікет-системи. Звіт включає в себе:

- посилання на архів з доданим зразком;
- інформацію про систему, на якій виявлено зразок;
- інформацію про наданий зразок шкідливого ПЗ;
- результати аналізу зразка.

Отримання детального звіту щодо зразка значно спрощує класифікацію, надає вичерпну інформацію про всю активність, яку здійснює зразок, тому допомагає швидко сформувати сигнатуру знайденого шкідливого ПЗ.

Оркестратор – це модуль, який керує процесом автоматичного аналізу зразків. Він перевіряє надходження нових зразків в систему, проводить їх статичний аналіз, в разі необхідності передає в пісочницю для динамічного аналізу. В результаті даний модуль перетворює зібрану інформацію у формат, зрозумілий людині, і завантажує її в тікет-систему.

Пісочниця – це інструмент, який використовується для автоматизації аналізу шкідливих зразків. З даним інструментом та існуючими рішеннями ми вже ознайомились в розділі 1. Зокрема в нашій системі пісочниця буде поєднувати функції статичного і динамічного аналізу зразків. Статичний аналіз дозволяє знайти спільні риси для сімейства шкідливих програм та відфільтрувати зразки, які вже задетектовані іншими антивірусними рішеннями. Динамічний аналіз слугує для виявлення прихованих дій, що здійснюються після запуску шкідливого зразка.

Підсистема управління сигнатурами – це веб-інтерфейс для роботи вірусного аналітика з базою даних сигнатур. Функціональні можливості даної підсистеми описано у розділі 3.

Генератор сигнатур – модуль, який дозволяє згенерувати ефективні сигнатури для цілого сімейства схожих шкідливих програм. При цьому виконується кластеризація великої кількості файлів на основі подібності. Далі для кожного сімейства схожих шкідливих програм в автоматичному формується єдине правило для детектування.

База даних сигнатур – нереляційна база даних, в якій зберігаються сформовані сигнатури для детектування шкідливого ПЗ. Взаємодія з цією базою відбувається через підсистему управління сигнатурами. Після внесенення нових сигнатур в базу відбувається їх експорт в спеціальний файл сигнатур, з яким вже працює фреймворк антивірусу.

Висновки до розділу

В даному розділі розглянута модульна структура системи, що проектується. На структурній схема виокремлено окремі функціональні блоки та вектори взаємодії між ними. Здійснено опис і розгляд особливостей кожного з функціональних модулей.

Модульна архітектура системи дозволяє легко інтегрувати нові джерела вхідних даних без внесення глобальних змін до компонентів. Розроблена архітектура системи відкриває можливості масштабування системи при необхідності.

5 ВИБІР ТА ОБҐРУНТУВАННЯ ЕЛЕМЕНТІВ ТА ТЕХНОЛОГІЙ

5.1 Вибір хмарної платформи

Враховуючи нефункціональні вимоги та особливості архітектури, розгортання переважної більшості структурних частин системи планується у хмарному середовищі. Розгортання системи у хмарі має кілька переваг, серед яких можна виділити [32]:

- масштабованість: хмарні платформи можуть автоматично масштабуватись в залежності від потреб користувачів та бізнесу;
- висока доступність: розгортання у хмарі дозволяє забезпечити високу доступність та надійність системи;
- ефективність: користування хмарним середовищем дозволяє скоротити витрати на обладнання, його обслуговування та управління системою;
- безпечність: коли компанії керують власними центрами обробки даних, вони можуть стати вразливими до стихійних лих або кібератак; хмарні сервіси зменшують дані ризики, адже забезпечують резервування власних сховищ.

Розглянемо найбільших гравців на ринку публічних хмарних технологій: Amazon Web Services (AWS) та Google Cloud Platform (GCP).

Amazon Web Services AWS є дочірньою компанією Amazon, яка пропонує послуги хмарних обчислень компаніям і приватним особам, починаючи з 2006 року. Загалом AWS керує понад 125 фізичними центрами обробки даних у різних країнах. Хмара AWS охоплює 99 зон доступності у 31 географічному регіоні по всьому світу [33]. Щороку кількість точок присутності збільшується, тому ці цифри постійно збільшуються.

AWS має значно більшу кількість сервісів та функцій порівняно з будь-яким іншим провайдером хмарних послуг. Хмара охоплює широкий спектр інфраструктурних технологій, таких як інструменти для обчислення, зберігання даних та бази даних, а також впроваджує передові інновації, такі як сервіси машинного навчання та штучного інтелекту, озера даних та аналітики, інтернет

речей тощо. Загалом станом на 2023 рік AWS надає понад 200 повнофункціональних сервісів, основні з яких перераховано на рисунку 5.1.

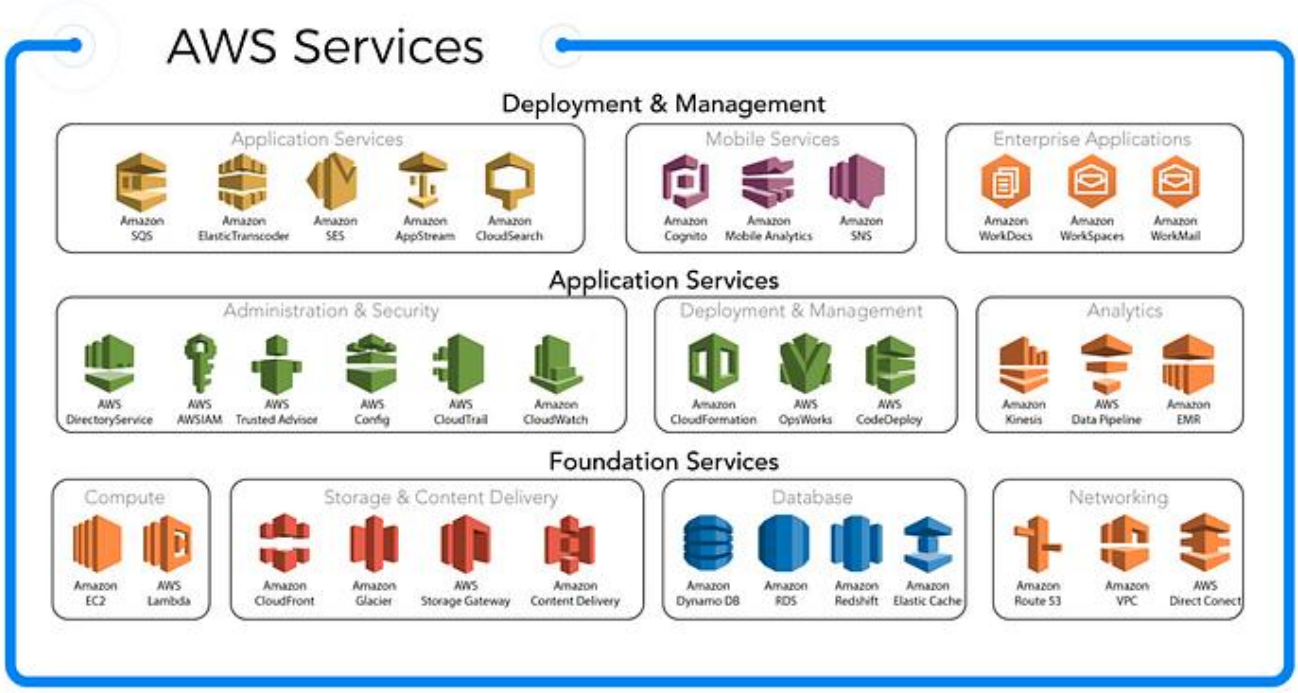

Рисунок 5.1 – Основні сервіси, які надає AWS[34]

Серед основних сервісів, які допоможуть в побудові будь-якої хмарної системи, можна виділити наступні.

Amazon Elastic Compute Cloud (EC2): це масштабоване віртуальне обчислювальне середовище, яке дозволяє користувачам запускати та керувати віртуальними машинами, які називають екземплярами. Для вибору доступні понад 500 типів екземплярів з широким вибором процесорів, сховищ, мереж, операційних систем відповідно до будь-яких потреб бізнесу.

Amazon Simple Storage Service (S3): це високомасштабоване, доступне, безпечне та довготривале сховище об’єктів, яка дозволяє користувачам зберігати та отримувати дані з будь-якої точки Інтернету.

AWS Lambda: це безсерверний обчислювальний сервіс, який дозволяє користувачам запускати код віртуальних програм або служб без необхідності налаштування серверів і програмного оточення.

Amazon Aurora: це сумісний із MySQL і PostgreSQL механізм реляційної бази даних, який поєднує в собі продуктивність і доступність традиційних

корпоративних баз даних із простотою й економічною ефективністю баз даних з відкритим кодом.

Amazon DynamoDB: це повністю керована служба бази даних NoSQL, яка забезпечує швидку та передбачувану продуктивність із плавною масштабованістю.

Amazon Elastic Container Service (ECS): це повністю керована служба для оркестрації контейнерів, яка дозволяє користувачам легко запускати, керувати та масштабувати Docker контейнери.

Amazon CloudFront – це веб-сервіс для доставки контенту, яка дозволяє доставляти статичний і динамічний контент по всьому світу, пропонуючи високу продуктивність і безпеку, а також високу швидкість передачі даних і низьку затримку.

Іншим популярною хмарною платформою є Google Cloud Platform (GCP). GCP, запропонована Google у 2011 році – це набір хмарних обчислювальних служб, які працюють на тій самій інфраструктурі, яку Google використовує внутрішньо для своїх клієнтських продуктів [35].

Google Cloud підтримує 37 регіонів по всьому світу. Ці регіони поділені на зони доступності. Кожна зона має власний набір послуг і обмежень.

Серед найпопулярніших сервісів Google Cloud Platform можна виділити наступні.

Compute Engine – це віртуальна машина, яка працює в інфраструктурі Google. Сервіс пропонує високомасштабовану обчислювальну потужність, яку можна налаштувати відповідно до потреб будь-якого робочого навантаження.

Kubernetes Engine – система управління кластером і оркестрації для координації контейнерів Docker. Сервіс заснований на відкритому проекті Kubernetes.

Cloud Storage – це сервіс для зберігання неструктурованих даних. Google Cloud Storage можна використовувати для різноманітних сценаріїв, зокрема для

роздачі вмісту веб-сайту, зберігання бекапів або розповсюдження великих об’єктів даних серед користувачів шляхом прямого завантаження.

Cloud SQL – повністю керована служба реляційної бази даних для MySQL, PostgreSQL і SQL Server.

На рисунку 5.2 відображено більш розгорнутий список доступних сервісів Google Cloud з розподілом за функціональним призначенням.

Рисунок 5.2 – Основні сервіси платформи Google Cloud [36]

Зважаючи на вже існуючі порівняльні аналізів хмарних сервісів [37] та з врахуванням власного досвіду роботи з хмарними інфраструктурами для побудови нашої системи було обрано хмарну платформу AWS. Можна виділити наступні переваги даної хмари:

- широкий вибір доступних сервісів для будь-яких потреб;
- гнучка цінова політика;
- кожен сервіс добре документований;
- зручний API та веб-інтерфейс для роботи із сервісами;
- добре розвинута мережа для доставки контенту (CDN);
- нативна інтеграція багатьох сервісів між собою.

5.2 Обґрунтування вибору системи аналізу шкідливого ПЗ

З урахуванням загальних вимог до проектованої системи було встановлено наступні критерії вибору системи для аналізу шкідливого ПЗ (пісочниці):

- підтримка ОС різних сімейств і версій;
- масштабованість;
- гнучкість налаштувань;
- можливість розширення функціоналу з використанням додаткових модулів;
- наявність добре документованого REST API.

Програмний комплекс Cuckoo Sandbox цілком відповідає даним вимогам.

Розглянемо більш детально архітектуру Cuckoo Sandbox (рисунок 5.3) [38], яка повинна дати чітке уявлення про основні компоненти, з яких складається Cuckoo, і як вони взаємодіють між собою.

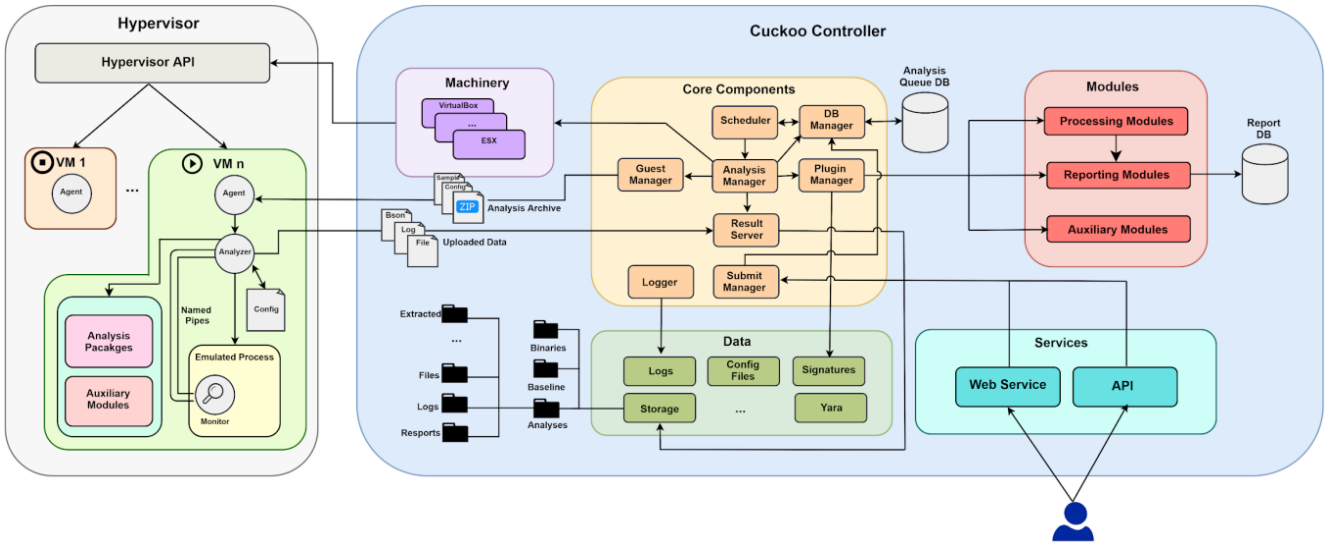

Рисунок 5.3 – Архітектура Cuckoo[38]

Як видно з наведеної вище схеми, систему можна розділити на дві частини: контролер (або господар) і гостьові машини. Контролер приймає файли як

завдання для емуляції, гості є групою машин, на яких відбувається фактичне виконання файлів.

Кожна емуляція починається з того, що користувач надсилає файл до системи. Це можна зробити кількома способами: через веб-інтерфейс, API або з використанням утиліти cuckoo submit. В будь-якому випадку задача отримає свій id і буде переформатована серверним кодом у вигляді JSON-тексту, який у свою чергу передається до *Submit Manager*. Він є першим компонентом для обробки запитів у ядрі Cuckoo та відповідає за читання, інтерпретацію та перетворення JSON на об’єкт, який потім можна відправити в базу даних SQL.

База даних є сутністю, яка зберігає в собі чергу аналізу і містить в собі таку інформацію про завдання, як статус, час початку і закінчення, гостьова машина, яка його обробила тощо. Окрім метаданих щодо завдання, база має поля, пов’язані з наданим зразком, наприклад, розмір, тип файлу, хеші тощо. Дана система підтримує бази даних MySQL, PostgreSQL або SQLite. Компонент, який здійснює взаємодію з базою та з менеджером задач називається *Database Manager*.

Завдання аналізу готове до отримання і виконання щойно воно потрапляє в чергу. Для цього Cuckoo використовує компонент *Scheduler*, який постійно опитує чергу на наявність нових задач та відправляє їх на обробку у мірі появи. Перед відправкою на виконання Scheduler виконує деякі перевірки аби переконатися, що базові умови для емуляції виконуються (наприклад, є доступні гостьові машини, є достатньо місця для зберігання звітів та інші).

Далі управління переходить до менеджера аналізу (*Analysis Manager*). Цей компонент відповідає за основний обсяг роботи щодо управління системою. Він виконує початкову підготовку гостьової машини, підготовку файлу конфігурації аналізу, нотифікує сервер результатів про ініційоване завдання та виконує оновлення бази даних черги аналізу щодо статусу завдання. Протягом процесу емуляції менеджер спілкується з різними компонентами, щоб слідкувати за процесом аналізу.

Перед проведенням емуляції необхідно запустити гостьову машину з існуючого знімка. Щоб це зробити, Analysis Manager повинен зробити виклик через компонент, який називається *Machinery*. Цей термін відноситься до набору модулів, які відповідають за взаємодію з програмним забезпеченням віртуалізації через API. Модуль здатний виконувати такі дії, необхідні в процесі емуляції, як запуск, зупинка та повернення стану гостьової машини. Слід відзначити, що Cuckoo одночасно може працювати тільки з одним менеджером віртуалізації.

До цього часу в основному виконувалася підготовка до аналізу. Тепер, коли система фактично готова його почати, керування передається гостьовому менеджеру (*Guest Manager*). Це клас, який керує зв'язком з гостьовим компонентом віртуальний машини, який називається агентом. Агент – це веб-сервер Python, який запущений на знімку гостьової машини та чекає на запит від гостьового менеджера.

Гостьовий менеджер, у свою чергу, готує zip-файл, який міститиме кілька ресурсів для аналізу (аналізатор, монітор) і пересилає їх агенту разом із файлом конфігурації та зразком, який має виконуватись. Аналізатор повинен визначити, який пакет необхідно застосувати для аналізу виконання файлу. Пакет аналізу – це клас, який описує, як аналізатор має виконувати емульований файл залежно від його типу (наприклад, для виконання PE або документа Office знадобляться зовсім різні методи).

На цьому етапі аналізатор вже може починати виконання зразка, але ще потрібно мати інструмент для відстеження поведінки запущених процесів. Для цього у виконуваний процес виконується ін'єкція DLL-компонента, який називається *monitor*. Далі під час виконання зразка монітор здійснює перехоплення функцій емульовного процесу. Дані з перехоплених функцій разом з іншими артефактами, такими як видалені буфери, файли, дампи мережевого трафіку передаються назад до аналізатора, а потім перенаправляються до контролер машини до компонента під назвою *Result Server*.

Тепер дані аналізу знаходяться на стороні контролера, але вони все ще необроблені та не готові до передачі користувачу. Тому для обробки результатів застосовується компонент *Analysis Manager*, який використовує різні модулі для обробки отриманих даних.

В кінці роботи, менеджер аналізу передасть керування модулям звітності, які можуть формувати готовий для використання звіт [38].

5.3 Інструмент YARA для ідентифікації шкідливого програмного забезпечення

Як вже зазначалося раніше, зразки шкідливого ПЗ можна ідентифікувати та класифікувати шляхом розпізнавання рядків або двійкових послідовностей в їх коді. Засіб YARA може допомогти створити правила для виявлення унікальних послідовностей у шкідливих програмах та інших об’єктах, що підлягають аналізу.

YARA – це популярний та універсальний інструмент, який іноді називають швейцарським ножем для дослідників шкідливих програм. Він призначений для ідентифікації і класифікації зразків шкідливого програмного забезпечення [39]. Для створення правил виявлення використовується синтаксис, схожий на регулярні вирази. Окрім регулярних виразів, правила YARA можуть містить звичайні текстові або шістнадцяткові послідовності. Ці правила складаються з набору рядків і логічного виразу, яке визначає логіку пошуку. Сформовані правила можна використовувати для сканування файлів із застосуванням утиліти yara або інтегруючи бібліотеку yara-python у власний програмний код.

Розглянемо структуру і синтаксис YARA-правил на прикладі (рисунок 5.4).

Правило містить наступні компоненти [40].

Рядок 1: кожне правило починається з ключового слова rule, за яким слідує ідентифікатор правила (“SampleRule” в нашому випадку). Ідентифікатор чутливий до регістру, може містити будь-який буквено-цифровий символ і знак

підкреслення, але перший символ не може бути цифрою. Довжина не більше 128 символів.

Рядок 2: містить метадані правила YARA. У цьому розділі можна вказати автора, дату створення правила, опис того, що робить правило тощо.

Рядки 5-8: секція strings містить рядки (текст, шістнадцяткові або регулярні вирази), які ми хочемо відшукати у файлі. Кожен рядок має власний ідентифікатор, що починається з символу $, за яким слід послідовність літер, цифр або підкреслення. Рядки також можуть містити додаткові модифікатори для заданих змінних. На ці змінні потім можна посилатися в секції умов.

Рядки 10-12: секція умов, що описує логіку правила. Ця секція повинна містити логічний вираз, який повертає істину, якщо рядок знайдено у файлі чи пам'яті процесу, або хибу у протилежному випадку. В умові також можна задати розташування рядка у файлі та виключити файли заданого розміру.

```
1  rule SampleRule{
2    meta:
3     author = "Andrii Rudenko"
4     description = "Example YARA rule"
5    strings:
6     $mz = "MZ"
7     $str1 = "Crash104.dll" fullword nocase wide
8     $hex1 = {7a 63 ?? 69 6f 6e 20 65 61}
9
10   condition:
11    $mz at 0 and ($s1 and $hex1)
12    and (filesize < 100KB)
13 }
```

Рисунок 5.4 – Приклад правила YARA для ідентифікації шкідливих зразків

Отже, правило, наведене в прикладі, описує виконувану програму (про це сигналізує заголовок MZ на початку файлу), що містить в своєму коді посилання на бібліотеку *Crash104.dll*, та містить послідовність, задану змінною *$hex1*, в якій 3 байт може бути будь-яким.

5.4 Алгоритм нечіткого хешування для пошуку подібних зразків

Розробники шкідливих програм використовують масу різних методів для приховування своїх творінь від детекції та статично-динамічного аналізу. З частиною таких методів ми ознайомились в розділі 1 дисертації. Однак і розробники антивірусного ПЗ не стоять на місці, використовуючи вдосконалені алгоритми пошуку сімейств схожих зразків.

На практиці в більшості випадків для створення нових різновидів шкідливих програм використовується вже існуюча кодова база або ядро вірусу. Розробники вірусів не витрачають дорогоцінний час на написання нових “якісних” вірусів, а просто використовують вже існуючи зразки.

Існуючий код може бути перезібраний за допомогою іншого компілятора, з нього можуть бути видалені або, навпаки, в нього можуть бути додані нові функції. Оновлюються деякі бібліотеки, змінюється порядок коду всередині файлу (при цьому застосовуються нові компоновщики, пакети, обфускація тощо). Сенс таких перетворень — надати добре знайомій антивірусу шкідливій програмі нового вигляду. В цьому випадку видозмінена версія вірусу якийсь час залишиться невідомою. Проте існують способи детектування такого роду перепакування та модифікацій.

Одним з таких методів є використання алгоритму нечіткого хешування. Якщо в криптографічних хеш-функціях суть алгоритму полягає в тому, що при найменшій зміні вхідних даних (навіть одного біта інформації) їх хеш також значно змінюється, то в нечітких алгоритмах хеш змінюється незначно або не змінюється зовсім. Тобто нечіткі хеші стійкіші до невеликих змін у файлі. Тому подібні функції дозволяють набагато ефективніше виявляти нові модифікації шкідливого ПЗ і не вимагають значних ресурсів для розрахунку.

Нечітке хешування – це набір методів, які обчислють шматкові хеші від вхідних даних, тобто використовує так зване шматкове хешування з контекстним пошуком (англ. context triggered piecewise hashing, CTPH) [41].

Алгоритм CTPH заснований на використанні змінного хешу, котрий обчислюється з використанням механізму “ковзаючого вікна” і стану. Стан зберігає хеш кількох останніх байтів даних, які містяться у поточному вікні, і побудований таким чином, що дозволяє усунути вплив попередніх байтів даних та додавати нові дані при зміні позиції ковзаючого вікна [42]. Проілюструємо на рисунку 5.5 техніку змінного хешу на прикладі алгоритму пошуку підрядків Рабіна-Карпа.

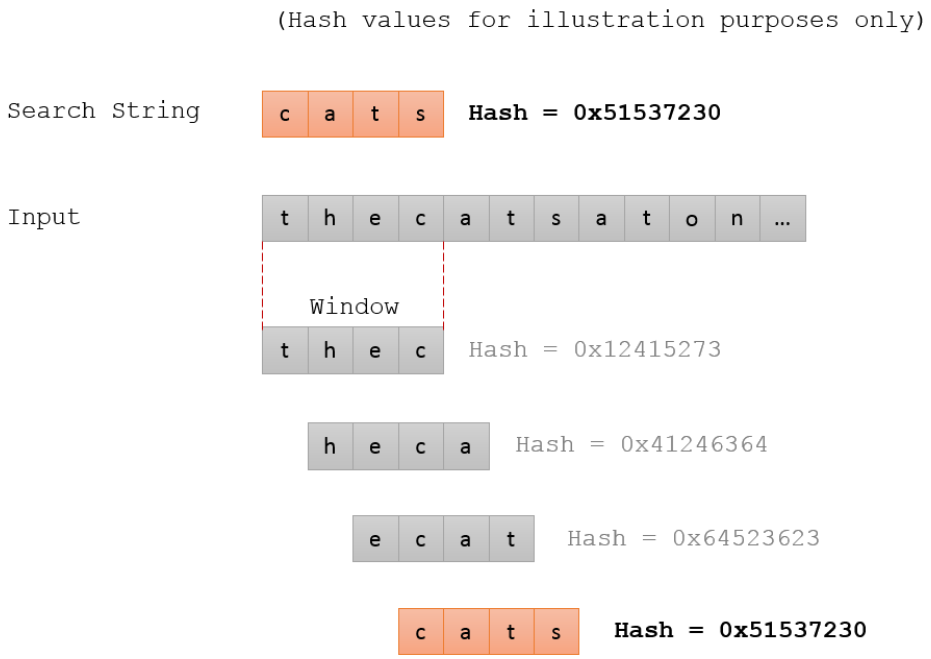

Рисунок 5.5 – Ілюстрація роботи алгоритму змінного хешу [42]

Для перевірки документів на подібність за допомогою CTPH обирається довільна точка у файлі. Далі відбувається обчислення традиційно хешу (наприклад MD5) для фрагменту файлу. Частина традиційного хешу зберігається, наприклад, останні два байти, і продовжується обчислення наступного фрагменту. Остаточний хеш CTPH складається із збережених частин традиційного хешу.

Наявні дослідження підтверджують, що використання функції нечіткого хешування є перспективним в сфері аналізу зразків шкідливого ПЗ [43]. Зокрема, особливості алгоритму можна використовувати в складі програмного забезпечення для кластеризації зразків.

В нашій системі буде застосовуватися програмна реалізація шматкового хешування з контекстним пошуком SSDeep, розроблена Джессі Корнблюм в результаті власного дослідження [44].

5.5 Обґрунтування вибору системи віртуалізації

Для аналізу зразків Cuckoo Sandbox використовує віртуальне середовище. На жаль, система не підтримує розгортання гостьових машин в хмарі, тому необхідно обрати гіпервізор, за допомогою якого буде здійснюватися віртуалізація. За замовчуванням, система працює з гіпервізором VirtualBox, проте таке рішення нам не підходить, оскільки вимагає наявності базової ОС, що вимагає додаткових ресурів. Cuckoo Sandbox підтримує більшість рішень програмних рішень для віртуалізації. Серед основних вимог при виборі системи віртуалізації були такі:

- підтримка системи модулем пісочниці Machinery;
- для більшої швидкої це має бути гіпервізор 1-го типу, тобто такий, що працює безпосередньо на “голому залізі” і не вимагає встановлення гостьової ОС.

Серед гіпервізорів даного типу можна виділити Xen, VMware ESXi, Hyper-V [45]. Для розгортання віртуального середовища нашої системи було обрано рішення VMware ESXi.

Серед переваг даного рішення слід виділити високу стабільність продукту, його простоту в адмініструванні, можливості масштабування, широкий спектр гостьових систем, які підтримуються. ESXi має дві версії: безкоштовну версію з обмеженим функціоналом та платну версію з розширеними можливостями, такими як централізоване управління всіма віртуальними машинами на всіх хостах проекту за допомогою платформи vCenter. Cuckoo Sandbox для своєї роботи потребує використання VMware API, яке доступне лише в ліцензійній версії. Окрім пісочниці, на ESXi буде розгорнуто додаткові віртуальні машини, необхідні для роботи дослідників шкідливого ПЗ.

5.6 Засоби реалізації сховища для зберігання зразків шкідливого ПЗ

Зважаючи на структуру та враховуючи вимоги до проектованої системи, одним з центральних компонентів є хмарне сховище для зберігання зібраних зразків потенційно небезпечного ПЗ. Для обраної хмарної платформи AWS таким об’єктним сховищем є *S3*.

Важливою перевагою сервісу є ряд вбудованих заходів безпеки для захисту даних. Amazon S3 здійснює шифрування даних як при завантажені об’єктів (за допомогою механізмів SSL/TLS) так і при зберіганні з використання серверного механізму шифрування (SSE). Усі корзини Amazon S3 за замовчуванням автоматично шифруються за допомогою шифрування на стороні сервера з використанням керованих ключів Amazon S3 (SSE-S3). Якщо потрібен більший контроль над своїми ключами, наприклад потрібно здійснювати ротацію ключів і керувати політиками доступу до них, можна використовувати шифрування на стороні сервера за допомогою власних ключів AWS Key Management Service (AWS KMS). Для шифрування Amazon використовує один із найнадійніших доступних блокових шифрів AES-256 [46].

Крім того, S3 пропонує гнучкі можливості щодо керування доступом до завантажених об’єктів. Можна обмежити доступ користувачів до корзини S3 на рівні аккаунту з використанням політики AWS Identity and Access Management (IAM), використовуючи списки контролю доступу (ACL), або налаштувавши політику доступу на рівні корзини (Bucket policy). Для захисту об’єктів від несанкціонованого доступу можна налаштувати багатофакторну аутентифікацію. Сервіс AWS CloudTrail надає можливість проводити моніторинг та аудит доступів до корзин та об’єктів S3.

Використання всіх цих вбудованих функцій дозволяє гарантувати, що зібрані зразки шкідливого ПЗ не потрапить в руки сторонніх осіб.

Ще однією важливою функцією S3 є можливість налаштування життєвого циклу об’єктів (Lifecycle policy). Завдяки цьому, можна реалізувати автоматичне видалення зразків через певний час, що дозволяє скоротити витрати на зберігання.

Для уникнення дублювання даних та для зменшення ресурсних витрат на аналіз зразків необхідно реалізувати механізм перевірки унікальності зібраних об’єктів. Тому для досягнення цих цілей перед завантаженням у сховище відбувається упакування зразка в архів та обчислення його контрольної суми (хешу). Отримані хеші потрібно зберігати в швидкодіючій NoSQL базі даних.

Amazon пропонує чудове рішення поставленої задачі – базу даних *DynamoDB*. Це швидкий і гнучкий сервіс нереляційних баз даних на основі пар «ключ-значення» для застосунків, що потребують стабільної затримки в кілька мілісекунд. На відміну від популярної NoSQL бази даних Redis, DynamoDB повністю керована, тож не потрібно турбуватися про адміністративну сторону підтримки та про розширення потужностей бази даних [47]. DynamoDB автоматично масштабує свої ресурси, динамічно підлаштовуючи свою пропускну спроможність в залежності від об’ємів трафіку.

Модель бази даних є нереляційною, дозволяючи використовувати такі структури даних як документи, графи та стовпчасті моделі. Дані DynamoDB зберігаються в таблицях, а взаємодія з ними відбувається за допомогою запитів GET і PUT, які відповідно є операціями читання та запису. Кількість елементів, які можна зберігати в таблиці, необмежена. Кожен запит DynamoDB виконується за первинним ключем, визначеним користувачем, який унікально ідентифікує кожен елемент.

DynamoDB здійснює автоматичну реплікацію в трьох зонах доступності для забезпечення високої доступності, довговічності та послідовності читання. Користувач також може вибрати реплікацію між регіонами, яка створює резервну копію таблиці DynamoDB в одному або кількох глобальних географічних розташуваннях.

Під час своєї роботи підсистема збору зразків перевіряє чи зразки потенційних зловмисних програм, розташовані на машині, вже були зібрані в сховище зразків. Для цього потрібно мати REST API, який дозволив би зовнішнім користувачам взаємодіяти з таблицею хешів в DynamoDB. Таку можливість забезпечує сервіс *Amazon API Gateway*.

Amazon API Gateway - це хмарний сервіс AWS, який надає можливість створення, розгортання та керування API для взаємодії з іншими хмарними сервісами та зовнішніми додатками. API Gateway дозволяє легко створювати REST, HTTP або WebSocket API, які можуть бути використані для доступу до збережених даних, обчислювальних ресурсів та інших служб AWS.

API Gateway може забезпечити:

- масштабованість: можливість масштабувати API залежно від потреб;
- керування доступом: можливість налаштування доступу до API та його ресурсів з використанням ролей та політик;
- захист API: можливість захистити API від DDoS-атак та інших загроз безпеки;
- моніторинг: можливість моніторити та аналізувати використання API та його ресурсів;
- інтеграцію з іншими сервісами AWS: можливість інтегрувати API Gateway з іншими сервісами AWS для забезпечення більш широкого спектру функціональності.

Зокрема, що важливо для розгортання нашої системи, API Gateway легко інтегрується з сервісом баз даних DynamoDB.

Для виконання програмного коду модулю генерації звітів скористаємось сервісом безсерверних обчислень *AWS Lambda*.

AWS Lambda – це платформа, яка дозволяє запускати код у відповідь на визначені події. Головною перевагою даного сервісу є те, що він безсерверний, тобто немає необхідності керувати серверами і налаштовувати оточення для виконання коду. Достатньо обрати процесорну архітектуру і програмне оточення, яке Lambda буде використовувати для виконання функції. Сервіс підтримує

багато мов програмування, включаючи Python, Node.js, Java, Ruby, Go та інші. Важливо, що AWS Lambda має добре документоване API, простий та зручний інтерфейс користувача, що дозволяє легко керувати функціями, здійснювати налагодження та моніторити їх виконання. Інтеграція з іншими сервісами всередині платформи значно розширює функціональні можливості. Зокрема, AWS Lambda дозволяє здійснювати виконання коду у відповідь на певні події від інших сервісів. В залежності від кількості запитів, сервіс може автоматично масштабуватися.

5.7 Вибір бази даних для зберігання сигнатур

Більшість компонентів системи обмінюються даними у структурованому текстовому форматі JSON. Дані про сигнатури зловмисних програм також будуть зберігатися в даному форматі. Тому важливо, щоб платформа бази даних сигнатур могла легко обробляти дані цього формату. Наш вибір впав на нереляційну об’єктну базу даних MongoDB.

MongoDB має гнучку модель даних, яка дозволяє зберігати неструктуровані дані. На відміну від реляційних баз даних MongoDB не використовує табличне представлення даних із заздалегідь заданою кількістю стовпців та типів даних. Дана система є документо-орієнтованою, у якій центральним поняттям є документ. Поля документів можуть містити різнорідну інформацію: числа, рядки, логічні значення, масиви або навіть вкладені документи. Поля можуть відрізнятися від документа до документа, а структуру даних можна змінювати з часом. Отже, все, що можна представити у вигляді JSON, можна зберігати в MongoDB і так само легко отримати в JSON.

Вбудовані методи індексації полів допомагають при роботі з великими базами даних. Загалом MongoDB забезпечує більш високий рівень продуктивності при роботі з неструктурованими даними в порівнянні з традиційними базами SQL-орієнтованими базами даних. Платформа підтримує реплікацію даних для забезпечення більш високої доступності і розподілу навантаження.

5.8 Вибір додаткових технологій

Вибір додаткових технологій включає в себе вибір мов програмування для реалізації окремих компонентів системи та програмних продуктів, що спрощують використання системи.

Для реалізації додаткових модулів статичного та динамічного аналізу та модулів взаємодії між підсистемами обрано мову програмування Python.

Вибір даної мови обумовлений наявністю численних переваг, серед яких:

- кросплатформеність: Python вбудований в платформи Linux/UNIX и MacOS, тому немає необхідності встановлювати додаткове ПЗ та витрачати час на написання окремого коду для кожного з сімейств ОС;
- обширна стандарта бібліотека і екосистема: Python має вбудований набір модулів для роботи з ОС, мережевими протоколами, архівами, криптографічними протоколами тощо; крім того існує значна кількість готових бібліотек для роботи з технологіями, що використовуються в нашій системі;
- швидкість розробки: простий синтаксис, динамічна типізація та якісне документування дозволяє писати код швидко і ефективно;
- асинхронне програмування: підтримка асинхронного коду дозволяє скоротити час виконання програм, що працюють з великими об’ємами даних; кожна одиниця такого коду виконується окремо від основного потоку, що істотно підвищує продуктивність деяких модулів нашої системи.

Для написання веб-інтерфейсу підсистеми керування сигнатурами було використано мову програмування C#.

C# – це об’єктно-орієнтована мова програмування, створена компанією Microsoft для розробки додатків для платформ .NET Framework та .NET Core. Вона поєднує в собі елементи мов C++ та Java, а також додає свої власні особливості, що забезпечує можливість швидкої розробки за стосунків різного роду. C# використовується для розробки настільних програм, веб-додатків, мобільних додатків та інших.

Для реалізації веб-застосунку обрано фреймворк ASP.NET Core, який спрощує розробку веб-API та користувацького інтерфейсу. Серед переваг даного інструменту можна виділити кросплатформеність, швидкість розробки, модульність та високу продуктивність. Висока сумісність з концепцією MVC дозволяє розділити програму на компоненти, що забезпечує більшу організацію та зрозумілість коду.

Одним з компонентів для організації робочого процесу аналізу зразків є програмна платформа для управління проєктами. В нашій системі даний інструмент необхідний для формування звітів щодо досліджуваних зразків та для організації спільної роботи аналітиків шкідливого ПЗ. Вибір впав на популярний інструмент від компанії Atlassian – JIRA.

Jira – це програмний продукт, створений для управління проєктами, що має велику популярність серед великих компаній. Задачею платформи є створення завдань, призначення виконавців, встановлення пріоритетів і відстеження виконання цих завдань.

Головними елементами JIRA є задача (англ. issue або ticket) і робочий процес (англ. workflow). Кожна задача включає в себе назву проєкту, тему, тип, пріоритет, компоненти та зміст. Задачі можуть бути розширені додатковими полями (в тому числі визначеними користувачем), додатками або коментарями. Задачі можна приєднувати одна до одної та створювати залежності між ними. Кожна задача має свій визначений в конкретний момент часу статус, наприклад «нова» в «в процесі», «закрита». Порядок зміни статусу задачі регламентується згідно заздалегідь налаштованого робочого процесу. Наприклад, найпростішим робочим процесом є «заплановано» – «в роботі» – «зроблено».

Основними перевагами даної програмної платформи є великий інтеграційний потенціал, можливість розширення функціоналу та велике різноманіття налаштувань. Можливість гнучкого налаштування робочого процесу важливо для організації роботи основних акторів нашої системи – вірусних аналітиків.

Висновки до розділу

В даному розділі були обрані технічні засоби для побудови системи по формуванню бази даних сигнатур шкідливих програм та визначено допоміжні технології для формування ефективних сигнатур.

Для розгортання компонентів системи обрано хмарну платформу AWS. Автоматичний аналіз досліджуваних разків був здійснюватися за допомогою програмного комплексу Cuckoo Sandbox, розвернутого в середовищі віртуалізації VMware ESXi.

При формуванні сигнатур окрім хеш-функцій буде застосовуватися інструмент на основі правил YARA, огляд якого здійснено в цьому розділі. Також розглянуто алгоритм нечіткого хешування SSDeep, який слугуватиме для виявлення і кластеризації подібних зразків. Для зберігання сформованих правил обрано нереляційну базу даних MongoDB.

6 РОЗГОРТАННЯ СИСТЕМИ ФОРМУВАННЯ БАЗИ СИГНАТУР КОМП'ЮТЕРНИХ ВІРУСІВ

6.1 Архітектура підсистеми зберігання зразків

На рисунку 6.1 зображено архітектуру даної підсистеми, розгорнутої в хмарній платформі AWS.

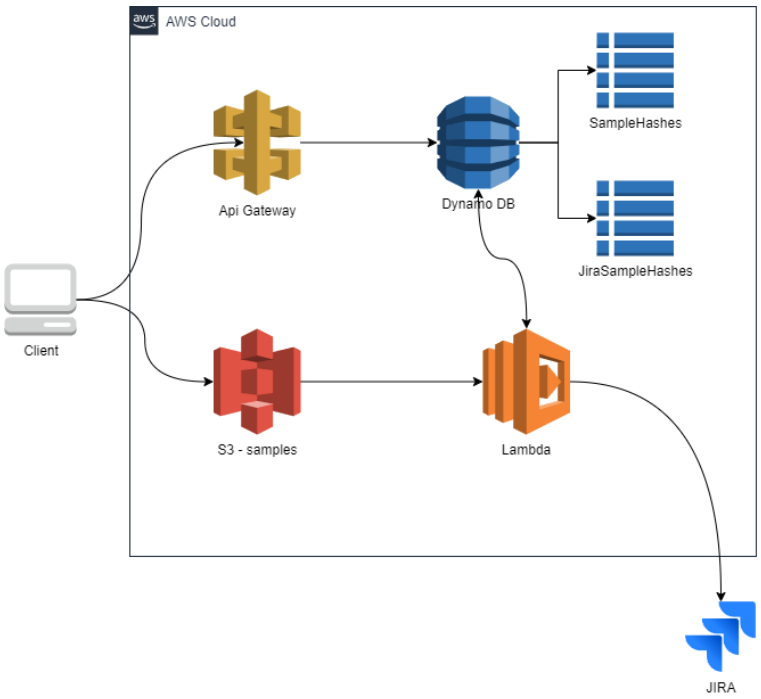

Рисунок 6.1 – Архітектура підсистеми зберігання зразків

Дані в систему надходять від модуля збору шкідливого програмного забезпечення, встановленого на ПК клієнта. Перед відправкою зразка в сховище відбувається перевірка того, чи не завантажувався аналогічний зразок в наше сховище. Для цього ПО на стороні клієнта робить POST запит */match-sample-hashes* на API Gateway. Сервіс API Gateway підтримує безпосередню інтеграцію з базою DynamoDB, яка містить перелік хешів для існуючих в сховищі зразків. Структуру бази описано в таблиці 6.1.

Таблиця 6.1 – Структура таблиць бази DynamoDB

Таблиця	Опис	Поля
SampleHashes	SHA-256 хеш наявних в сховищі зразків	SourceSampleHashId
JiraSampleHashes	Описує зв’язок між наявними в сховищі зразками і створеними звітами в тікет-системі	SampleHashId TicketKey

Якщо API відповідає, що знайденого зразка в базі ще немає, відбувається передача архіву зі зразком в сховище S3. Разом з ним передається розгорнутий звіт у форматі JSON, що містить інформацію про систему користувача. Для користувачів ОС MacOS збирається наступна інформація.

Загальна інформація про ОС:

- UUID (унікальний ідентифікатор для кожної кінцевої машини);
- версія ОС;
- локаль (мова системи);
- країна.

Розширена системна інформація:

- інформація про розширення ядра;
- історія введених команд;
- запущені процеси;
- список встановлених програм;
- список встановлених пакетів;
- список програм, що завантажуються при старті системи;
- статистика мережевих інтерфейсів;
- список системних користувачів;
- список файлів, які відповідають критеріям шкідливого ПЗ;

- файл /etc/hosts, який містить список доменних імен та відповідних ним IP-адрес.

Зібрана інформація може знадобитися вірусному аналітику для розуміння особливостей роботи шкідливого ПЗ та для виявлення спільних рис різних зразків.

Модуль генерації звітів є відправною точкою нашої системи для процесу дослідження шкідливого ПЗ. Модуль здійснює обробку доданих в сховище даних та формує зручний для обробки людиною звіт. Даний модуль реалізований мовою програмування Python та розгорнутий в середовищі AWS Lambda. Тригером для запуску Lambda функції є факт появи нового об'єкту в сховищі S3: архіву зі зразком шкідливого ПЗ або JSON-файлу з інформацією про систему.

Генерується звіт двох видів.

1) Звіт про зразок – включає хеш отриманого зразка, базову інформацію про нього, посилання на файл в сховищі S3. Пізніше, після проведення автоматичного аналізу до звіту прикріплюються результати цього аналізу (рис 6.2).

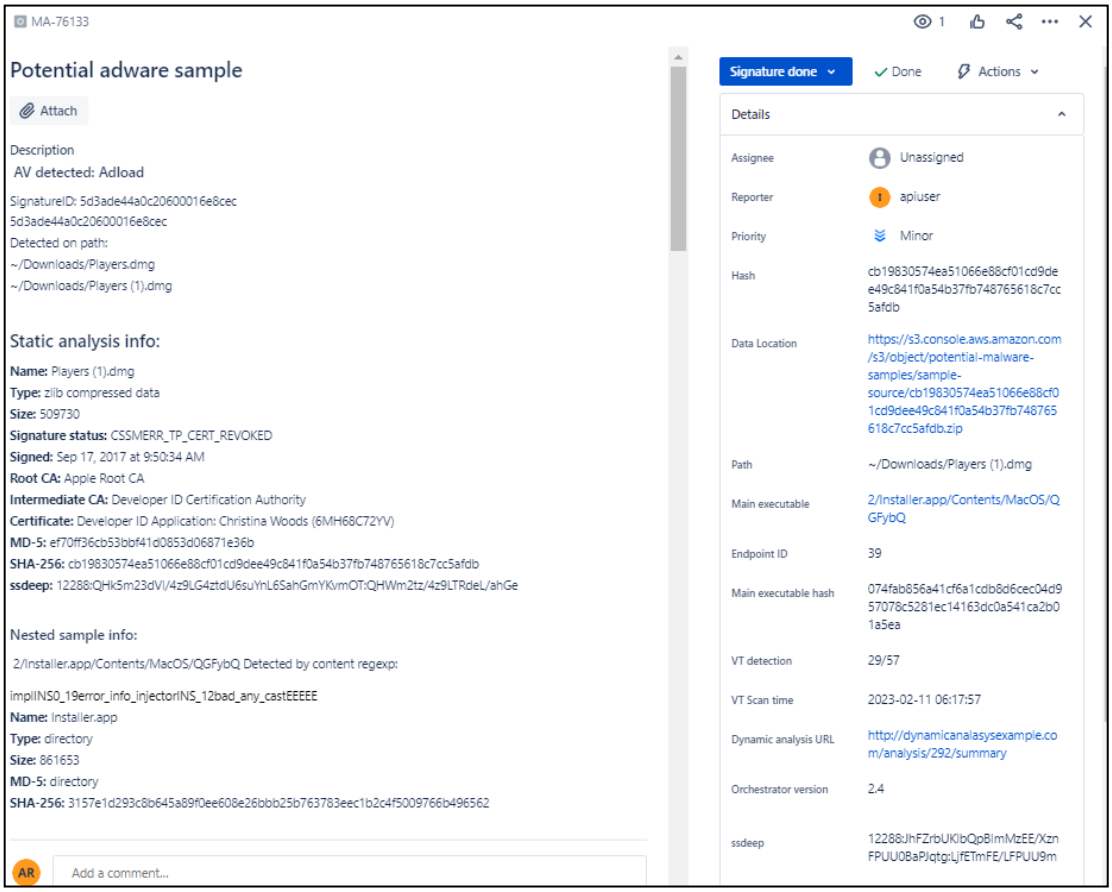

Рисунок 6.2 – Приклад згенерованого звіту про зразок в платформі JIRA

2) Звіт про кінцеву машину – містить інформацію про персональний пристрій користувача, на якому було виявлено потенційно шкідливе ПЗ (рисунок 6.3).

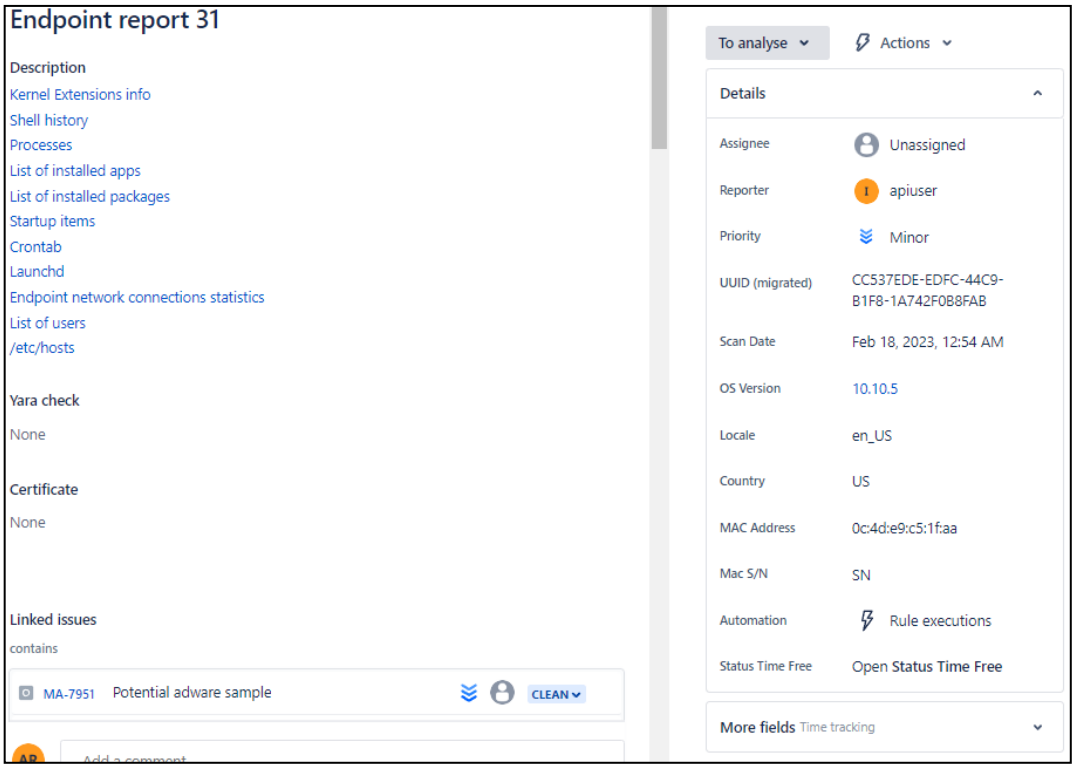

Рисунок 6.3 – Приклад згенерованого звіту про кінцеву машину

6.2 Архітектура підсистеми автоматичного аналізу зразків

Аналізатор зразків шкідливого ПЗ – це програмний модуль, реалізований мовою Python, який призначений для автоматизації процесу масового аналізу зразків шкідливих програм. Модуль працює з даними, отриманими зі сховища зразків. Він складається з двох функціональних підмодулів:

- оркестратор – скрипт orchestrator.py, слідкує за появою нових зразків в сховищі та здійснює роботу по автоматизації процесу їх дослідження;
- аналізатор – скрипт analyser.py виконує збір статичної інформації зразків.

Алгоритм роботи кожного з підмодулів у вигляді блок-схем наведено на плакаті 4 та плакаті 5 додатку А відповідно.

Розглянемо загальний алгоритм роботи вказаної підсистеми.

Крок 1. Здійснюється пошук не проаналізованих звітів (заявок) про знайдені зразки. Вибірка здійснюється шляхом створення запиту в тікет-систему JIRA. Прикладом такого запиту мовою запитів JQL:

project=PMA and createdData >= -1d and status = "To analyse" and type = "Sample" and "Orchestration version" is EMPTY ORDER by key DESC

В залежності від потреб аналітика, запит може бути змінений вручну.

Крок 2. На основі базової інформації щодо зразка, наведеної в звіті, відбувається перевіряється на відповідність критеріям білого списку. В залежності від типу досліджуваного зразка такими критеріями можуть бути: цифровий підпис програми, ідентифікатор пакету, інформація про розширення браузера. Якщо файл знаходить в білому списку статус заявки змінюється на “Clean” і система переходить до аналізу наступного зразка.

Крок 3. Завантаження зразка зі сховища S3 на локальний носій і його розпакування.

Крок 4. Перевірка зразка власним антивірусним фреймворком: зразок потенційно небезпечної програми копіюється в усі можливі директорії, де він був виявлений раніше (ця інформація прикріплена раніше до звіту) і запускається його сканування. Якщо в результаті сканування зразок видалений з усіх можливих директорій, це означає, що його сигнатура вже наявна в нашій базі і статус заявки змінюється на “Signature Done”. Якщо зразок після сканування не видалений, то необхідно провести його дослідження, статус заявки змінюється на “To analyse”. Якщо зразок був видалений тільки з частини можливих директорій, це означає, що слід вдосконалити сигнатури, до заявки додається мітка “partial_detection”. Загальний робочий процес вірусного аналітика в менеджері проєктів JIRA зображено на рисунку 6.4.

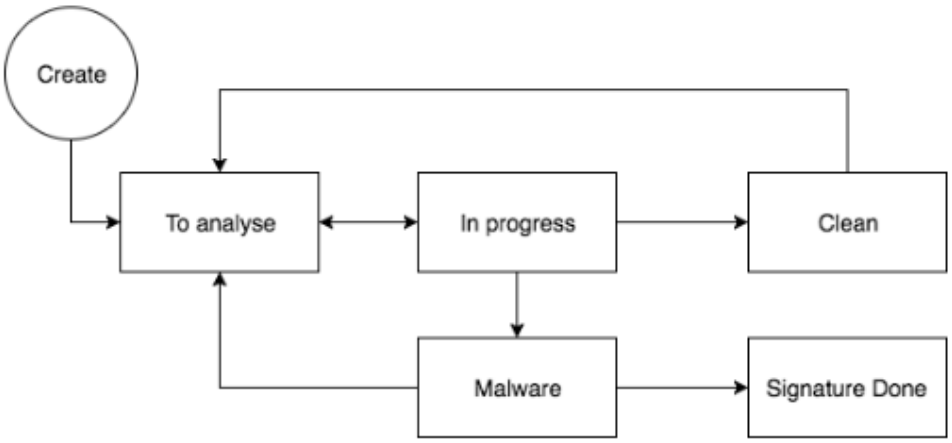

Рисунок 6.4 – Діаграма робочого процесу вірусного аналітика в JIRA

Крок 5. Визначення типу зразка шляхом перевірки розширення та файлових сигнатур (Magic bytes), тобто перших байтів файлу[24]. Типи файлів, які підтримуються системою, зазначені в нефункціональних вимогах.

Крок 6. Вилучення статичної інформації про зразок.

Основні поля, які вилучаються з кожного досліджуваного зразка та їх опис наведено в таблиці 6.2.

Таблиця 6.2 – Статична інформація, що вилучається зі зразків

Параметр	Опис
Name	Назва файлу
SHA-256	Контрольна сума файлу, обчислена алгоритмом SHA-256
Type	Тип файлу, визначений на попередньому етапі
DeveloperSignature1	Перший рядок цифрового підпису програми
DeveloperSignature2	Другий рядок цифрового підпису програми
DeveloperSignature3	Третій рядок цифрового підпису програми
SignatureStatus	Статус цифрового підпису програми (дійсний, відхилений, прострочений)
Ssdeep	Нечіткий хеш для пошуку подібних зразків
Original file path	Шлях до файлу

В залежності від типу досліджуваного зразка вилучається додаткова інформація. Наприклад, для виконуваних файлів Mach-O [48] збирається інформація про бібліотеки програми, імпортовані функції, експортовані функції та інше.

Крок 7. Відправка зразка для проведення динамічного аналізу в Cuckoo Sandbox на різних версіях ОС. В Cuckoo не відправляються зразки, які були внесені раніше в білий список. Зразок виконується всередині ізольованої віртуальної машини та збирається наступна інформація:

- мережева активність;
- системні виклики;
- список створених, модифікованих та тих, до яких здійснено доступ файлів і директорій;
- видалені об’єкти;
- знімки екрана віртуальної машини.

Крок 8. Перетворення зібраної інформації в зручний для обробки людиною формат і прикріплення її до звіту про зразок в JIRA.

На плакаті 6 додатку А наведено діаграму послідовності, що описує шлях, яких проходить зразок в процесі його аналізу.

6.3 Розгортання віртуального середовища для автоматизованого аналізу зразків

Розгортання програмного комплексу для автоматизованого аналізу зразків Cuckoo Sandbox здійснено у віртуальному середовищі VMware ESXi. Архітектура даного комплексу передбачає наявності головної керуючої машини (хоста) і гостьових машин, на яких буде здійснюватися емуляція. Враховуючи нефункціональні вимоги, для розгортання зазначених компонентів створено віртуальні машини з характеристиками, наведеними в таблицях 6.3 та 6.4.

Таблиця 6.3 – Характеристики керуючої машини

Компонент	Характеристика
CPU	8 ядер з тактовою частотою 3.7 ГГц
RAM	64 Гб
Диск	4 Тб
Операційна система	MacOS 10.15

Таблиця 6.4 – Характеристики гостьових машин

Компонент	Характеристика
CPU	4 ядер з тактовою частотою 3.2 ГГц
RAM	8 Гб
Диск	100 Гб
Операційна система	MacOS 10.14, MacOS 10.15, MacOS 11, MacOS 12, MacOS 13

Налаштування керуючої машини включає в себе кілька етапів, серед яких:

1) встановлення і налаштування оточення Python та залежностей;
2) встановлення додаткових системних бібліотек;
3) встановлення MongoDB для роботи веб-інтерфейсу на основі фреймворку Django;
4) встановлення останньої стабільної версії cuckoo;
5) встановлення утиліти tcpdump для можливості аналізу мережевої активності під час виконання зразка;
6) завантаження коду для роботи інтеграцій та додаткових модулей аналізу.
7) конфігурація Cuckoo.

Конфігурація пісочниці відбувається за допомогою конфігураційних файлів. В таблиці 6.5 здійснено опис основних файлів налаштувань та параметрів, які слід задати для роботи Cuckoo Sandbox.

Таблиця 6.5 – Опис основних параметрів конфігурації Cuckoo Sandbox

Параметр	Файл конфігурації	Опис
machinery	cuckoo.conf	Задає технологію віртуалізації для взаємодії з гостьовими машинами аналізу
[resultserver] ip, port	cuckoo.conf	IP-адреса та порт, на якому працюватиме сервер результатів
[sniffer]	auxiliary.conf	Увімкнення та шлях до аналізатора трафіку (tcpdump)
[mitm]	auxiliary.conf	Увімкнення проксі-серверу для аналізу https-трафіку
[vsphere] label platform snapshot	vsphere.conf	Параметри для розвертання гостьових віртуальних машин: назва, тип ОС, номер знімку віртуальної машини. Можна налаштувати декілька різних машин для проведення аналізу
[memory] [network] [dropped] [screenshots]	processing.conf	Увімкнення доступних модулів для обробки результатів аналізу: дамп пам’яті, аналіз трафіку, аналіз вилучених файлів, знімки екрана під час виконанння
[mongodb]	reporting.conf	Параметри підключення локальної БД для зберігання результатів аналізу

Процес налаштування гостьових машин дещо простіший і включає в себе наступні етапи:

1) встановлення Python і залежностей;
2) конфігурація проксі для перехоплення https-запитів;
3) встановлення бібліотек для аналізу браузер них розширень;
4) встановлення і запуск cuckoo агента;
5) встановлення додаткового ПЗ для здійснення аналізу;
6) створення знімку сконфігурованої віртуальної машини.

Для покращення якості аналізу окрім стандартних модулів статичного аналізу, розгорнуто наступні допоміжні модулі:

- Whitelist – модуль, що допомагає уникнути подальшого аналізу надійних зразків (таких як програми з дійсними підписами тощо);
- VT check – модуль для перевірки зразків на VirusTotal [25] за допомогою хешу зразка; дозволяє визначити чи задетектований зразок іншими популярними антивірусними рішеннями;
- антивірусний модуль пісочниці – перевірить наданий зразок за допомогою встановленого в пісочниці антивірусного клієнта;
- статичний аналіз розширень браузера – це інструмент, який виконує статичний аналіз розширень браузера;
- Linker – модуль зв’язує зразки, які мають щось спільне;
- автоматизований модуль створення звітів, який створює простий, зрозумілий для людини звіт щодо аналізу.

Для динамічного аналізу розвернуто і налаштовано наступні модулі:

- модуль аналізу поведінки – передбачає перевірку роботи зразка в лабораторних умовах, щоб зрозуміти його активність у файловій системи, процеси і мережеву діяльність;
- автоклікер – інструмент, який дозволяє автоматизувати інсталятори, які потребують взаємодії з людиною;
- проксі-сервер – інструмент, який дозволяє аналізувати мережевий трафік з тестової системи;

- модуль для аналізу розширень (плагінів) браузера;
- модуль вилучених файлів, який збирає інформацію про вилучені файли в системі після аналізу;
- модуль оцінки, який встановлює оцінку зловмисності для кожного проаналізованого зразка.

Після проведення всіх налаштувань відбувається запуск пісочниці та веб-інтерфейсу командою “cuckoo && cuckoo web runserver 0.0.0.0:80”. Передача зразків на аналіз здійснюється через API. На рисунку 6.5 зображено підсумок аналізу підозрілого файлу. Звіт також включає оцінку шкідливості кожної з активностей, що здійснює програма (рисунок 6.6).

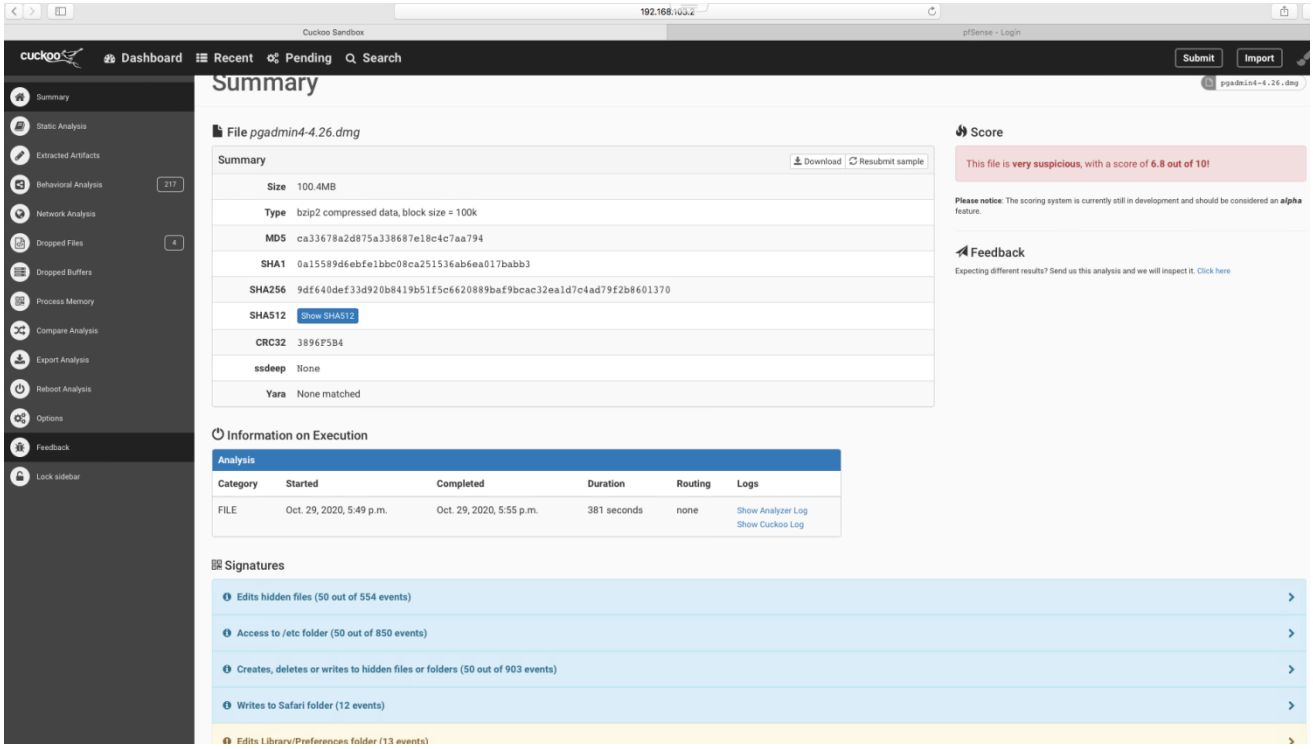

Рисунок 6.5 – Результат аналізу підозрілого файлу в середовищі Cuckoo Sandbox

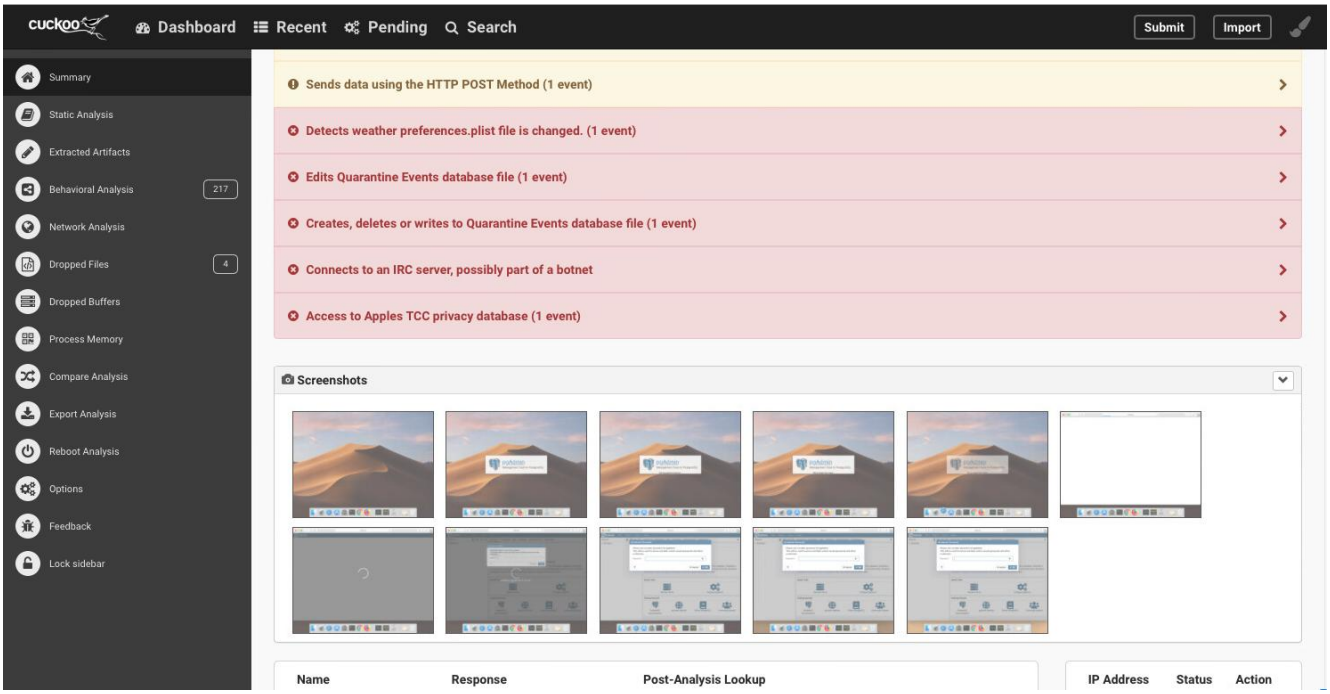

Рисунок 6.6 – Оцінка шкідливості проаналізованого зразка

Панель зліва дозволяє переглянути більше детальні результати аналізу кожного з модулів. Наприклад, на рисунку 6.7 зображено звіт про мережеву активність.

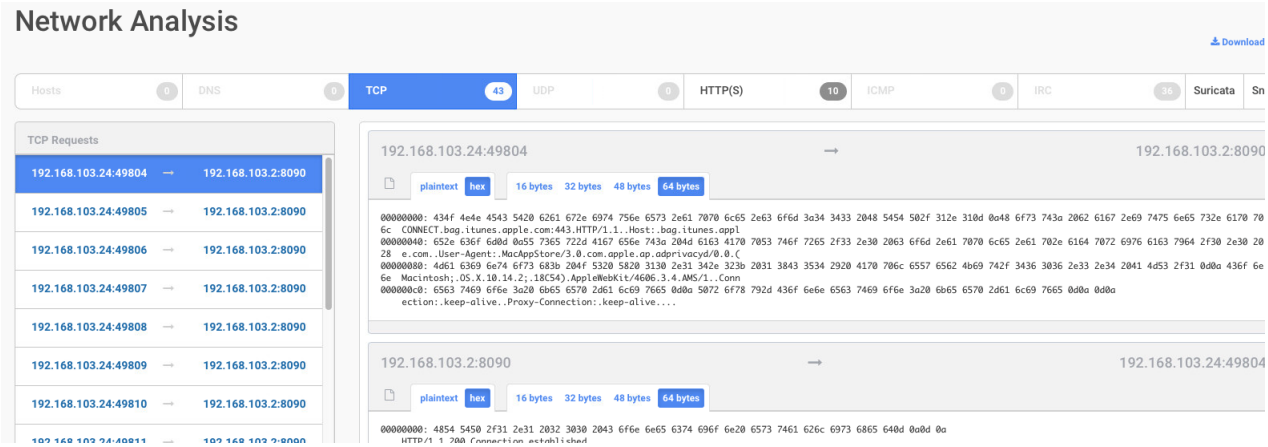

Рисунок 6.7 – Результати аналізу мережевої активності програми

Підсумки аналізу автоматично прикріпляються до заявки про зразок, що видно на рисунку 6.2.

6.4 Застосування алгоритму нечіткого хешування модулем генерації сигнатур

Для того, щоб зробити генерацію правил Yara більш швидкою і ефективною вирішено розділяти зразки на кластери. Для цього скористаємося особливістю алгоритму нечіткого хешування SSDeep, яка дозволяє оцінити процент подібності файлів.

Скористаємось інструментом кластеризації з відкритим кодом ssdc, який, групує файли, використовуючи ssdeep як алгоритм порівняння. Результати записуються у файл tar, який розміщує файли в каталозі з файлами, що порівнюються. Один файл може входити до кількох груп.

Недоліком даного інструменту є те, що він намагається створювати кластери, якщо процент подібності файлів більше 0. Якщо створити сигнатури для групи таких файлів, вони будуть не досить точними. Тому вирішено модифікувати даний інструмент та додати можливість задавати мінімальний процент подібності.

Постає задача підібрати оптимальне значення проценту подібності, який дозволяв би створювати мінімальну кількість груп зразків та не спотворювати точність сформованих сигнатур для кожної з груп. Для дослідження цього параметру візьмемо групу з 523 зразків шкідливого ПЗ сімейства Pirrit та проведемо кластеризацію з різним значенням проценту подібності. Засіб ssdc дозволяє також сформувати граф для візуалізації результатів кластеризації. На рисунку 6.8 зображено граф кластеризації групи зразків з параметрами проценту подібності за замовчуванням. Кожен вузол представляє зразок шкідливого програмного забезпечення, а кожне ребро представляє результат порівняння ssdeep, більший за нуль.

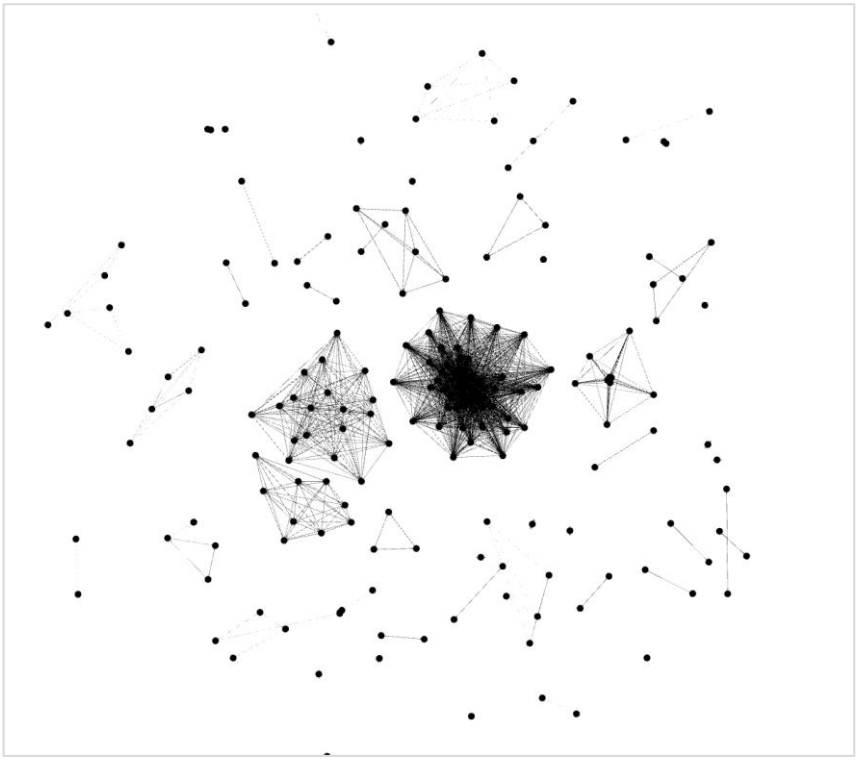

Рисунок 6.8 – Граф результатів кластеризації групи зразків з процентом подібності більше 0

В результаті створено досить велику кількість дрібних груп, для кожної з яких треба створити окреме правило YARA. Для автоматизації створення правил скористаємося утилітою генерації правил yaraGen.

Щоб зменшити кількість груп, а відповідно і правил будемо поступово збільшувати процент подібності та повторювати кластеризацію. Далі будемо генерувати правила утилітою yaraGen та перевіряти їх утилітою yara.

Емпіричним шляхом встановлено, що найкращі результати кластеризації без втрати точності демонструються, якщо вказати процент подібності 44. На рисунку 6.9 зображено граф кластеризації з процентом подібності 44.

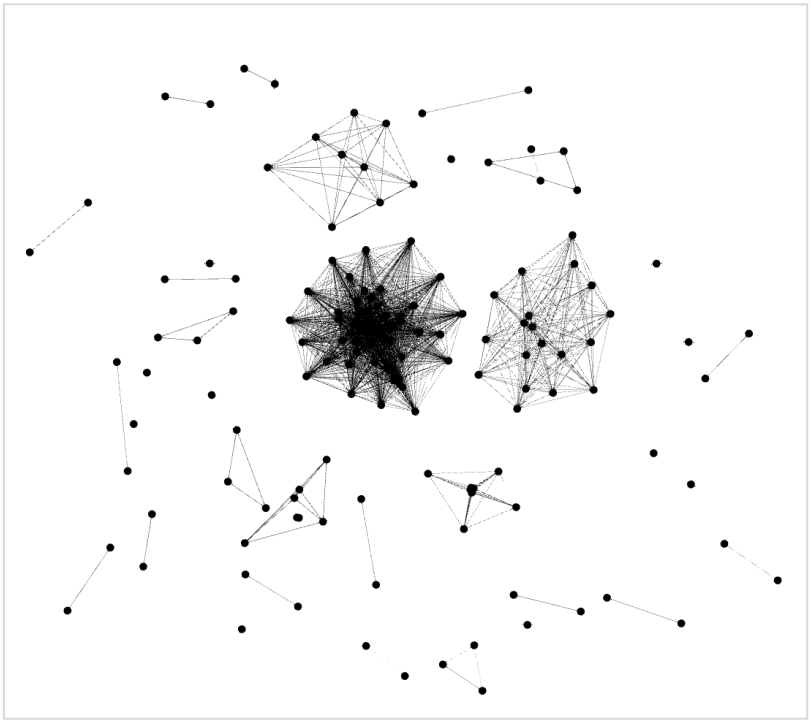

Рисунок 6.9 – Граф результатів кластеризації групи зразків з процентом подібності більше 44

Візуальне представлення за допомогою графів демонструє зменшення кількості груп зразків.

Для створення ефективних правил Yara розроблено скрипт yara_clusterer.py. Скрипт є простим API між аналітиком зловмисного ПЗ і нашими інструментами: інструментом кластеризації ssdc і генераторами правил Yara (Yargen, Yabin). Це дозволяє генерувати правила Yara за досить короткий проміжок часу.

Даний модуль спочатку перевіряє наявність більше 50 зразків у вказаній директорії та пробує кластеризувати їх за схожістю 44%. Це дозволяє зменшити кількість зразків, необхідних для створення правила для сімейства та скоротити розмір правил Yara без втрати продуктивності. Наприклад: на початку ми маємо 1000 зразків сімейства зловмисних програм “Adload”, після кластеризації ми маємо 700 зразків. Правило буде створено для цих 700 зразків, але виявить усі 1000 зразків даного сімейства шкідливих програм. Якщо кількість зразків менше 50, модуль просто створить окреме правило Yara для кожного з них. Немає сенсу кластеризувати менше 50 зразків.

6.5 Розробка веб-інтерфейсу підсистеми управління сигнатурами

Веб-інтерфейс слугує проміжною ланкою для взаємодії вірусного аналітика з базою даних сигнатур. З його допомогою відбувається перегляд, редагування, активація, деактивація та внесення нових сигнатур програмного забезпечення в базу даних. Більш детально з даними процедурами можна ознайомитись, розглянувши сценарії використання системи.

При розробці даного додатку використано технологію ASP.NET Core, яка реалізує концепцію проектування модель-представлення-контролер (MVC). Даний підхід передбачає поділ системи на три компоненти, між якими розподілені функції [49].

Модель – складова, яка представляє дані та бізнес-логіку програми. Модель відповідає за доступ до даних та їх оновлення відповідно до бізнес-правил.

Представлення – це складова, яка відповідає за відображення даних та взаємодію з користувачем, тобто визначає зовнішній вигляд програми.

Контролер – це компонент, який приймає вхідні дані від користувача, обробляє їх та передає моделі для оновлення даних. Контролер також відповідає за взаємодію між моделлю та представленням, що дозволяє керувати відображенням даних.

Контролер представлений кількома класами, серед яких:

- UsersController, що відповідає за дії над користувачами системи: створення, авторизація, редагування, зміна паролю, відновлення паролю та інші;
- EnvironmentsController – відповідає за підтримку робочих оточень (development, stage, production);
- ItemsController – відповідає за взаємодію з функціональними сторінками застосунку (Explore, Create, Edit);
- VariablesController – здійснює обробку параметрів, що зберігаються в базі даних.

Модуль представлення містить функціонал, необхідний для відображення користувацького інтерфейсу. Для створення готових елементів інтерфейсу була використана бібліотека компонентів Bootstrap Material та jQuery. Загальний зовнішній вигляд сторінки перегляду сигнатур зображено на рисунку 6.10, приклад форми редагування сигнатури на рисунку 6.11.

Авторизація користувачів реалізована за допомогою вбудованого функціоналу ASP .NET Identity. Для зберігання даних користувачів слугує локальна БД Sqlite, а функціонал взаємодії з нею реалізується класом DomainContext з використанням Entity Framework.

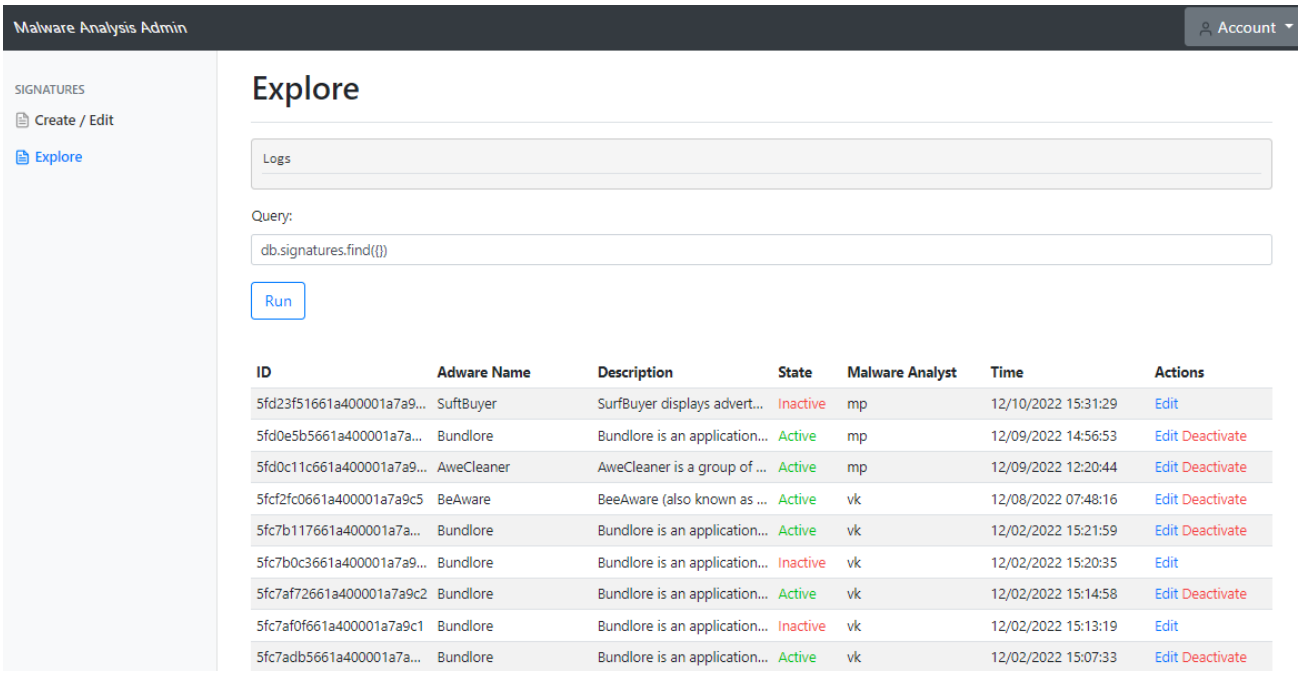

Рисунок 6.10 – Приклад сторінки перегляду сигнатур

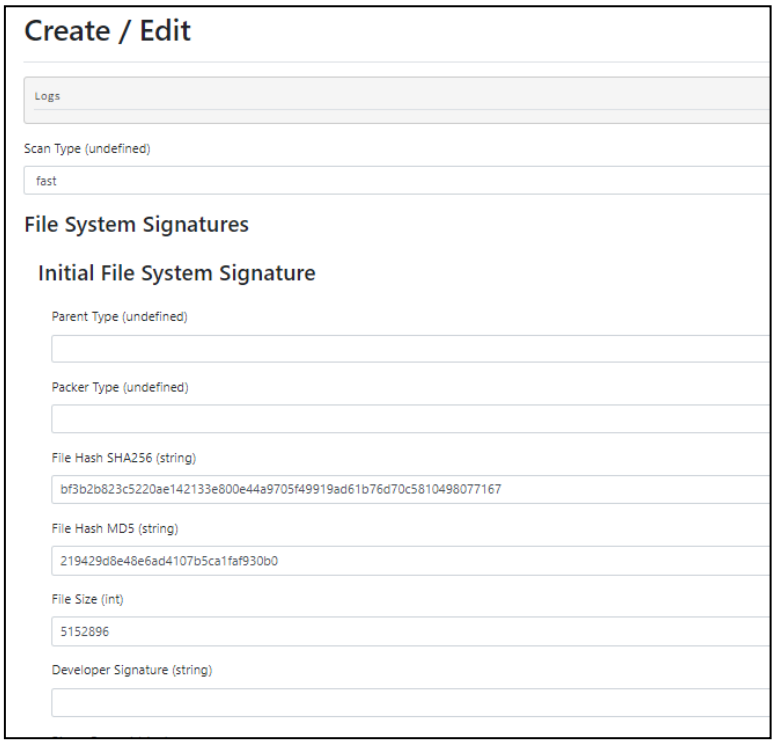

Рисунок 6.11 – Зовнішній вигляд сторінки редагування сигнатур

6.6 Розгортання бази даних вірусних сигнатур

Для забезпечення високої доступності бази даних сигнатур, балансування навантаження та забезпечення можливості горизонтального масштабування вирішено розгортати кластерний варіант бази MongoDB. Архітектура кластеру зображена на рисунку 6.12.

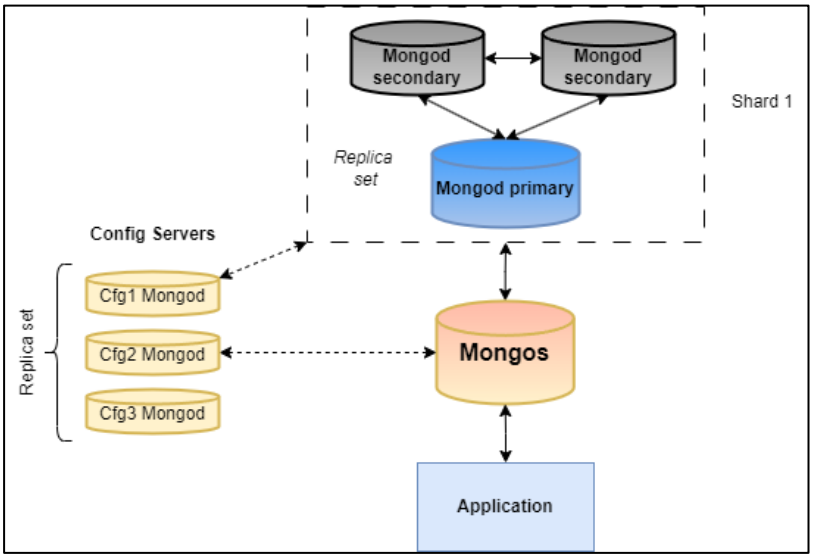

Рисунок 6.12 – Структура кластеру MongoDB

Взаємодія між веб-застосунком і базою даних сигнатур відбувається через компонент Mongos. Mongos в MongoDB – це балансувальник, який слугує посередником між додатком та кластером. Екземпляри mongos направляють запити читання та операції запису на шард.

Шард (Shard) – це компонент, який зберігає в собі підмножину даних колекції, це дозволяє розподілити навантаження на декілька серверів для забезпечення більшої продуктивності та можливості зберігання більшого обсягу даних. Кожен шард являє собою повторюючу копію (Replica Set).

Replica Set (набір реплік) – це механізм, який забезпечує відновлюваність і доступність даних шляхом реплікації (копіювання) даних між декількома вузлами даних. Replica Set складається з одного головного вузла (primary node), який приймає всі записи до бази даних, та кількох вузлів-реплік (secondary node), які копіюють дані з головного вузла. Якщо головний вузол стає недоступним через збій або відключення, Replica Set автоматично обирає новий головний вузол з наявних вузлів-реплік.

Кожен сервер конфігурації (Config Server) містить в собі метадані про кластер. Тобто інформацію про те, яким чином дані розкидані по шардах [50].

Коротко розглянемо структуру розгорнутої бази даних. Дані щодо сформованих сигнатур шкідливого програмного забезпечення зберігаються в колекції signatures. Серед основних полів колекції можна виділити:

- file_system_signature – об’єкт, який містить собі вкладену інформацію про шкідливий файл: хеш, назву файла, шляхи, цифровий підпис програми, регулярний вираз, що описує вміст файлу;
- yara_signature_base64 – YARA-правило, що описує сигнатуру, закодоване алгоритмом base64;
- network_signatures – об’єкт, що описує мережеву активність програми і включає IP-адреси і домени, з якими взаємодіє шкідлива програма;

- adware_metadata – містить метадані про знайдений зразок: назву, тип, джерело, короткий опис і номер заявки в JIRA, яка містить детальний опис зразка;
- adware_analyst_id – ідентифікатор аналітика, який створив сигнатуру;
- false_positive_level – число від 0 до 10, що характеризує рівень точності сигнатури;
- signature_status – описує поточний статус сигнатури: активний чи неактивний.

Детальна структура колекції signatures відображена на плакаті 8 додатку А.

Висновки до розділу

Даний розділ містить опис архітектури окремих структурних компонентів системи по формуванню бази даних сигнатур комп’ютерних вірусів.

Сховище зразків, база даних та веб-інтерфейс управління сигнатурами розгорнуто в хмарній платформі AWS з врахуванням вимог до безпеки, масштабованості, доступності та функціональності. Вибір таких безсерверних рішень як S3, Lambda, API Gateway, DynamoDB дозволяє розгорнути сховище зразків без необхідності його подальшого адміністрування.

Для організації автоматизованого аналізу отриманих зразків розроблено модуль “оркестратор”, алгоритм роботи якого описано в даному розділі. Оркестратор реалізує частину функціоналу, що стосується статичного аналізу та створює завдання для динамічного аналізу, який виконується в ізольованому віртуальному середовищі (пісочниці). Особливості процесу розгортання та налаштування пісочниці Cuckoo Sandbox також описано в цьому розділі.

Результатом автоматизованого аналізу отриманих зразків є звіт, на основі якого вірусні аналітики формують сигнатури. Виконання операцій з сигнатурами здійснюється аналітиками з використанням веб-інтерфейсу управління сигнатурами, реалізованого за допомогою фреймворку C# ASP.NET. Для зберігання сформованих сигнатур використовується кластерний варіант бази даних MongoDB. Діаграму розгортання системи наведено на плакаті 7 додатку А.

7 ТЕСТУВАННЯ РОБОТИ СИСТЕМИ

Дана система була впроваджена в компанії, що розробляє програмне забезпечення у сфері комп'ютерної безпеки. Оцінка ефективності системи проводиться шляхом дослідження статистики отриманих зразків та їх сигнатур, зібраної протягом одного календарного року.

Спочатку підведемо статистику щодо типів шкідливого ПЗ, сигнатури якого сформовано з використанням даної системи. Для цього здійснено вибірку з бази даних сигнатур з агрегацією по полю "$adware_metadata.adware_type". Запит і результати виконання відображено на рисунку 7.1.

```
> db.signatures.aggregate([ {$group : {_id:"$adware_metadata.adware_type", count:{$sum:1}}},{$sort:{count:-1}}])
{ "_id" : "Adware", "count" : 5889 }
{ "_id" : "PUP", "count" : 543 }
{ "_id" : "Malware", "count" : 349 }
{ "_id" : "Unwanted", "count" : 50 }
```

Рисунок 7.1 – Результати агрегації типів зібраного шкідливого ПЗ

Для візуального відображення отриманих результатів скористаємось круговою діаграмою (рисунок 7.2).

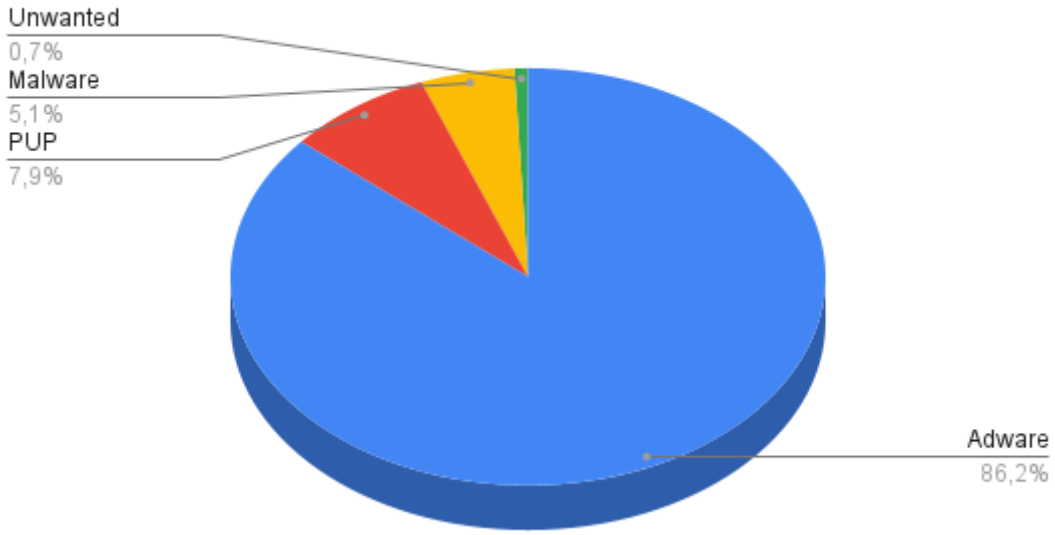

Рисунок 7.2 – Співвідношення типів знайденого шкідливого ПЗ

Таким чином, бачимо, що найбільше сигнатур створено для рекламного шкідливого програмного забезпечення. На другому місці знаходяться потенційно небажані програми. Отримана статистика у процентному співвідношені корелює з загальною статистикою, зібраною дослідниками Malwarebytes [51] щодо типів шкідливого ПЗ на MacOS.

Далі більш детально проаналізуємо сімейства знайдених шкідливих програм шляхом вибірки сигнатур з бази по полю “$adware_metadata.adware_name”. На рисунку 7.3 представлено результати вибірки у вигляді гістограм.

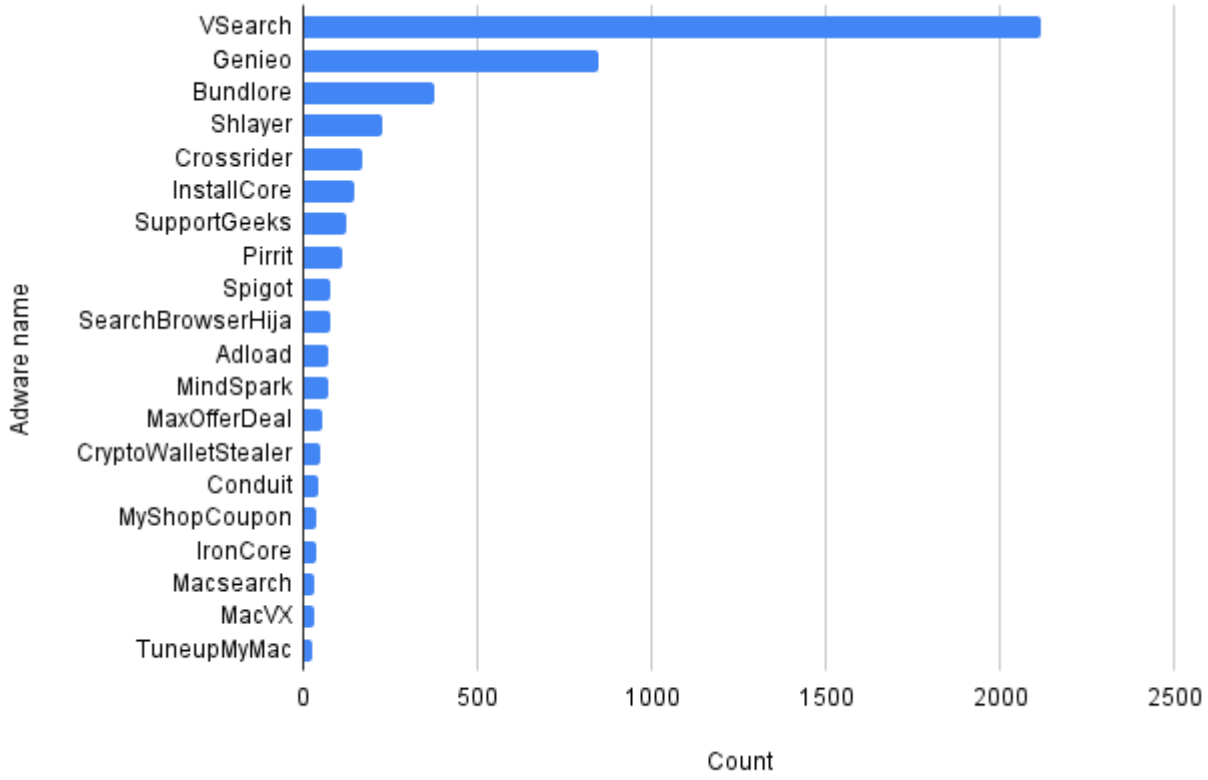

Рисунок 7.3 – Статистика сімейств шкідливих програм, зібраних системою

В таблиці 7.1 наведено опис окремих представників шкідливого ПЗ, зібраних з використанням проектованої системи, для розуміння специфіки їх роботи та методів поширення.

Таблиця 7.1 – Опис шкідливих програм, зібраних системою

Назва шкідливого ПЗ	Тип шкідливого ПЗ	Опис
VSearch	Adware	Рекламна програма, яка перетворює певні слова на веб-сторінці на гіперпосилання або відображає спливаючу рекламу
Genieo	Adware	Після встановлення він змінює домашню сторінку веб-браузера та пошукову систему за умовчанням
Bundlore	Adware	Програма, яка містить рекламне ПЗ, встановлює панелі інструментів або відображатиме спливаючу рекламу на комп'ютері
Shlayer	Adware/Trojan	Програма, яка часто йде разом із програмами, які показують рекламу, встановлює панелі інструментів й має інші незрозумілі наміри
Crossrider	Adware/Trojan	Рекламний троян, який змушує веб-браузер показувати небажану рекламу та робити пошукові редіректи
Pirrit	Adware	Рекламне ПЗ, яке створює проксі-сервер на інфікованих комп'ютерах Mac і розміщує рекламу на веб-сторінках
SearchBrowserHijacker	PUP	Змушує браузер змінити основну пошукову систему та домашню сторінку
CryptoWalletStealer	Malware	Розширення Chrome, яке викрадає криптовалютні рахунки

Для кількісної оцінки виявлених і внесених в базу сигнатур нових зразків шкідливого ПЗ, які не задетектовані іншими антивірусними фреймворками, скористаємось вбудованими засобами аналітики JIRA. Інструмент дозволяє здійснювати розширений пошук заявок по різним критеріям, використовуючи мову запитів Jira Query Language (JQL). Заявки про зразки, для яких сформовано сигнатуру мають статус “Signature done”. Також існує мітка “Clean_by_VT”, яка додається оркестратором під час статичного аналізу сервісом VirusTotal, якщо зразок не задетектований жодним із існуючих антивірусних фреймворків.

Для пошуку зразків виконаємо наступний запит «project = MA AND issuetype = Sample AND status = “Signature done” AND labels = Clean_by_VT». В результаті отримуємо 772 знайдених звіти. Всього зразків зі статусом “Signature done” 10886. Таким чином, можна стверджувати, що наша система дозволила сформувати сигнатури для приблизно 7% невідомих раніше шкідливих програм для ОС сімейства MacOS. Додавання нових джерел отримання зразків та залучення більшої кількості кваліфікованих вірусних аналітиків в майбутньому дозволять збільшити даний показник.

ВИСНОВКИ

Під час виконання магістерської дисертації було розроблено модель та реалізовано систему по формуванню бази даних сигнатур шкідливого програмного забезпечення.

Розроблена система є ефективним інструментом для роботи вірусних аналітиків, оскільки дозволяє в автоматизованому режимі робити збір та дослідження зразків шкідливого програмного забезпечення та подавати результати дослідження в зручному вигляді. На основі сформованих звітів щодо аналізу зразків аналітики мають можливість швидко й ефективно створювати сигнатури вірусів та потенційно небажаних програм. Отримані сигнатури зберігаються в базі даних сигнатур, що є частиною системи, та в подальшому можуть використовуватися антивірусним модулем для детекції шкідливих програм.

Перш ніж розпочати проектування системи, був здійснений детальний аналіз типів шкідливого ПЗ, розглянуто особливості функціонування та способи поширення кожного з типів та проведена класифікація за різними ознаками. Огляд методів виявлення та аналізу шкідливого програмного забезпечення дозволив виділити переваги та недоліки основних підходів, що застосовуються в антивірусній сфері та врахувати їх при проектуванні системи.

Також був здійснений огляд і порівняння існуючих засобів для автоматизованого аналізу шкідливого ПЗ. Виділення позитивних і негативних особливостей кожного з рішень дозволив обрати для нашої системи програмну платформу Cuckoo Sandbox, на основі якої буде здійснюватися аналіз зразків. Обрана система має наступні переваги: відкритий код і модульність, що дозволяє реалізувати додаткові модулі аналізу зразків; підтримка багатьох файлових форматів; наявність добре документованого API.

Далі були сформовані функціональні і нефункціональні вимоги до системи у відповідності до бізнес-потреб та з врахуванням ситуації на ринку. Аналіз переваг і недоліків схожих систем допоміг у формуванні вимог до нашої системи.

Основними ролями, що задіяні в системі визначено вірусного аналітика та адміністратора. Для кожної з них розроблено сценарії використання, що дозволяє зрозуміти, як користувачі можуть взаємодіяти з окремими компонентами системи і які функції можуть виконувати.

В процесі розробки структури було виділено основні компоненти, на базі яких розгортається система: підсистема зберігання зразків, підсистема автоматичного аналізу зразків, модуль генерації сигнатур, підсистема управління сигнатурами та база даних сигнатур.

Для впровадження та реалізації функціоналу системи були проаналізовані сучасні хмарні технології та інші технологічні рішення, що дозволять зробити використання системи більш зручним та ефективним. Використання алгоритму нечіткого хешування SSDeep допомагає виявити та кластеризувати подібні зразки, а інструмент YARA створити ефективні правила, які допоможуть виявити видозмінені варіанти шкідливих програм.

Визначивши основні технології було здійснено розгортання компонентів системи на хмарній платформі AWS. Окремі компоненти системи, зокрема підсистема автоматичного аналізу зразків розгорнуті у віртуальному середовищі VMware ESXi.

Для оцінки ефективності системи був проведений аналіз сигнатур зібраних шкідливих програм. Результати використання системи дозволяють підвищити рівень антивірусного захисту, зокрема у виявленні нових зразків рекламного програмного забезпечення та потенційно небажаних програм для пристроїв на базі MacOS, та тим самим збільшити комерційну привабливість антивірусного продукту. Подальшим напрямком розвитку системи є залучення нових джерел для пошуку зразків, впровадження підтримки нових операційних систем та файлових форматів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Malware Statistics & Trends Report. [Електронний ресурс] – Режим доступу: https://www.av-test.org/en/statistics/malware/ (дата звернення 13.03.2023)

2. Koret J. The Antivirus Hacker's Handbook / J. Koret, E. Bachaalany. – Wiley, 2015. – 77-85 с.

3. What is Malware? [Електронний ресурс] – Режим доступу: https://www.trellix.com/en-us/security-awareness/ransomware/what-is-malware.html (дата звернення 16.03.2023)

4. What Is a Computer Virus? | Computer Virus Definition | Avast. [Електронний ресурс] – Режим доступу: https://www.avast.com/c-computer-virus (дата звернення 16.03.2023)

5. Топ-10 комп'ютерних вірусів, які призвели до величезних збитків. [Електронний ресурс] – Режим доступу: https://10guards.com/ua/articles/10-worst-computer-viruses-in-history/ (дата звернення 17.03.2023)

6. Differences Between Virus and Malware | Virus vs Malware. [Електронний ресурс] – Режим доступу: https://byjus.com/free-ias-prep/difference-between-virus-and-malware/ (дата звернення 17.03.2023)

7. Skoudis E. Malware: Fighting Malicious Code / E. Skoudis, Z. Zeltser. – Prentice Hall Professional, 2004. – 52-353 с.

8. What is Backdoor Attack? Examples and Prevention Malware. [Електронний ресурс] – Режим доступу: https://www.wallarm.com/what/what-is-a-backdoor-attack (дата звернення 18.03.2023)

9. Історія вірусу NotPetya: чи варто остерігатися подібних кібератак в майбутньому? [Електронний ресурс] – Режим доступу: https://www.imena.ua/blog/notpetya-cyberattack/ (дата звернення 18.03.2023)

10. Aycock J. Spyware and Adware / John Aycock. – Springer, 2012. – 146 с.

11. Cryptocurrency is fueling the ransomware boom. Here's how to protect yourself? [Електронний ресурс] – Режим доступу:

https://venturebeat.com/security/cryptocurrency-is-fueling-the-ransomware-boom-heres-how-to-protect-yourself/ (дата звернення 19.03.2023)

12. Aycock J. Computer Viruses and Malware / John Aycock. – Springer, 2010. – 11-80 с.

13. YARA Rules Guide: Learning this Malware Research Tool. [Електронний ресурс] – Режим доступу: https://www.varonis.com/blog/yara-rules (дата звернення 23.03.2023)

14. Classification of malware based on integrated static and dynamic features / I.Rafiqul, T. Ronghua, L. Batten, S. Versteeg. // Journal of Network and Computer Applications. – 2013. – №36. – С. 646–656.

15. Casper Malware: After Babar and Bunny, Another Espionage Cartoon. [Електронний ресурс] – Режим доступу: https://www.welivesecurity.com/2015/03/05/casper-malware-babar-bunny-another-espionage-cartoon/ (дата звернення 23.03.2023)

16. Welcome to YARA’s documentation. [Електронний ресурс] – Режим доступу: https://yara.readthedocs.io/en/v3.4.0/index.html (дата звернення 23.03.2023)

17. Yara Rules Project · GitHub. [Електронний ресурс] – Режим доступу: https://github.com/Yara-Rules (дата звернення 23.03.2023)

18. Robin S. An Introduction to Malware / Sharp Robin. // Technical University of Denmark. – 2007. – С. 21.

19. What is Heuristic Analysis? [Електронний ресурс] – Режим доступу: https://usa.kaspersky.com/resource-center/definitions/heuristic-analysis (дата звернення 25.03.2023)

20. Евристичний аналіз: переваги та недоліки методу. [Електронний ресурс] – Режим доступу: https://antivirus.co.ua/evristicheskij-analiz/ (дата звернення 25.03.2023)

21. Malware Analysis Overview. [Електронний ресурс] – Режим доступу: https://www.n-able.com/blog/malware-analysis-steps (дата звернення 23.03.2023)

22. Malware Analysis Techniques — Basic Static Analysis. [Електронний ресурс] – Режим доступу: https://nasbench.medium.com/malware-analysis-techniques-basic-static-analysis-335a7286a176 (дата звернення 25.03.2023)

23. Sikorski M. Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software / M. Sikorski, A. Honig. – No Starch Press, 2012. – с. 9-205

24. What Is a File Signature? — Definition by ThreatDotMedia. [Електронний ресурс] – Режим доступу: https://threat.media/definition/what-is-a-file-signature/ (дата звернення 26.03.2023)

25. VirusTotal. Services Overview. [Електронний ресурс] – Режим доступу: https://www.virustotal.com/gui/services-overview (дата звернення 26.03.2023)

26. Nasi E. Bypass Antivirus Dynamic Analysis / Emeric Nasi. – Sevagas, 2014. – С. 21.

27. What is Cuckoo? — Cuckoo Sandbox v2.0.7 Book. [Електронний ресурс] – Режим доступу: https://cuckoo.readthedocs.io/en/latest/introduction/what/ (дата звернення 29.03.2023)

28. Cuckoo Sandbox Overview. [Електронний ресурс] – Режим доступу: https://www.varonis.com/blog/cuckoo-sandbox (дата звернення 29.03.2023)

29. Deep Malware Analysis - Joe Sandbox Technology. [Електронний ресурс] – Режим доступу: https://www.joesecurity.org/joe-sandbox-technology (дата звернення 29.03.2023)

30. Malware Analysis - Advanced Threat Detection Platform | VMRay. [Електронний ресурс] – Режим доступу: https://www.vmray.com/why-vmray-advanced-threat-detection-and-analysis-platform/technology/ (дата звернення 29.03.2023)

31. Malware Sample Sources — New & Maintained. [Електронний ресурс] – Режим доступу: https://infosecwriteups.com/malware-sample-sources-a3c7f306adea (дата звернення 02.04.2023)

32. How Cloud Computing Can Benefit Your Small Business. [Електронний ресурс] – Режим доступу: https://www.businessnewsdaily.com/4427-cloud-computing-small-business.html (дата звернення 02.04.2023)

33. AWS Global Infrastructure. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/ru/about-aws/global-infrastructure/ (дата звернення 02.04.2023)

34. Understanding AWS and how it works from the perspective of a project manager. [Електронний ресурс] – Режим доступу: https://medium.com/@NALSengineering/understanding-aws-and-how-it-works-from-the-perspective-of-a-project-manager-658c545b318a (дата звернення 02.04.2023)

35. Why Google Cloud. [Електронний ресурс] – Режим доступу: https://cloud.google.com/why-google-cloud (дата звернення 03.04.2023)

36. Introduction to Google Cloud Platform. [Електронний ресурс] – Режим доступу: https://maelfabien.github.io/bigdata/gcps_1 (дата звернення 03.04.2023)

37. Коваленко А. А. ПОРІВНЯЛЬНИЙ АНАЛІЗ ОРГАНІЗАЦІЇ ХМАРНОЇ ІНФРАСТРУКТУРИ / А. А. Коваленко, О. С. Ляшенко, Р. О. Ярошевич. // Advanced Information Systems. – 2021. – №5. – С. 108–113.

38. Lechtik M. Inside Cuckoo Sandbox: A view into Cuckoo’s internals and emulation flow / Mark Lechtik. // eForensics Magazine. – 2019. – №3. – С. 30–42.

39. Hsu T. Hands-On Security in DevOps / Tony Hsu, – Packt, 2018. – С. 444–460.

40. How to Get Started with YARA for Malware Analysis. [Електронний ресурс] – Режим доступу: https://www.section.io/engineering-education/getting-started-with-yara-for-malware-analysis/ (дата звернення 04.04.2023)

41. How to Identify Malware Similarities with Fuzzy Hashing. [Електронний ресурс] – Режим доступу: https://any.run/cybersecurity-blog/how-to-identify-malware-similarities/ (дата звернення 04.04.2023)

42. Context Triggered Piecewise Hashing To Detect Malware Similarity. [Електронний ресурс] – Режим доступу: https://labs.nettitude.com/blog/context-

triggered-piecewise-hashing-to-detect-malware-similarity/ (дата звернення 05.04.2023)

43. Єремізін О. ПЕРСПЕКТИВИ ВИКОРИСТАННЯ НЕЧІТКОГО ХЕШУВАННЯ В АНТИВІРУСНОМУ ЗАХИСТІ / О. Єремізін, І. Стьопочкіна. // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. – 2016. – №1. – С. 80–84.

44. Kornblum, J. D. Identifying almost identical files using context triggered piecewise hashing / Jesse Kornblum, – Digital Investigation. – 2006. – №3. – С. 30–42.

45. Portnoy M. Virtualization Essentials / Matthew Portnoy. – Sybex, 2016. – С. 19–35.

46. Amazon S3 Security and Access Management. [Електронний ресурс] – Режим доступу: https://aws.amazon.com/s3/security/ (дата звернення 07.04.2023)

47. AWS DynamoDB: Non-Relational Database. [Електронний ресурс] – Режим доступу: https://awsnewbies.com/dynamodb/ (дата звернення 07.04.2023)

48. Universal binaries & Mach-O Format. [Електронний ресурс] – Режим доступу: https://book.hacktricks.xyz/macos-hardening/macos-security-and-privilege-escalation/mac-os-architecture/universal-binaries-and-mach-o-format (дата звернення 07.04.2023)

49. Understanding Model View Controller in ASP.NET MVC. [Електронний ресурс] – Режим доступу: https://www.dotnettricks.com/learn/mvc/understanding-model-view-controller-in-aspnet-mvc (дата звернення 07.04.2023)

50. Cottrell N. MongoDB Topology Design: Scalability, Security, and Compliance on a Global Scale / Nicholas Cottrell. Apress, – 2020. – 288 с.

51. 2020 State of Malware Report. [Електронний ресурс] – Режим доступу: https://www.malwarebytes.com/resources/files/2020/02/2020_state-of-malware-report.pdf (дата звернення 07.04.2023)

ДОДАТОК А. Графічний матеріал