Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

(повна назва)

Кафедра Безпеки інформаційних технологій

(повна назва)

# АТЕСТАЦІЙНА РОБОТА

## Пояснювальна записка

рівень вищої освіти другий (магістерський)

Програмний комплекс для аналізу та виявлення шкідливого програмного забезпечення

(тема)

Виконала: Шандула В.О.

(прізвище, ініціали)

студентка 2 курсу, групи БДІРм-19-1

Спеціальність 125 Кібербезпека

(код і повна назва спеціальності)

Тип програми освітньо-професійна

(освітньо-професійна або освітньо-наукова)

Освітня програма «Безпека державних інформаційних ресурсів»

(повна назва освітньої програми)

Керівник доц. Федюшин О.І.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.

(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)
Кафедра Безпеки інформаційних технологій
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 125 Кібербезпека
(код і повна назва)
Тип програми освітньо-професійна
(освітньо-професійна, або освітньо-наукова)
Освітня програма «Безпека державних інформаційних ресурсів»
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

**ЗАВДАННЯ**
НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Шандулі Вікторії Олександрівні*
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмний комплекс для аналізу та виявлення шкідливого програмного забезпечення*
затверджена наказом по університету від «22» жовтня 2020 р. № 1413Ст

2. Термін подання студентом роботи до екзаменаційної комісії 22 грудня 2020 р.

3. Вихідні дані до роботи:
Теоретичні відомості щодо функціонування шкідливого програмного забезпечення та методів аналізу та виявлення використовуваних на практиці; програмні засоби, що використовуються для виявлення підозрілої активності; літературні джерела, щодо ефективності використання різних методів аналізу потенційно шкідливого програмного забезпечення; мова програмування для реалізації продукту Python.

4. Перелік питань, що потрібно опрацювати в роботі
1. Провести аналіз функціонування шкідливого програмного забезпечення.
2. Дослідити основні властивості і структуру PE файлів.
3. Розглянути методи статичного аналізу з метою виявлення ознак шкідливого ПЗ.
4. Розглянути методи динамічного аналізу з метою виявлення ознак шкідливого ПЗ.
5. Розглянути програмні засоби, що використовуються для виявлення підозрілої активності.
6. Розробити архітектуру та наповнення програмного комплексу.
7. Зробити висновки за результатами тестування та досліджень.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій *Презентаційний матеріал у вигляді слайдів*

## КАЛЕНДАРНИЙ ПЛАН

| № | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання | 01.09.2020 | виконано |
| 2 | Збір та аналіз літературних джерел за темою атестаційної роботи | 02.09.2020-20.09.2020 | виконано |
| 3 | Аналіз функціонування шкідливого програмного забезпечення | 21.09.2020-04.10.2020 | виконано |
| 4 | Дослідження основних властивостей і структури PE файлів | 05.10.2020-18.10.2020 | виконано |
| 5 | Розгляд методів та інструментів статичного аналізу з метою виявлення ознак шкідливого ПЗ | 19.10.2020-01.11.2020 | виконано |
| 6 | Розгляд методів та інструментів динамічного аналізу з метою виявлення ознак шкідливого ПЗ | 02.11.2020-15.11.2020 | виконано |
| 7 | Розробка архітектури та наповнення програмного комплексу | 16.11.2020-29.11.2020 | виконано |
| 8 | Тестування результатів роботи, обробка результатів досліджень | 30.11.2020-09.12.2020 | виконано |
| 9 | Оформлення пояснювальної записки та матеріалів презентації | 10.12.2020-21.12.2020 | виконано |
| 10 | Представлення роботи до захисту | 22.12.2020 | виконано |

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ _____доц. Федюшин О.І._____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ

Пояснювальна записка атестаційної роботи містить: 96 с., 12 рис., 2 дод., 39 джерел.

РЕ-ФАЙЛ, N-ГРАМ, ГРАФ, БІНАРНА СТРУКТУРА, СТАТИЧНИЙ АНАЛІЗ, МЕТОДИ СТАТИЧНОГО АНАЛІЗУ, МЕТОДИ ДИНАМІЧНОГО АНАЛІЗУ, ОБФУСКАЦІЯ, МОНІТОРИНГ СТАНУ ОПЕРАЦІЙНОЇ СИСТЕМИ

Об'єкт дослідження – процес аналізу та виявлення шкідливого програмного забезпечення.

Предмет дослідження – методи детектування та аналізу шкідливого програмного забезпечення.

Метою атестаційної роботи є розробка програмного комплексу для аналізу та виявлення шкідливого програмного забезпечення.

Методи дослідження – методи експерименту, графові, статистичні методи, методи сигнатурного та евристичного аналізу, n-грами.

В атестаційній роботі розроблений програмний комплекс для аналізу та виявлення шкідливого програмного забезпечення. Процес детектування потенційно вірусного програмного забезпечення здійснюється за допомогою аналізу структури файла, зокрема елементів формату PE-файлу, та методів статичного та динамічного аналізу поведінки програмного забезпечення. Ці види аналізу реалізовано за допомогою окремих модулів на мові Python.

Даний програмний комплекс може бути використаний при швидкому або поглибленому аналізі файлів на предмет безпечного використання, або як окремий компонент антивірусного програмного забезпечення.

# ABSTRACT

Master`s thesis: 96 pages, 12 figures, 2 appendices, 39 sources.

PE-FILE, PORTATIVE EXECUTED FILE, N-GRAME, GRAPH, BINARY STRUCTURE, STATICAL ANALISIS, METHODS OF STATICAL ANALYSIS, METHODS OF DYNAMIC ANALYSIS, OBFUSCATION, MONITORING OF THE STATE OF THE OPERATING SYSTEM

The object of the study is the process of analysis and detection of malicious software.

Subject of research - methods of detection and analysis of malicious software.

The major goal of this thesis is to develop a software package for analysis and detection of malicious software.

Methods of research experimental methods, graph, statistical methods, methods of signature and heuristic analysis, n-grams.

In the certification work the software complex for the analysis and detection of malicious software is developed. The process of detecting potentially viral software is performed by analyzing the structure of the file, in particular elements of the PE file format, and methods of static and dynamic analysis of software behavior. These types of analysis are implemented using separate modules in Python.

This software can be used for quick or in-depth analysis of files for safe use, or as a separate component of anti-virus software.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CFG – граф потоку керування.

Overlay – частина PE файлу, що знаходиться за кінцем останньої секції.

PE – портативний виконуваний файл.

Деобфускація – ропакування секцій PE-файлу.

Ланцюг Маркова – послідовність станів з головним правилом: в кожний момент часу система Може перебувати тільки в одноМу стані.

Обфускація – пакування секцій PE-файлу.

# ВСТУП

Сьогодні зі зростанням впливу та ролі комп'ютерів у сферах, що пов’язані з наукою, економікою, військовими справами, культурним життям соціуму та звичайного побуту, загроза кіберпорушень з наступним отриманням збитку різного розміру через надходження зловмисного програмного забезпечення (ПЗ) дедалі росте. Не кажучи про те, що цей збиток може бути руйнівним для, наприклад, цілого підприємства. Кіберпорушники навчилися не тільки перехоплювати дані, красти майно інтелектуальної власності, блокувати доступ до важливої пам'яті комп'ютера, або ж до окремих її частин, ставати на заваді нормального фукнціонування комп’ютера, або перетворювати його на недієздатне залізо, але і красти обчислювальні потужності, використовуючи їх для своїх цілей.

Шкідливе ПЗ утворюється з зростаючою швидкістю та складністю. Їх все важче детектувати. Зростає ймовірність помилитися, при пошуку шкідливого ПЗ, видалити корисний "чистий" файл, нашкодити самому собі. Деякі спеціалісти пропонують користуватись ліцензованим ПЗ для виявлення вірусного ПЗ, проте варто пам'ятати, що і вони не універсальні, адже ґрунтуються на базах сигнатур, які додаються з часом і не можуть бути довільними, оскільки між появою вірусу з новою сігнатурою та додаванням її в базу даних, повинен пройти час, за який спеціалісти досліджують це сімейство вірусів, іноді навіть помилково приймають рішення про невіднесення його сигнатури. Все це накладає свої складності на проблему пошуку загроз. Ще одним ускладнюючим фактором стає здатність вірусів до зміни власного коду в процесі виконання, і не тільки.

Проте, відомо, що портативні виконувані файли є найбільш поширеним джерелом загроз вірусного зараження. Люди часто завантажують ПЗ з ненадійних сайтів, або стають жертвами обману. Після чого вже не можуть згадати, коли саме вірус міг потрапити на комп'ютер. Тому рекомендується

завжди перевіряти, що саме було завантажено.

Дана робота присвячена методам перевірки таких файлів, витягу інформаціїї з PE-header, аналізу інструкцій виконання. Для автоматизації процесу перевірки буде побудований програмний комплекс. В процесі роботи над проектом були розглянуті основні типи вірусів та методи їх детектування, принципи роботи аналізаторів та класифікаторів, це дозволило обрати найбільш ефективні інструменти аналізу та організувати роботу комплексу з метою виявлення шкідливого ПЗ.

# 1 ШКІДЛИВЕ ПРОГРАМНЕ ЗАБЕЗЧЕННЯ

## 1.1 Поняття шкідливості файла

Сьогодні комп'ютерні програми і додатки розробляються з великою швидкістю. З розвитком комп'ютерної техніки та інтернету, все більше людей починають турбуватися про їхню безпеку та безпеку своїх даних. Розвиток шкідливого програмного забезпечення призводить до розвитку більш сучасних механізмів захисту. За 2017 рік світова громадськість познайомилася з WannaCry, Petya, NotPetya. Аналіз такої кількості даних вимагає від антивірусних компаній все зростаючі зусилля. Віруси стає все більш складно виявити. Сучасне шкідливе ПЗ здатне як впроваджуватися і заражати чисті файли системи і інших програм, так і самостійно поширюватися і міняти своє тіло виконання в процесі життєдіяльності. Застосовуються різні засоби приховуввання шкідливого коду. Використовуються інструменти шифрування секцій, коду, даних. Шкідливе ПО може не тільки вкрасти або пошкодити дані користувача, вимагати гроші, уповільнити роботу комп'ютера, але і перетворити комп'ютер користувача в шпигуна, заволодіти його управлінням.

Традиційний підхід до виявлення шкідливих програм заснований на зіставленні сигнатур досліджуваних файлів [1]. Процедура полягає в наступному:

– новий вірус / шкідливе ПЗ починає поширюватися;

– експерти антивірусних компаній отримують зразки для дослідження поведінки вірусу;

– експерти привласнюють вірусу унікальну сигнатуру, що представляє собою послідовність інструкцій;

– сигнатура додається в базу даних сигнатур шкідливих програм;

– клієнти повідомляються про оновлення бази сигнатур;

– клієнти оновлюють їх бази сигнатур, таким чином стаючи

захищеними від даного виду шкідливого ПЗ.

Варто відзначити, що сигнатури дозволяють гарантовано визначити тип вірусу. Це дозволяє додатково вносити в базу сигнатур методи лікування вражених файлів, вірусів. Цей простий підхід володіє де-якими недоліками:

– можливість захищати клієнта тільки від відомих вірусів. Не можливо отримати сигнатуру абсолютно нового вірусу, не дослідивши його. Тут варто обмовитися, що сигнатури, як правило, створюються так, щоб покривати не один, а якомога більшу кількість, сімейств вірусів. Однак завжди існує така зміна тіла вірусу, при якому сигнатура перестає виявлятися:

– постійне зростання бази даних сигнатур. Зі збільшенням кількості вірусів, їх родин, а також здатності вірусів змінюватися збільшується і швидкість наповнення бази;

– при появі вірусу і до оновлення бази сигнатур, клієнт вразливий для нової шкідливої програми.

## 1.2 Класифікація шкідливих файлів

В ході вивчення шкідливої програми, коли ми стикаємось з зразком шкідливого ПО, ми повинні віднести даний зразок до конкретного сімейства шкідливих програм, щоб краще розуміти характеристики та функціонал, які відповідають раніше проаналізованим зразками. Порівняння підозрілого файлу з раніше проаналізованими зразками або зразками, що зберігаються в публічному або приватному сховище, може дати уявлення про сімейство шкідливих програм та їх характеристик. Так фахівець розумітиме, на що звертати увагу при відновленні роботи комп’ютера. Хоча криптографічні хеш-функції (MD5 / SHA1 / SHA256) є відмінним методом для виявлення ідентичних зразків, вони не допомагають в ідентифікації подібних зразків. Дуже часто автори шкідливих програм змінюють дрібні аспекти шкідливих програм, що повністю змінює значення хеш-функції. У наступних розділах

описано ряд методів, які можуть допомогти вам в порівнянні та класифікації підозрілого файлу.

Але перед цим, для розуміння побудови шкідливих програм, розглянемо трохи докладніше їх функціонали. Залежно від своєї поведінки вони діляться на три основних класу - трояни (Trojan), віруси (Virus) і черв’яки (Worm).

1.2.1 Трояни

Це резидентні шкідливі виконувані файли, що функціонують в межах системи. Потрапивши в систему, вони розташовуються в ній і починають виконувати своє призначення.

Найчастіше вони призначені для крадіжки різноманітних паролів, надання зловмисникам віддаленого доступу до заражених систем для завантаження інших шкідливих файлів з Інтернету.

Цей вид шкідливих програм самостійно не може розповсюджуватись. Найчастіше трояни потрапляють в систему за допомогою черв'яків, про яких буде розказано нижче, або завантажуються на комп'ютер іншими троянами.

Залежно від специфіки дій відбувається розподіл троянів на підкласи - Trojan-Spy (шпигунські програми), Trojan-PSW (програми, які крадуть паролі), Trojan-Downloader (програми, які завантажили з Інтернету інші шкідливі файли).

Окремо варто відзначити підклас троянів Backdoor. Це трояни, що дозволяють зловмисникові отримати повний або частковий контроль над зараженою системою за допомогою віддаленого доступу. Backdoor вважаються найбільш небезпечним видом троянів. Крім несанкціонованого доступу до системи, зараженої даними видами троянів, існує ще одне застосування цих шкідливих файлів. Уявімо, що зловмисникові вдалося заразити велику кількість комп'ютерів. (А число може бути дійсно величезним. Наприклад, нещодавно гучний мережний черв'як Net-Worm.Win32.Kido заразив близько десяти мільйонів комп'ютерів). Це означає, що він має в своєму розпорядженні гігантські обчислювальні ресурси. Мережу заражених комп'ютерів прийнято називати ботнетом (botnet) або зомбі-мережею. За допомогою ботнетів зазвичай

здійснюються атаки сайтів і сервісів, кінцева мета яких - шантаж власників ресурсу, який знаходиться під атакою.

Також існує не менш небезпечний підклас Rootkit (руткіт). У нього входять, як правило, драйвери, що глибоко впроваджуються в систему. Завдяки використанню функцій рівня ядра, а також доступу до системних структур, цей вид вірусів особливо складно виявити. Зазвичай руткіти призначені для приховування файлів або процесів, присутніх в системі. Шкідливий драйвер здатний перехоплювати всі системні виклики і перенаправляти їх на свій код.

1.2.2 Черв’яки

Черв’яки - шкідливі виконувані файли, що використовують для поширення мережу та уразливості. Уразливість - та чи інша недосконаліть системи, яку можна використовувати для виконання несанкціонованих дій.

Певного призначення у черв'яків немає. Вони можуть переносити троянів, так само як трояни красти паролі і надавати віддалений доступ до системи.

Для свого поширення черв'яки використовують комп'ютерні мережі.

Види черв'яків. Залежно від способів розмноження черв’яки діляться на Email-Worm (поширення за допомогою email), P2P-Worm (поширення за допомогою p2p мереж), IM-Worm (за допомогою IM клієнтів), IRC-Worm (з допомогою IRC, MIRC і інших каналів) і Net-Worm (за допомогою вразливостей систем).

Найнебезпечнішим з видів черв'яків є мережні черв’яки Net-Worm. Вони розповсюджуються, використовуючи "дірки" в системах. Серед відомих представників цього підкласу черв'яків можна назвати Net-Worm.Win32.Slammer, що розповсюджувався за допомогою помилки в програмному забезпеченні MS SQL. Net-Worm.Win32.Kido, згаданий вище, також є представником підкласу мережних черв'яків.

1.2.3 Віруси

В побуті прийнято все шкідливе ПЗ називати вірусами, проте це лише один з видів ПЗ. А саме віруси - шкідливі програми, що заражають файли, що

знаходяться в системі. В даному випадку мова йде про PE віруси - віруси, що заражають виконувані файли.

Як і в випадку черв'яків функціонал може бути різним.

Віруси розповсюджуються шляхом зараження інших виконуваних файлів. Потрапивши в систему, вірус шукає файли, що йому підходять. Потім він їх заражає. Зараження відбувається по-різному, в залежності від виду вірусу. При виконанні користувачем зараженого файлу разом з оригінальним кодом файлу виконується і код вірусу, користувач при цьому ні про що не підозрює. Таким чином, поки користувач грає, наприклад, в заражену косинку, код вірусу заражає такі файли. Такий вид вірусів іменується інфекторами (infector). Існують також віруси-компаньйони, що копіюють себе в папки під ім'ям існуючих в цих папках файлів. Оригінальні файли віруси перейменовують і приховують. При запуску вірусу-компаньйона спочатку відбувається виконання шкідливого файлу. Після чого вірус запускає перейменований їм файл. Таким чином користувач навіть не підозрює про зараження системи.

Віруси - одні з найбільш шкідливих та складних до виявлення програм. Оскільки віруси заражають крім усього іншого і системні файли, видалення заражених файлів неможливе - їх треба лікувати. Виявити заражений файл вкрай важко, оскільки в більшості випадків код вірусу малий у порівнянні з кодом зараженої програми.

## 1.3 Мета аналізу та детектування шкідливого ПЗ

Аналіз шкідливих програм - це вивчення поведінки шкідливого ПЗ. Мета аналізу шкідливого ПЗ - зрозуміти роботу шкідливих програм і методи їх виявлення та усунення. Він включає в себе аналіз підозрілого виконуваного файлу в безпечному середовищі для визначення його характеристик і функціональних можливостей, щоб можна було збудувати кращу оборонну стратегію для захисту мережі організації.

Основним мотивом проведення аналізу шкідливих програм є

отримання інформації з зразка шкідливого ПЗ, яка може допомогти в реагуванні на шкідливий інцидент. Метою аналізу шкідливих програм є визначення можливостей шкідливого ПЗ, його виявлення та зміст. Це також допомагає у визначенні ідентифікованих моделей, які можуть бути використані для лікування і запобігання майбутніх інфекцій. Ось деякі з причин, чому ви будете виконувати аналіз шкідливих програм:

- щоб визначити характер і призначення шкідливого ПЗ. Наприклад, це може допомогти визначити, чи є шкідливе ПЗ засобом для крадіжки інформації, HTTP-ботом, спам-ботом, руткітом, кілоггерів або RAT і т.д.;
- щоб отримати уявлення про те, як система була зламана і які наслідки;
- для виявлення мережних індикаторів, пов'язаних з шкідливим ПЗ, які можуть потім бути використані для виявлення аналогічних інфекцій за допомогою моніторингу мережі. Наприклад, під час аналізу, якщо ви визначите, що шкідлива програма зв'язується з конкретним доменним ім’ям / IP-адресою, ви можете використовувати це доменне ім’я / IP-адресу для створення підпису і відстежувати мережний трафік для ідентифікації всіх хостів, зв'язавшись з цим доменним ім’ям/ IP-адресою;
- щоб витягти індикатори хостів, такі як імена файлів і ключі реєстру, які, в свою чергу, можуть бути використані для визначення аналогічної інфекції з використанням хостового моніторингу. Наприклад, якщо ви дізнаєтеся, що шкідлива програма створює розділ реєстру, ви можете використовувати цей ключ реєстру як індикатор для створення підпису або сканування вашої мережі, щоб визначити хости, які мають однаковий розділ реєстру;
- визначити намір і мотив зловмисника. Наприклад, під час вашого аналізу, якщо ви виявите, що шкідливе ПЗ краде банківські облікові дані, то можна зробити висновок, що мотив зловмисника - грошова вигода.

Тільки визначивши досліджуваний файл, як вірус, можна отримати його сигнатуру і додати в базу. Більш того, розробники вірусів навчилися успішно обходити пошук сигнатур, використовуючи обфускацію коду тіла вірусу. Поліморфні і метаморфічні шкідливі програми змінюють свій зовнішній вигляд в процесі життєдіяльності. Все це спонукало антивірусні компанії розробляти альтернативні методи. А саме, можна виділити два великих напрямки досліджень [2]:

– статичний аналіз (аналіз структури бінарного файлу, його атрибутів, логічних структур, потоку виконання і даних);
– динамічний аналіз (відстеження дій програми при виконанні, побудова її профілю).

Кожен з методів має свої переваги і недоліки. Так, як правило, для кращого детектування шкідливих програм вони використовуються одночасно. У кожному з цих методів існує ймовірність помилково виділити наявність у файлі вірусу, коли насправді файл чистий. У таких випадках передбачуване лікування може зіпсувати файл, що спричинить до втрати інформації.

Динамічний аналіз дозволяє обійти обфускацію бінарного файлу. Так, наприклад, автори вірусів широко використовують системи упаковки, шифрування коду і даних, обфускацію функцій і потоку управління. Але ті ж методи використовуються і для створення додатків розробниками, щоб захистити інтелектуальну власність, ускладнити реверс інжиніринг.

Метод виділяє кілька основних дій, таких, як видалення файлу, запис в файл, спілкування з мережею, відкриття порту на прослуховування, розсилка листів та інші. Профіль цього файлу, його діяльності вивчається експертом або методами машинного навчання для винесення вердикту про шкідливість зразка.

Проте, динамічний аналіз можливий лише при виконанні коду, що робить операційну систему більш вразливою, а також, де-які віруси можуть аналізувати інше працююче ПЗ і маскувати себе, і, тим самим, поводяться по-різному в тестовому і робочому оточенні. Таким чином, потрібно створювати

максимально схожі оточення, але це бачиться ще однією проблемою, яка потребує вирішення.

Статичний аналіз здатний доповнювати динамічний, надаючи інформацію про атрибути бінарного файлу. Статичний метод аналізує програму до виконання, витягує атрибути з бінарного файлу, підраховує статистики і на основі цієї інформації виносить вердикт про загрозу цього файлу. Такий підхід безпечний – вердикт виноситься ще до виконання файлу, але погано працює на файлах з обфускацією, запакованими секціями. Також, зі збільшенням розміру файлу, час, необхідний для аналізу, збільшується. Крім цього, для розробки якісного статичного аналізатора, потрібно розуміти роботу завантажувача бінарного файлу, різницю між документацією і дійсною поведінкою завантажувача. Віруси можуть використовувати частину полів бінарного файлу для своїх потреб. Наприклад, у вигляді зберігання даних або адреси виконання шкідливого коду.

# 2 СТАТИЧНИЙ АНАЛІЗ

## 2.1 Статичний аналіз заголовку

Надалі розлянемо методи статичного тестування та методи прийняття рішень за статичним аналізом. Перший пункт відноситься до перевірок, які застосовуються до коду програми або заголовку. Другий пункт накопичує дані отримані в ході тестів та, використовуючи евристики, приймає рішення стосовно питання «чи є цей файл шкідливим».

Більшість методів статичного тестування грунтуються на добуванні структур PE (Portable Executable) файлу, як в деревовидному вигляді, так і у вигляді сукупності полів. Використовуються як ознаки, одержувані з машинного коду, викликів функцій, так і ознаки, одержувані з розгляду файлу в вигляді послідовності байт, визначення статистик, ентропій, підрахунок n-грам. Проте є і такі, що грунтуються на геш-сумі, а бо цифровому відбитку.

## 2.2 Формат PE

Для аналізу досліджуваного файлу класифікатор отримує інформацію від PE-парсера. Знання про PE (Portable Executable) формат необхідно для оцінки корисності інформації, яка витягується, і виділення найбільш значущих атрибутів.

Далі наводиться короткий опис PE формату, його структури, як для 32 бітних, так і 64 бітних систем.

Повна специфікація доступна на сайті Microsoft [3], проте, в документі є двозначності. Деякі моменти покриті статтею Криса Касперскі [4]. Потрібно розуміти, що специфікація, наведена на сайті Microsoft, має, скоріше, оглядовий характер. Як насправді відбувається завантаження можна спостерігати лише експериментально. В особливо цікавих випадках один і той же файл може бути

запущений одним завантажувачем, але запустивши його іншим, ми можемо привести всю систему до перезапуску. Варто пам'ятати, що завантажувач при запуску бінарного файлу може його змінювати.

«Portable Executable (PE, портативний виконуваний) – формат виконуваних файлів, об'єктного коду та динамічних бібліотек, який використовується в 32-і 64-розрядних версіях операційної системи Microsoft Windows. Формат PE являє собою структуру даних, що містить всю інформацію, необхідну PE-завантажувачу для відображення файлу в пам'ять [5].

Всі PE файли можна розділити на файли з розширенням EXE і DLL. DLL файли призначені для експортування функцій або даних для використання іншими програмами. Тобто, вони зазвичай можуть бути запущені в контексті інших програм. Такі файли мають розширення .dll, .sys, .ocx, .cpl, .fon, .drv [6]. EXE файли запускаються в своїх власних процесах. Вони зазвичай мають розширення .exe і не експортують символи. Незважаючи на такий умовний поділ, формат не забороняє створювати гібриди. Наприклад, EXE файли може експортувати символи.

Структура PE файлу представлена на схемі (рисунок 1.1) . Атрибути цієї структури, а також статистика по послідовностям байт цієї структури використовується для отримання ознак.

Будь-який PE файл починається з MS-DOS Stub'a. Залишений для сумісності і являє собою заглушку. Перші два байта якої повинні бути рівними "MZ". Атрибут e_lfanew, являє собою зсув PE File заголовка щодо початку файлу і вказує на сигнатуру "PEx0x0". Не обов'язково, щоб e_lfanew вказувало на область відразу після MS-DOS Stub, що також відображено на схемі (Рисунок 1.1) – як невикористана область. Секції знаходяться відразу після PE File заголовка.

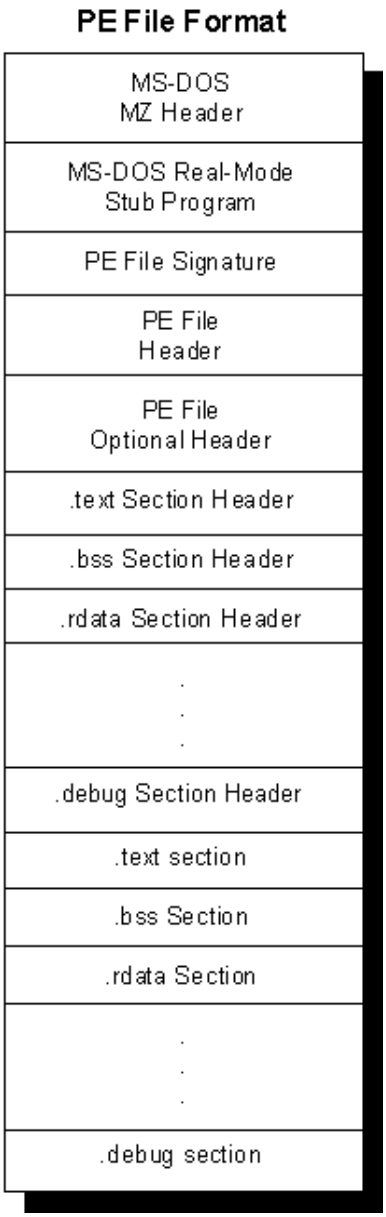


Рисунок 1.1 – Структура PE файлу

Після PE сигнатури "PEx0x0" в PE File Header розташовується COFF File Header. Має фіксований розмір. Містить загальну інформацію про файл, таку як: тип цільової машини (x64, ARM, MIPS і так далі), кількість секцій – NumberOfSections, дата створення файлу, розмір опционального заголовка – SizeOfOptionalHeader. Атрибути всього файлу – Characteristics (саме тут містяться прапори – чи є файл EXE або DLL).

Далі йде опціональний заголовок (Optional Header). Хоч ця структура і має таку назву, її присутність обов'язкова для завантаження файлу. Структура містить уточнюючу інформацію про файл. Таку, наприклад, як 32х або 64х бітове адресний простір. Сумарний розмір секцій коду, ініціалізованих і

неініціалізованих даних (відповідно до [4] ці поля ніким не перевіряються також як і відносні базові адреси кодової секції та секції даних). У деяких антивірусів є евристики на значення адреси точки входу – AddressOfEntryPoint. Передбачається, що точка входу розташовується в першій секції файлу (як правило .text секція), базовий адрес завантаження програми (ImageBase) повинен бути кратний 64кб. Також, для аналізу корисні вирівнювання – FileAlignment, SectionAlignment, а вірніше поля PE файлу, від яких вимагається вирівнювання. Також, в опціональному заголовку міститься каталог даних (DataDirectory), кількість елементів якого – NumberOfRvaAndSizes. В каталозі даних (DataDirectory) кожен елемент – структура, яка містить покажчик та розмір. Кожна із записів має певну роль. Так, IMAGE_DIRECTORY_ENTRY_EXPORT – покажчик на таблицю експорту функцій і даних (зустрічається в основному в DLL). Після опціонального заголовка йде таблиця секцій (Section Table), має NumberOfSections секцій. Як правило, секції мають наступний порядок.

Спочатку описана секція з кодом, після – кілька секцій ініціалізованих даних, а після – секція неініціалізованих даних. Кожна секція має ім'я (не має ніякого значення при завантаженні файлу), адреса початку секції в пам'яті і в файлі (вирівняні), віртуальну і фізичну довжину секції (VirtualSize і SizeOfRawData). Віртуальні адреси секцій повинні йти підряд, не накладаючись і не утворюючи пропусків. Поле Characteristics є права доступу до секції і особливості її завантаження.

Таблиця експорту потрібна для зв'язку експортованих функцій з їх адресами. Містить покажчики на 3 таблиці: таблиця імен, ординалів, адрес. Таблиця імпорту співвідносить виклики функції динамічних бібліотек з їх адресами. Існує 2 режими імпорту: стандартний, що зв'язує (bound import), відкладений (delay import).

Аналізатор повинен швидко працювати навіть на слабких машинах, тому завдання полягає у підборі найцінніших з точки зору визначення шкідливого ПЗ ознак. Тут стають видні і головні проблеми статичного аналізу.

Одна з основних – зашифровані, упаковані секції. Віруси шифрують свій код виконання так, що він більше виявляє загрози з точки зору статичного аналізатора. Для вирішення цієї проблеми або додають дешифратор, або ґрунтують відповідь на статистиці по цій секції. У першому випадку завдання постає окремим дослідженням, яке не буде розглянуто тут. Більшість дешифраторів використовують евристики і перебирають відомі методи шифрування, а є ті, які дешифрують при виконанні бінарного файлу. При підрахунку ентропії ми опираємося на припущення, що якісне шифрування вирівнює частоту байт секцій, тим самим збільшуючи її ентропію.

## 2.3 Ознаки зараження файлу

Далі ми розглянемо, які структури та мітки можуть бути цікавими при статичному аналізі файлу.

Практично всі атрибути структури COFF File варто розглядати при детектуванні вірусів, зараження файлу. Виключимо з розгляду дату створення файлу – вона не повинна впливати на ймовірність зараження, але вибірка може виявитися некоректною за цим параметром.

Далі, в опціональному заголовку є кілька покажчиків: IMAGE_DIRECTORY_ENTRY_SECURITY представляє особливий інтерес, він вказує на Certificate Table. Таблицю, що знаходиться на диску. При IMAGE_DIRECTORY_ENTRY_SECURITY! = 0 практично виключена ймовірність того, що файл є шкідливим. Також, якщо IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR! = 0, то файл представляє собою .NET додаток, що складається з байт-коду, що також знижує ймовірність шкідливості файлу.

Також варто розглянути таблиці секцій, так і їх атрибути. Для детекції вірусів і аналізу також корисні поля таблиці експорту, імпорту.

Для аналізу PE файлу варто аналізувати назви бібліотек і функцій (API). Тут виникає питання про спосіб подання цієї інформації. Вектор ознак повинен

бути фіксований, що накладає певні обмеження. З одного боку ми повинні вибирати ті ознаки часто зустрічаються в API, з іншого – ті, які статистично значущі при відділенні шкідливого ПЗ від чистих файлів. Виклавши загальну картину структур PE файлів, ми можемо приступити до формування атрибутів і їх відбору. Стає видно такі властивості одержуваних ознак. Ознаки поділяються на ролі і значення структури, з якої були вилучені:

- чисельна ознака (кількість секцій, символів, розмір опціонального заголовка, адреса точки входу);
- прапор присутності (секції, арі таблиці імпорту);
- Значення функції від послідовності байт (ентропія секцій).

Тут ми не розглядаємо функції з порівняно довгою послідовністю байт, так, наприклад, ентропію всього файлу, розподіл певних символів в файлі або подання файлу в вигляді зображення.

Також слід приділяти увагу ресурсам, необхідним для виконуваного файлу, такі як значки, меню, діалоги і рядки, що зберігаються в розділі ресурсів (.rsrc) виконуваного файлу. Найчастіше зловмисники зберігають таку інформацію, як додаткові виконавчі файли, документи-обманки і дані про конфігурацію, в розділі ресурсів, тому вивчення ресурсів може дати цінну інформацію про файл. Секція ресурсу також містить інформацію про версії, яка може дати відомості про походження, назві компанії, автора програми і про авторські права.

Просто вивчивши вміст секції ресурсу, можна багато чого довідатися про характеристики шкідливого ПЗ.

## 2.4 Статичний аналіз тіла програми

Статичний аналіз найчастіше застосовують для пошуку вразливостей - ділянок коду, наявність яких може призвести до порушення конфіденційності, цілісності або доступності інформаційної системи. Однак ті ж технології можна

застосовувати для пошуку і інших помилок або особливостей коду, наприклад, ознак шкідливості коду.

В загальному вигляді задача статичного аналізу алгоритмічно нерозв'язна (наприклад, по теоремі Райса). Тому доводиться або обмежувати умови задачі, або допускати неточність в результатах (пропускати уразливості, давати помилкові спрацьовування). Виявляється, що на реальних програмах робочим виявляється другий варіант.

Існує безліч платних і безкоштовних інструментів, які заявляють про пошук вразливостей в додатках, написаних на різних мовах програмування. Розглянемо, як зазвичай влаштований статичний аналізатор. Далі мова піде саме про ядро аналізатора, про алгоритми. Звичайно, інструменти можуть відрізнятися по доброзичливості інтерфейсу, по набору функціональності, по набору плагінів до різних систем і зручності використання API.

У схемі роботи статичного аналізатора можна виділити три основні кроки.

Побудова проміжного представлення (проміжне представлення також називають внутрішнім поданням або моделлю коду).

Застосування алгоритмів статичного аналізу, в результаті роботи яких модель коду доповнюється новою інформацією.

Застосування правил пошуку вразливостей до доповненої моделі коду.

У різних статичних аналізаторах можуть використовуватися різні моделі коду, наприклад, вихідний текст програми, потік лексем, дерево розбору, трьохадресний код, граф потоку керування, байткод - стандартний або власний - і так далі.

Аналогічно компіляторам, лексичний і синтаксичний аналіз застосовуються для побудови внутрішнього подання, найчастіше - дерева розбору (AST, Abstract Syntax Tree). Лексичний аналіз розбиває текст програми на мінімальні смислові елементи, на виході отримуючи потік лексем. Синтаксичний аналіз перевіряє, що потік лексем відповідає граматиці мови програмування, тобто отриманий потік лексем є вірним з точки зору мови. В

результаті синтаксичного аналізу відбувається побудова дерева розбору - структури, яка моделює вихідний текст програми. Далі застосовується семантичний аналіз, він перевіряє виконання більш складних умов, наприклад, відповідність типів даних в інструкціях присвоювання.

Дерево розбору можна використовувати як внутрішнє уявлення. Також з дерева розбору можна отримати інші моделі. Наприклад, можна перевести його в трьохадресних код, за яким, у свою чергу, будується граф потоку керування (CFG). Зазвичай CFG є основною моделлю для алгоритмів статичного аналізу.

2.4.1 Граф

Одним з основних алгоритмів статичного аналізу є аналіз відновленої структури файлу. Завдання такого аналізу – визначити послідовність та схему викликів інструкцій та дані, якими оперує програма на даному етапі. Інформація може бути різна, наприклад, тип даних або значення. Залежно від того, яку інформацію потрібно визначити, можна сформулювати завдання аналізу потоку даних.

В теорії побудови компіляторів описані рішення задачі внутрішнього процедурного аналізу потоку даних (відстежити дані необхідно в рамках однієї процедури / функції / методу). Рішення спираються на теорію алгебраїчних решіток і інші елементи математичних теорій. Вирішити завдання аналізу потоку даних можна за поліноміальний час, тобто за прийнятний для обчислювальних машин час, якщо умови задачі задовольняють умовам теореми про можливості розв'язання, що на практиці відбувається далеко не завжди.

Для постановки конкретного завдання внутрішнього процедурного аналізу, крім визначення інформації, яку шукаємо, потрібно визначити правила зміни цієї інформації при проходженні даних за інструкціями в CFG. Нагадаємо, що вузлами в CFG є базові блоки - набори інструкцій, виконання яких відбувається завжди послідовно, а дугами позначається можлива передача управління між базовими блоками.

Для кожної інструкції визначаються безлічі:

– інформація, породжувана інструкцією;

- інформація, знищувана інструкцією;
- інформація в точці перед інструкцією;
- інформація в точці після інструкції.

Метою аналізу потоку даних є визначення множин і для кожної інструкції програми. Основна система рівнянь, за допомогою якої вирішуються завдання аналізу потоку даних, визначається наступним співвідношенням (рівняння потоку даних):

Друге співвідношення формулює правила, за якими інформація «об'єднується» в точках злиття дуг CFG (- попередники в CFG). Може використовуватися операція об'єднання, перетину і деякі інші.

Алгоритми рішення задач аналізу потоку даних шукають максимальні нерухомі точки. Існує кілька підходів до вирішення: ітеративні алгоритми, аналіз сильно зв'язкових компонент, T1-T2 аналіз, інтервальний аналіз, структурний аналіз і так далі. Існують теореми про коректність вказаних алгоритмів, вони визначають область їх застосування на реальних завданнях.

2.4.2 N-gram

N-грама — послідовність з n елементів. З семантичної точки зору, це може бути послідовність звуків, складів, слів або букв. З іншого боку, n-грама це модель представлення тексту у вигляді ланцюга Маркова. В свою чергу ланцюг Маркова – послідовність елементів, що відповідає правилу: потік на кожному відрізку часу може бути в тільки одному з n станів, при чому, якщо він знаходиться в стані i, то ймовірність того, що він перейде в стан j дорівнює $p_{ij}$. На практиці частіше зустрічається N-грами як ряд слів, стійкі словосполучення називають колокацією. Послідовність з двох послідовних елементів часто називають біграм, послідовність з трьох елементів називається триграма. Не менш чотирьох і вище елементів позначаються як N-грами, N замінюється на кількість послідовних елементів.

Ймовірність послідовності слів можна оцінити як вказано у формулі (1.1)

$$p(w_1^i) = p(w_i|w_1^{i-1})p(w_{i-1}|w_1^{i^{-2}}) \dots p(w_1) \qquad (1.1)$$

Для цих ймовірностей виконується формула (2.2):

$$p\left(w_i|w_i i_{-n+1}^{-1}\right) = \frac{c\left(w_{i-n+1}^{i}\right)}{c\left(w_{i-n+1}^{i-1}\right)}, \tag{1.2}$$

Де $C(w_{i-n+1}^{i})$ частота потрібної N-грами.

В аналізі файлів на шкідливість N-грама також може використовуватись. Наприклад, за наявністю де-яких часто повторюваних послідовностей з бітів або символів.

2.4.3 Sequence

Sequence аналіз визначається послідовністю тегів або станів. Наприклад, маємо один об’єкт, потім він трансформується в інший, після комбінується з другим і тд. Саме цю послідовність і називають sequence. Модель sequence має схожість з моделлю N-грами. Тобто формується послідовність виконання програми.

2.5 Ознаки та евристики

Під час вилучення ознак передбачається застосування різних евристик, отриманих на основі знань в предметній області. Цю групу ознак планується активно поповнювати новими евристиками для більш успішної роботи майбутнього класифікатора виконуваних файлів.

У даній роботі ми розглянемо кілька ознак і детально обговоримо причини, за якими ці ознаки можуть бути інформативними.

Робота цих команд досить докладно описана в розділі [7]. В даному випадку важливим моментом є запис VA точки повернення командою call перед передачею керування.

Досить часто в виконуваних файлах зустрічається послідовний виклик call і команди pop, розташованої за адресою передачі управління. Як неважко помітити, після виконання двох цих команд в одному з регістрів виявиться VA

точки повернення з процедури, або, інакше кажучи, VA інструкції, наступної за call. Навіщо це може бути потрібно?

Розглянемо це на прикладі найпростіших вірусів, вони дописують свій код у кінці файлу і підміняють точку входу на себе. Тіло вірусу, як і будь-який код, складається з послідовних команд і викликів процедур. Зазвичай для виклику процедури досить знати її RVA, оскільки на етапі виконання є інформація про ImageBase і система сама вираховує VA, за яким передається виконання. Тепер припустимо, що виклик процедури стався з тіла вірусу. Якби він користувався адресацією щодо ImageBase зараженого файлу, йому б довелося кожен раз при зараженні перераховувати всі RVA, які входять до складу його коду. Це пов'язане з тим, що файли мають різний розмір, як фізичний, так і віртуальний.

Для уникнення подібних проблем, віруси надходять у такий спосіб. Усе RVA вони відраховують відносно де-якої базової інструкції, що знаходиться в початку їх коду. В цьому випадку всі RVA процедур, які використовуються вірусом, статичні від одного зараженого файлу до іншого. Для виконання тіла вірусу залишається одна задача – визначити VA цієї самої базової інструкції. Тут їм і допомагає зв'язка Call pop. Вставивши її на початку свого коду вони отримують VA, необхідне їм для подальшої роботи.

Однак легальні програми також можуть користувати цю комбінацію команд. Наприклад, багато пакувальників використовують її для тих же цілей, що і віруси. Пакувальники – програми, основним завданням яких є шифрування або скорочення фізичного обсягу виконуваного файлу. При виконанні запакованого файлу спочатку відбувається робота тіла розпакувальник, яке впроваджується в файл на етапі пакування. У цьому сенсі упаковані файли чимось схожі на файли, заражені вірусом.

У будь-якому випадку, наявність у файлі зв'язки call ... pop в більшості випадків підозріло.

Процес пошуку в файлі зв'язки команд call і pop відбувається досить просто. Пропонується здійснити лінійний прохід файлу. При зустрічі коду

операції call береться аргумент цієї команди. Далі перевіряється, яка операція знаходиться за адресою переходу. Якщо ми знаходимо там код операції pop, то автоматично прирівнюємо ознака одиниці.

Зауважимо, що розглянутий спосіб в деяких випадках може допускати помилки. Це пов'язано з ймовірністю випадкового присутності шуканої комбінації байт в файлі, що не має відношення до команд call і pop. Однак передбачається, що ця ймовірність дуже мала.

Було прийнято рішення рахувати число команд call і jmp, що зустрічаються в файлі. Ця евристика заснована на тому, що досить часто обфускація коду виконується  саме за допомогою цих команд. Код при цьому розривається на частини, які безладно переставляються місцями, і перехід від однієї частини до іншої здійснюється вставляються командами jmp або call. У таких файлах спостерігається досить високе число викликів цих команд, що відрізняє їх від файлів, що не мають обфускації. Розташування точки входу. Як було сказано, віруси часто заражають файли, дописуючи свій код в кінець і змінюючи точку входу на себе. У цьому випадку точка входу буде знаходитися десь внизу файлу. Чисті ж програми, в більшості випадків, мають точку входу у верхній частині файлу. Згадані вище пакувальники також досить часто мають точку входу внизу файлу.

Користуючись цими мыркуваннями було прийнято рішення витягувати безперервні ознаки, що дорівнюють відношенню RVA точки входу до віртуального розміру файлу.

Щільність файлу – це евристика запакованності файлу. Велика кількість ситуацій, в яких знання про запакованою файлу може стати вирішальним. Як приклад наведемо все той же руткіт. Для того щоб зрозуміти, запакований файл чи ні, існує багато відомих підходів. Одні засновані на обчисленні ентропії файлу, інші – на щільності ненульових байт в файлі або в його частинах. У даній роботі для простоти підраховувалася сумарна щільність ненульових байт в файлі. Передбачається, що дана ознака в поєднанні з іншими ознаками зможе сказати щось про запаковування файлу.

Частота команди xor - ця ознака витягується виходячи з таких міркувань. Якщо шкідливий файл використовує шифрування з ключем, то в ньому швидше за все буде високий рівень числа команди xor. Також за допомогою команди xor часто відбувається обфускація коду. При цьому, наприклад, замість команди mov eax, 0 використовується множинний виклик команди xor eax, eax.

Як було згадано вище, одним з найбільш поширених способів зараження PE файлів є дописування коду вірусу в кінець файлу. Досить часто віруси для цього створюють додаткову секцію, в яку і записують свій код. Таким чином, файли з великим числом секцій можуть бути для нас підозрілі.

Ця евристична ознака заснована приблизно на тих же міркуваннях, що і попередня. Використовуючи імена секцій також можна встановити, якою мовою програмування написаний файл або яким пакувальником він запакований.

У кожній секції виконуваного файлу є відповідний їй набір прапорів. Ці набори знаходяться в структурах, що описують секції. Серед інших в цьому наборі є прапор, який визначає можливість виконання коду, що знаходиться в даній секції.

У більшості випадків весь код виконуваного файлу знаходиться в одній секції. Якщо файл заражений вірусом, то висока ймовірність наявності двох виконуваних секцій.

Таким чином, наявність у файлі двох і більше виконуваних секцій – ознака, що вказує на можливе зараження даного файлу інфектором.

Нагадаємо, що overlay – частина PE файлу, що знаходиться за кінцем останньої секції. При завантаженні програми в пам'ять ця частина ігнорується.

Досить часто шкідливі файли зберігають використовувану ними інформацію саме в overlay. Послу завантаження в пам'ять вони можуть звертатися до власного тілу на жорсткому диску і зчитувати необхідну інформуцію з overlay.

Наявність overlay, з іншого боку, може свідчити про те, що перед нами –

файл, що саморозпаковується. Багато архівів зберігають свої ресурси саме в цій області файлу.

Так чи інакше, присутність оверлею – цікава ознака. Його комбінація з іншими ознаками може дати непогані результати.

Наприклад, наявність у файлі overlay і рядка NullSoft практично однозначно вказує на те, що перед нами – NSIS архів, що саморозпаковується (Nullsoft Scriptable Install System). Досвід показує, що в більшості випадків упакований драйвер, який імпортує KeServiceDescriptorTable – руткіт. Для того щоб зрозуміти, запакований файл чи ні, існує багато відомих підходів. Одні засновані на обчисленні ентропії файлу, інші – на щільності ненульових байт в файлі або в його частинах.

## 2.6 Інші методи статичного аналізу

Зловмисники можуть використовувати різні трюки, щоб приховати свій файл, модифікуючи його формат і змінюючи його зовнішній вигляд, щоб таким чином ввести користувача або антивірусні програми в оману. Замість того щоб покладатися на розширення файлу та дані його заголовку, можна використовувати цифровий підпис файлу для визначення його типу. І це одна із евристик, адже якщо тип файлу не співпадає з його відображенням, з великою ймовірністю він є шкідливим.

Цифровий підпис файлу - це унікальна послідовність байтів, яка прописана в його заголовку. Різні файли мають різні підписи, котрі можна використовувати для визначення їх типу та складу. Виконувані файли Windows, мають підпис файлу MZ або шістнадцяткові символи 4D 5A у перших двох байтах файлу. У [8] представлений зручний архів для анілізу сигнатур файлу.

### 2.6.1 Цифрові відбитки

Метод звірення інформації за допомогою цифрових відбитків включає в себе генерацію значень хеш-сум для підозрілого виконуваного файлу в залежності від його вмісту. Алгоритми криптографічного хешування, такі як

MD5, SHA1 або SHA256, вважаються стандартом де-факто для генерації хеш-сум зразків шкідливих програм. Наступний список описує використання криптографічних хеш-функцій.

- Ідентифікація зразка шкідливого ПО на ім'я файлу неефективна, тому що один і той же зразок може використовувати різні імена файлів, але хеш-сума, яка розраховується на основі вмісту файлу, залишиться колишньою. Отже, цей параметр для підозрілого файлу є унікальним ідентифікатором протягом усього аналізу.
- При проведенні динамічного аналізу, коли шкідлива програма виконується, вона може скопіювати себе в інше місце або додати ще один фрагмент шкідливого коду. Маючи хеш-суму зразка, можна визначити, збігається знову переміщений / скопійований зразок з вихідним чи ні. Ця інформація допоможе вам вирішити, потрібно проводити аналіз одного зразка або декількох.
- Хеш-сума часто використовується як індикатор для обміну даними з іншими фахівцями в галузі безпеки, щоб допомогти їм ідентифікувати зразок.
- Хеш-сума може використовуватися, щоб визначити, чи був зразок раніше виявлений в інтернеті або в базі служби, що здійснює аналіз підозрілих файлів і посилань, такий як VirusTotal.

2.6.2 Вилучення рядків та FLOSS

Рядки - це послідовності друкованих символів ASCII і Юникода, вбудовані в файл. Витяг рядків може підказати, як функціонує програма, і розповісти про індикатори, що вказують на підозрілий двійкового коду. Наприклад, якщо шкідлива програма створює файл, ім'я файлу зберігається у вигляді рядка в довічним файлі. Або якщо шкідлива програма дозволяє доменне ім'я, контрольоване зловмисником, це ім'я згодом зберігається у вигляді рядка. Рядки, витягнуті з виконуваного файлу, можуть містити посилання на імена файлів, URL-адреси, доменні імена, IP-адреси, команди атаки, ключі реєстру і т. Д. Хоча рядки і не дають чіткого уявлення про мету і можливості файлу, вони можуть підказати, на що здатна шкідлива програма.

У більшості випадків автори шкідливих програм використовують прості методи обфускації рядків, щоб уникнути виявлення. У таких випадках ці приховані рядки не будуть відображатися в утиліті strings та інших інструментах, призначених для вилучення рядків. FireEye Labs Obfuscated String Solver (FLOSS) - інструмент, призначений для автоматичної ідентифікації та вилучення обфусцірованних рядків з шкідливої програми. Він може допомогти вам визначити рядки, які автори шкідливих програм хочуть заховати. FLOSS також можна використовувати, як і утиліту strings, для вилучення легким для читання рядків (ASCII і Юнікод).

FLOSS комбінує в собі кілька підходів статичного аналізу для пошуку обфусцірованних ASCII рядків в уже згадуваному файлі. А саме, аналізує потік управління для розбору файлу на функції, базові блоки. Використовує евристики для пошуку декодуючих функцій. Емулює поведінку функцій декодування для вилучення читаються ASCII рядків. Такий інструментарій покликаний допомогти статичному аналізу з зашифрованими файлами і отримати більше інформації про них. Використовуючи FLOSS, наприклад, можна отримати список імпортованих арі і бібліотек. У даній роботі ми не будемо використовувати цей інструментарій. Завдання полягає в створенні швидкого і якісного алгоритму машинного навчання. Для більш глибокого аналізу варто досліджувати відмовостійкість подібних бібліотек і порівняти існуючі інструменти деобфускації. У даній роботі це питання не висвітлюється.

Таким чином, було проведено загальний огляд методик і підходів до задачі статичного аналізу шкідливого ПЗ. Більш докладні роботи, що представляють собою повний огляд методів можна знайти в [9], [10].

На відміну від попередніх досліджень, мета даної роботи полягає в:

- узагальненні отриманих результатів досліджень;
- отриманні найбільш важливих атрибутів для завдання виявлення шкідливого ПЗ на типовому для користувачів наборі файлів в умовах обмежених ресурсів і часу;
- реалізації движка в програмному продукті з використанням

сучасних алгоритмів машинного навчання.

2.6.3 Обфускація-деобфускація

Один з найцікавіших підходів до статичного аналізу виконуваних файлів представлений в [11]. Автори розглядають задачу детектування файлу як "гру в обфускацію-деобфускацію ". Мається на увазі, що відбувається постійна гонка між антивірусними компаніями, що детектують нові модифікації шкідливих файлів, і зловмисниками, які постійно намагаються приховувати функціонал своїх шкідливих файлів.

Таким чином, основне завдання антивіруса – провести деобфускацію заплутаного коду і отримати оригінальну послідовність команд для отримання уявлення про роботу файлу. Деобфускація при цьому – завдання, яке є оберненим до обфускації. В ході роботи автори в основному працювали з поліморфними вірусами – вірусами, які обфускують свій код при подальшому зараженні.

Для вирішення цього завдання автори пропонують використовувати так звані графи управління або, інакше, граф потоку керування (Control Flow Graph). Це досить відоме поняття, тому опишемо його лише в кількох словах. Граф управління – це структурна модель програми, здатна показувати зв'язок між її елементами. Вершинами графа управління є оператори розгалуження, зустрічаються в коді програми, і частини коду, де ці розгалуження зливаються. дуги – оператори обробки і передачі інформації.

Алгоритм детектування, запропонований авторами, складається з двох етапів. Спочатку на основі досліджуваного файлу будується його граф управління, потім починає працювати обфускатор, створений авторами. Його завдання – на основі графа управління згенерувати деяке число похідних графів, що не відрізняються за функціоналом від оригінального. По суті, на цьому етапі відбувається отримання графів управління обфускованих версій досліджуваного файлу. Серед отриманих таким чином графів вже шукаються ознаки, згенеровані на основі наявних шкідливих файлів.

В ході експериментів алгоритм продемонстрував безпомилкове

детектування всіх обфускованих версій вірусу. Також він пройшов усі випробування з чистими файлами.

Головна проблема такого підходу – велика кількість часу, яку займає процес генерації графа і подальша його обфускація. Час роботи алгоритму з файлом розміру 1 Мегабайт досягав 800 секунд. Очевидно, що такий алгоритм не може бути поки що застосований в складі антивірусів.

## 2.7 Методи прийняття рішень

Мета роботи [5] полягала в створенні PE парсеру – PortEX, здатного також виявляти аномалії в структурах бінарних файлів. Було проведено дослідження стратегій зараження файлів, виділені найбільш ймовірні атрибути чистих файлів. Також проводилася оцінка залежності ентропії секцій PE файлу і ймовірності того, що файл шкідливий. Для отримання ймовірності того, що файл шкідливий застосовувалися евристики. Були визначені:

- бустери (шаблони, які вказують на поведінку або зовнішній вигляд шкідливого ПО). Збільшують імовірність того, що файл буде ідентифікований як шкідливий;
- стоппери (шаблони, які вказують на поведінку або зовнішній вигляд, який є нетиповим для шкідливого ПО). Зменшують імовірність того, що файл буде визначений як шкідливий.

Обчислювалися сумарний бустер і сумарний стоппер на основі евристик і статистики по зібраним файлам і на основі отриманого значення видавалася ймовірність шкодочинності файлу. Ентропія секцій PE-файлу також впливала на ймовірноість шкідливості файлу. Межі та визначення бустерів і стопперів є евристики, засновані на статистиці з багатьох файлів, які можна поліпшити методами машинного навчання.

В [3] аналіз будується на добуванні 197 атрибутів з бінарного файлу. Атрибути представляють собою або характер присутності (DLLs referred, APIs referred), або значення полів структур файлу. Виділення значущих атрибут

відбувається на основі статистичних тестів (t-тест, $\chi$ 2-тест). Використовуючи відібрані 19-20 ознак навчають моделі бустінга, випадкового лісу. Відсутня глибший аналіз PE-формату, розгляд ділянок файлу в вигляді послідовності байт. Також немає інформації про швидкість роботи алгоритму.

Досить повна картина вибору важливих атрибут PE файлу наведена в [6]. Нижче наведена таблиця (Таблиця 1) використовуваних ознак в попередніх роботах на дану тематику. Ознаки – ознаки, витягнуті з PE файлу. Або з заголовка, або з тіла. У колонці «Структура» показані додаткові ознаки, що збираються з файлу. Додатково введені наступні скорочення: API: Application Programming Interface, BYT: Byte code, FC: Function Call, STC: Structural features, OP: Operation code

Таблиця 2.1 – Методи статичного аналізу з типами отриманих ознак

| Рік | Стаття | Тип | Структура |
|---|---|---|---|
| 2008 | Ye et al. [7] | детекція | Itemset |
| 2009 | PE-Miner [10] | детекція | - |
| 2009 | Tabish et al. [11] | детекція | N-gram |
| 2009 | Tabish et al. [11] | детекція | Sequence |
| 2009 | Hu et al. [13] | детекція | Graph |
| 2010 | Sami et al. [14] | детекція | Itemset |
| 2011 | Natara j et al. [15] | класифікація | - |
| 2012 | Jacob et al. [16] | класифікація | N-gram |
| 2013 | Santos et al. [17] | детекція | Sequence |
| 2014 | Nissim et al. [18] | детекція | N-gram |
| 2015 | DLLMiner [19] | детекція | Tree |

Завдання стояло в класифікації шкідливих програм на типи (malware family). Стаття підготовлена на основі конкурсу kaggle, де учасникам Microsoft надала 21741 файлів без заголовка PE файлу (PE header) для класифікації по 9

типам. У дослідженні основний упор був зроблений на виборі кращого набору атрибут. Вибір проводився як на основі полів структур PE файлу, так і розглядаючи файл у вигляді послідовності байт. Більш конкретно, уявімо класифікацію ознак, що розглядаються в статтях:

– hex dump-based features (Ознаки, витягнуті з послідовності байт файлу):

а) n-грам. Послідовність N байт. 1-gram - частота байтів і була обрана в статті;

б) метадані. Наприклад, розмір файлу;

в) ентропія. Підрахунок ентропії всього файлу або методом ковзного вікна;

г) подання файлу як зображення [15];

д) довжини рядків. Витяг довжин можливих рядків ASCII символів з файлу;

– Features extracted from disassembled files (Витяг ознак з структур PE файлу):

а) метадані. Наприклад, кількість рядків у файлі;

б) частота спеціальних символів. Частота символів: -, +, *,], [,?, @;

в) коди операцій. Підрахунок статистики по асемблерним інструкцій;

г) регістри. Підрахунок статистики по використовуваних регістрів;

д) application Programming Interface. Перевірка присутності / відсутності певного набору API викликів;

е) секції. Статистика по секціях;

ж) установка байт. Статистика по db, dw і dd інструкцій.

Така класифікація дозволяє в повній мірі побачити дослідження в даній області по вилученню ознак і поточні результати. Підрахунок статистики по довгій послідовності байт представляє собою дорогу операцію. А саме, підрахунок статистики по всьому файлу або подання файла як зображення.

Також були оригінальні ідеї з інших статей, а саме, подання файлу в вигляді зображення [15]. У статті наведено графік важливості ознак для визначення класу шкідливого ПЗ. В якості алгоритму використаний бустінг на

вирішальних деревах (GBoost). У реальних же умовах у нас доступна інформація з PE header.

Для повноти картини наведемо короткий опис оригінального дослідження [13], в якому PE файл представлявся у вигляді чорно-білого зображення..

Сукупність електронних даних представляли у вигляді матриці (0: чорний, 1: білий). Ширина зображення фіксувалася в залежності від розміру файлу (32 - <10kB, 64, 128, ..., 1024 -> 1000kB). На основі зображень були визначені загальні патерни, текстури для 25 типів шкідливих ПО. Для аналізу текстури застосовувався фільтр Габора. Ознаки, отримані із зображень використовували в класифікаторі - метод k-найближчих сусідів з метрикою Евкліда. Хоч дослідження і представляє інтерес, але в силу наших вимог, а саме, суворого обмеження часу роботи движка, ми змушені відмовитися від цього виду ознак.

2.7.1 Naive Bayes

Це один з найуспішніших алгоритмів навчання для категоризації тексту, заснований на правилі Байеса, який передбачає умовну незалежність між класами. Правило Байеса – закон теорії ймовірності, який розраховує ймовірність де–якої події за умови виконання взаємозалежної або незалежної. Використовуючи це правило та спільні ймовірності вибіркових спостережень і класів, алгоритм намагається оцінити умовні ймовірності класів при спостереженні.

2.7.2 Підтримка векторних машин

Або SVM. Це методика, яка базується на аналізі тестової групи. Після чого на площині креслять пряму, а все об’єкти розташовуються або з одного боку, або іншого. При аналізі нового об’єкта, він також розташовується на цій схемі, і в залежності від його розташування приймається рішення про віднесення його до одної з двох груп. Бувають SVM, що приймають рішення про віднесення файлу не до двох груп, а більше. На схумі (Рисунок. 2.1) показано, як виглядає система.

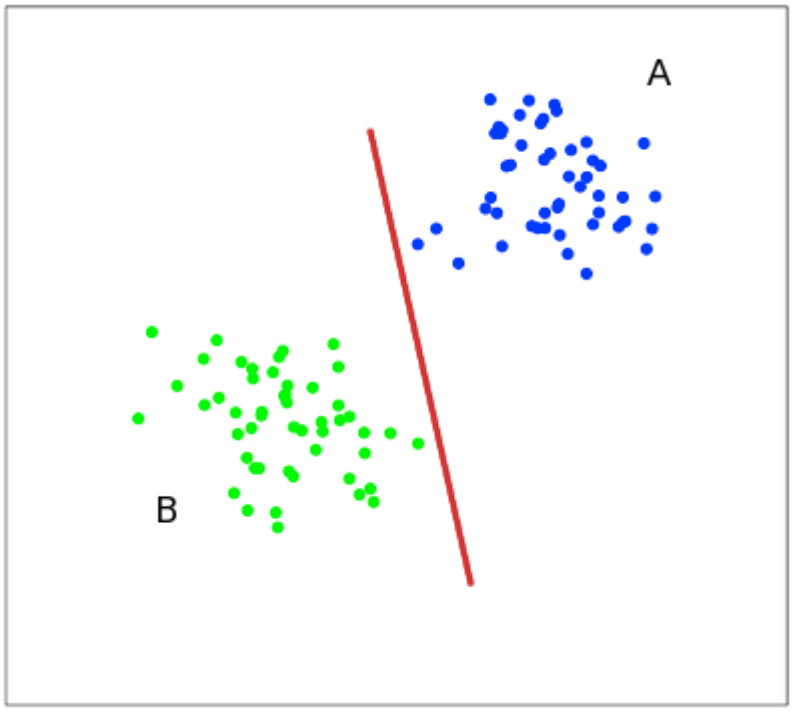


Рисунок. 2.1 – Схема SVM

2.7.3 Дерево рішень

Створюється двійкове дерево, де кожна нода – бінарна відповідь на одне запитання по одному атрибуту. Одна вітка – позитивна відповідь, інша – негативна. Таким чином кожний вузол відповідає послідовності відповідей та їх значень. Кожний лист має клас. Для того, щоб передбачити клас, шлях від кореня знаходиться в залежності від значення відповіді на кожній ноді. Відповіді отримують зверху вниз, шукаючи коефіцієнт посилення інформації кожного атрибуту, котрий дорівнює очікуваному зменшенню ентропії, викликаним розділенням об’єктів в співвідношенні з атрибутами.

2.7.4 Мультіагентний підхід

Мультіагентна система – система, утворена декількома взаємодіючими інтелектуальними агентами. Багатоагентні системи можуть бути використані для вирішення таких проблем, які складно або неможливо вирішити за допомогою одного агента або монолітної системи. У [22] наведено кортежі, за допомогою яких можна підстроїти роботи мульти агентної інтелектуальної

системи під необхідність пошуку шкудливого коду в ПЗ.

МАС виявлення і аналізу шкідливих програм може бути представлена у наступної структури (Рисунок 2.2).

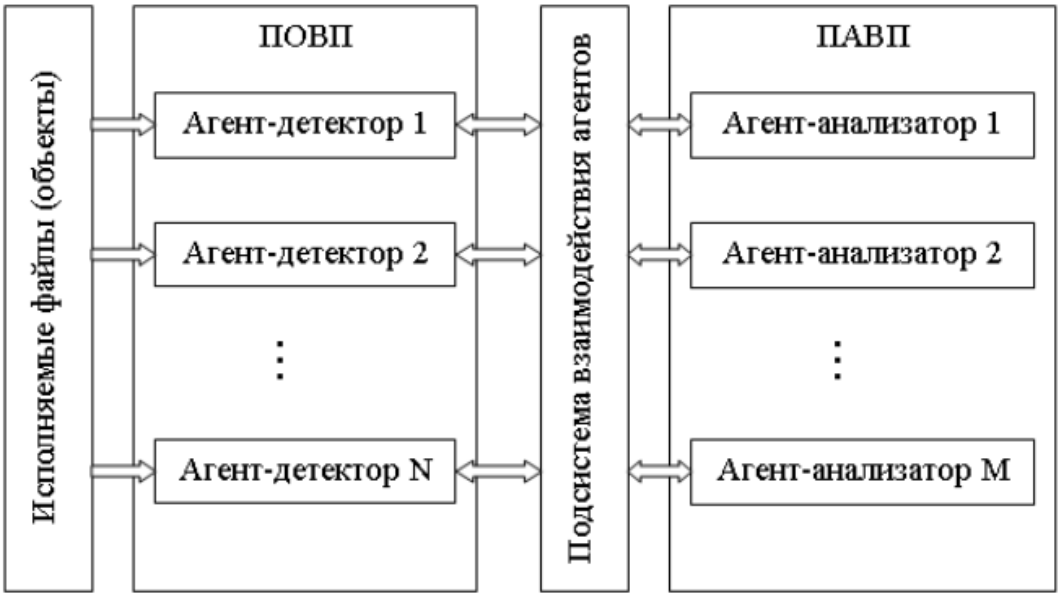


Рисунок. 2.2 – Структура МАС виявлення і аналізу шкідливих програм

У запропонованій структурі МАС можна виділити дві основні підсистеми:

- підсистему виявлення шкідливих програм (ПОВП);
- підсистему аналізу шкідливих програм (ПАВП).

Обидві підсистеми складаються з набору агентів, кожен з яких працює незалежно один від одного і вирішує загальну задачу, покладену на систему.

Виділена також окрема підсистема, що відповідає за взаємодію агентів в системі. ПОВП забезпечує збір даних, які використовуються для аналізу ситуації, і являє собою сукупність програмних функцій і алгоритмів, що забезпечують ПАВП даними для аналізу, в якості яких можуть виступати байтовий код файлу, текстові рядки всередині файлу, API Log, одиничне дію програми в рамках ОС або цілий ланцюжок таких дій і ін.

У даній системі ПОВП представлена набором агентів-детекторів які

займаються моніторингом ОС. Агент-детектор є програмним агентом, і з технічної точки зору являє собою драйвер, який працює в режимі ядра ОС і здійснює шляхом зіставлення атрибутів-ознак виконуваної програми з атрибутами-ознаками агента-детектора низькорівневий перехоплення стандартних API-функцій, виклик яких може привести до потенційної загрози безпеки комп'ютера. Драйвер, що перехопив функцію, яка змінює реєстр, формує пакет, в якому міститься необхідна для подальшого аналізу інформація про процес, який викликав цю функцію, і параметрах реєстру, які він запитував на зміну.

Шкідлива програма - це, з одного боку, файл з певним вмістом, з іншого - сукупність дій, вироблених в ОС, з третьої - сукупність кінцевих ефектів в ОС. Тому і ідентифікація програми може бути проведена на різних рівнях: по ланцюжках байт, по діям, за впливом на ОС і т.д. Відповідно до цього виділяються наступні способи збору даних для виявлення шкідливих програм, кожен з яких реалізує окремий агентдетектор і за допомогою яких можна виявити більшість вірусів:

– монітор списку автозавантаження;

– монітор завантажувального розділу;

– монітор розділу HKLM \ SOFTWARE \ Microsoft \ WindowsNT \ CurrentVersion \ Winlogon реєстру;

– монітор системного файлу C: \ windows \ system32 \ drivers \ etc \ hosts.

На один вірус може спрацювати більше одного агента-детектора, отже, необхідно враховувати і комбінації спрацьовують детекторів. Наприклад, якщо вірус намагається прописати себе в список автозавантаження, ніщо не заважає йому прописати себе в C: \ windows \ system32 \ drivers \ etc \ hosts адресу сайту, звідки він може сам захитатися в разі, якщо його виявлять і видалять зі списку автозавантаження.

Так само існують комбінації детекторів, які ніколи не спрацюють, наприклад, якщо вірус намагається прописатися в завантажувальний розділ,

значить він запуститься до завантаження ОС, отже, йому не потрібно намагатись переписати себе у список автозавантаження.

## 2.8 Класифікатор

У попередніх розділах говорилось про класифікацию шкідливого ПЗ з метою виявлення схожих ознак, функціоналу та особливостей та використування минолого досвіду зіткнення з ними. Існує багато засобів та інструментів класифікації. Кілька з них буде розглянуто у цьому розділі.

### 2.8.1 Нечітке гешування

Відмінний спосіб порівняти файли на схожість. Ssdeep (ssdeep.sourceforge.net) - корисний інструмент для створення нечіткого хешу для зразка, і він також допомагає у визначенні процентного схожості між зразками. Цей метод корисний при порівнянні підозрілого файлу із зразками зі сховища для ідентифікації схожих. Це може допомогти визначити зразки, що належать до одного сімейства шкідливих програм або до однієї і тієї ж групи суб'єктів.

### 2.8.2 Геш імпорт

Хешування імпорт - ще один метод, який можна використовувати для ідентифікації пов'язаних зразків і зразків, використовуваних одними і тими ж групами зловмисних об'єктів. Геш імпорту (або imphash) - метод, при якому значення хеш-функції обчислюються на основі імен бібліотеки / імпортованих функцій (API) і їх конкретного порядку в виконуваному файлі. Якщо файли були скомпільовані з одного і того ж джерела і однаковим чином, ці файли будуть, як правило, мати однакове значення imphash. В ході дослідження шкідливих програм, якщо вам будуть потрапляти зразки з однаковими значеннями imphash, це означає, що вони мають одну і ту ж таблицю адрес імпорту і, ймовірно, пов'язані між собою.

Наявність однакового гешу імпорту у файлів не обов'язково означає, що вони представляють одну і ту ж групу загроз; вам може знадобитися

зіставити інформацію з різних джерел, щоб класифікувати свої шкідливі програми. Наприклад, можливо, що ці зразки були створені з використанням складального комплекту, який є загальним для всіх груп; в таких випадках зразки можуть мати однаковий геш імпортує.

2.8.3 Геш секцій

Подібно гешування імпорту, гешування секцій також може допомогти у визначенні пов'язаних зразків. Коли виконуваний файл завантажується в pestudio, він обчислює MD5 кожної секції (.text, .data, .rdata та інш.).

2.8.4 YARA

Зразок шкідливого ПЗ може містити багато рядків або індикаторів двійкових файлів; розпізнавання рядків або двійкових даних, які є унікальними для зразка шкідливого ПЗ або сімейства шкідливих програм, може допомогти в їх класифікації. Експерти з безпеки класифікують шкідливі програми на основі унікальних рядків і індикаторів двійкових файлів, присутніх в двійковому коді. Іноді шкідливі програми також можуть бути класифіковані на основі загальних характеристик. YARA є потужним засобом ідентифікації і класифікації шкідливого ПЗ. Порівняння і класифікація шкідливих програм можуть створювати правила YARA на основі текстової або двійковій інформації, що міститься в зразку. Ці правила складаються з набору рядків і логічного виразу, яке визначає його логіку. Як тільки правило написано, ви можете використовувати його для сканування файлів із застосуванням утиліти YARA або використовувати yara-python для інтеграції з інструментальними засобами. Правила YARA утворюються з наступних компонентів :

– ідентифікатор правила: це ім'я, яке описує правило (suspicious_strings в попередньому прикладі). ідентифікатори правила можуть містити будь-який буквено-цифровий символ і знак підкреслення, але перший символ не може бути цифрою. Ідентифікатори правила чутливі до регістру, і їх кількість не може перевищувати 128 символів;

– визначення рядка: це розділ, де визначені строки (текст, шістнадцяткові або регулярні вирази), які будуть частиною правила. Ця

секція може бути опущена, якщо правило не спирається на будь-які рядки. Кожен рядок має ідентифікатор, що складається з символу $, за яким слід послідовність літер та цифр і підкреслення. Виходячи з попереднього правила, розглядайте $ a, $ b і $ c як змінні, що містять значення. Ці змінні потім використовуються в секції умов;

– секция условий: это не дополнительная секция. Здесь находится логика правила. Эта секция должна содержать логическое выражение, указывающее условие, при котором правило будет соответствовать или нет.

# 3 ДИНАМІЧНИЙ АНАЛІЗ

Динамічний аналіз (поведінковий аналіз) включає в себе аналіз зразка шляхом виконання його в ізольованому середовищі і моніторинг його діяльності, взаємодії і впливу на систему.

В ході динамічного аналізу, коли шкідлива програма виконується, ви будете проводити різні дії з моніторингу. Мета полягає в зборі даних в режимі реального часу, що відносяться до поведінки шкідливого ПО і його впливу на систему.

## 3.1 Моніторинг процесів

Включає в себе моніторинг активності процесів і вивчення властивостей підсумкового процесу під час виконання шкідливих програм. Напйбільш поширеним методом є технології HIPS, VIPS та Пісочниця (sandbox)., яка передбачає контролю активності, заснована на перехопленні звернень до ядра ОС і блокуванні виконання потенційно небезпечних дій ПЗ, яке працює в режимі користувача, виконуваних без відома користувача [23]. За допомогою власного драйвера перехоплює всі звернення ПЗ до ядра ОС.


Рисунок 3.1 - Схема взаємодії процесів і ядра ОС в методі HIPS

У разі спроби виконання потенційно небезпечного дії з боку ПЗ, HIPS-система блокує виконання даної дії і видає запит користувачеві, який вирішує

дозволити або заборонити виконання даної дії. Схема взаємодії процесів і ядра ОС в методі HIPS показано на рис. 3.1.

Переваги систем, побудованих на методі HIPS:

– низьке споживання системних ресурсів;

– не вимогливі до апаратного забезпечення ПК (можуть працювати на

– різних платформах);

– можливість визначення загроз нульового дня; - можливість визначення руткітів, які працюють в режимі користувача.

## 3.2 Моніторинг файлової системи та IFT

Цей метод ключає в себе моніторинг активності файлової системи в режимі реального часу при виконанні шкідливого коду. Існує кілька інструментів та методів для виявлення несанкціонованного втручання у файлову систему та зміни інормації на дисках. Підходи відстеження інформаційного потоку (IFT) використовуються для моніторингу програм з точки зору того, як програма обробляє дані. Під час аналізу підозрілі або зацікавлені дані забруднюються або маркуються перед їх обробкою, а потім спостерігається їх поширення під час обробки даних. Відстеження потоків інформації через дані про забруднення також ыменуэться аналізом забруднень [25]. У роботі [26] автор ілюструє процес відстеження інформаційного потоку на простому прикладі, розглянутому нижче в таблиці 3.1. Вважається, що розташування стеку, z, є небезпечним місцем. Отримані дані x із мережі вважаються підозрілими та зараженими. Під час обробки x копіюється в інше місце, y. Тепер під час виконання програми, якщо програма переходить у вже позначене розташування z, буде піднятий сигнал, що вказує на шкідливу активність.

Таблиця 3.1 – Простий процесс IFT

| Виконання файлу | IFT |
|---|---|
| …. | Tag(z) = 1 Підозріле місце позначено як небезпечне |
| Отримано (& x); | Tag(x) =1 Небезпечні дані позначені тегами |
| y = x | Tag(y) = Tag(x) |
| … | … |
| Jmp z | Файл є небезпесний, оскільки z вже позначено як підозрілий |

Деякі ключові поняття, що використовуються в процесі аналізу IFT, описані нижче:

Забруднені джерела та місця призначення:

Забруднені джерела - це компоненти системи відстеження інформаційного потоку, яка позначає зацікавлені або підозрілі дані в системі. Вони забруднюють дані і надають їх як вхідні дані до системи. З іншого боку, забруднені пункти призначення - це елементи, які реагують на зіпсовані дані, введені джерелами забруднень у спеціальному механізмі. Відповіді можуть викликати інше джерело або викликати попередження.

Прямі забруднені дані: як випливає з назви, прямі забруднені дані стосуються безпосереднього маркування операнда. Загальним правилом забруднення даних є те, що коли зіпсоване значення джерела переміщується або копіюється до іншого операнда, операнд призначення також забруднюється [27]. Однак система IFT може містити більше одного безпосередньо зіпсованого операнда, залученого до однієї (або декількох) інструкцій (ій) з чіткими мітками, як показано в таблиці 3.2. Слід дотримуватися керівних принципів, які визначають, як слід поводитись із цим зіпсованим операндом. Наприклад, найбільш підозрілий або зацікавлений операнд можна відстежувати за іншим, або їх можна аналізувати в комбінованому вигляді під загальною

міткою.

Таблиця 3.2 – Пряме пошкодження даних

| Виконання файлу | IFT |
|---|---|
| x = 0 | Tag(y) = Tag(x) Спрямовані заражені дані з чіткими тегами |
| y = y + x | Tag(y) = Tag(x) + Tag(x) Спрямовані заражені дані різними мітками в арифметичній операції |

Адреси зараження: зараження покажчика або адреси використовується, коли система відстеження потоку інформації використовується для спостереження за неконтрольованими експлоїтами даних або аналізу витоку конфіденційної інформації, і здійснюється шляхом генерації адреси за допомогою заражених даних [28]. Коли адреса A заражена, вона виявляється як шкідлива дія, якщо під час виконання програми де-посилання на A відмінено. Розглянемо приклад у таблиці 3.3 нижче.

Таблиця 3.3 - Пошкодження адреси

| Виконання файлу | IFT |
|---|---|
| x = 0 | Tag(y) = Tag(x) Спрямовані заражені дані різними тегами |
| A = &x | Пошкоджена адреса A, згенерована з використанням зіпсованого значення X |
| A = 1 | Пошкоджена адреса A знята з посилань - Створити сигнал тривоги |

Контрольні залежності: Окрім забруднення даних та адрес, відстеження потоків інформації також може виконуватися з використанням контрольного забруднення потоку. Наприклад, інструкція X залежить від керування

командою B, якщо B контролює виконання A [28]. Ілюстрація забруднення потоку управління наведена в таблиці 4. Це означає, що адресу або дані можна відстежувати лише в тому випадку, якщо інший пошкоджений операнд, від якого залежить його виконання, також обробляється системою.

Політика зараження: Політика зараження - це набір правил, що визначають, як слід проводити аналіз неприємностей у системі відстеження інформаційного потоку, і охоплює три аспекти аналізу неприємностей; вступ, розповсюдження та перевірка [28]. Правила введення забруднення визначають спосіб введення забруднення в систему, розповсюдження визначає, як слід оновлювати статуси забруднень для даних, отриманих із забруднених або незаплямованих операндів, і проводиться перевірка для перевірки стану забруднення та відповідних дій, які потрібно вжити.

Надмірне та помірне зараження: це два типи помилок, які можуть виникнути під час аналізу відстеження інформаційного потоку [28]. Надмірність виникає, коли система позначає дані як зіпсовані, коли вони не походять від забрудненого джерела. Помірне відбувається, коли в процесі аналізу втрачається значення, яке має бути зарпаження в процесі відстеження інформаційного потоку

## 3.3 Моніторинг реєстру

Включає в себе моніторинг змінених ключів реєстру і ключів, до яких був отриманий доступ, а також даних реєстру, які зчитуються / записуються шкідливим файлом.

У Microsoft Windows файл реєстру є базою даних про конфігурацію комп'ютера. Реєстр містить інформацію, на яку постійно посилаються багато різних програм. Інформація, що зберігається в реєстрі, включає обладнання, встановлене в системі, використовувані порти, профілі для кожного користувача, налаштування конфігурації програм та багато інших параметрів системи. Це основне місце зберігання всієї інформації про конфігурацію для

багатьох програм Windows. Реєстр Windows також є джерелом усієї інформації про безпеку: політики, імена користувачів та паролі. Реєстр також зберігає велику частину важливої інформації про час виконання e-rat guratimr, необхідну для запуску програм. Реєстр організований ієрархічно у вигляді дерева. Кожен запис у реєстрі називається ключем і має відповідне значення. Одним із прикладів ключа реєстру є HKCU\Software\America Online\AOL Instant Messengae (TM)\ CurrentVersion\Users\amiuser\Login\Password

Це ключ, який використовується миттю AOL. програма обміну повідомленнями. Цей ключ зберігає версію пароля encrypttxl для. ім'я користувача. Після запуску. Програма обміну миттєвими повідомленнями AOL запитує цей ключ у. реєстру, щоб отримати збережений пароль для локального користувача. Доступ до інформації з реєстру здійснюється за допомогою окремих звернень до реєстру або запитів. Інформація, пов’язана із запитом реєстру, - це ключ, тип запиту, результат, процес згенерував запит і чи був запит успішним.

Реєстр Windows є ефективним джерелом даних для моніторингу атак, оскільки багато атак виявляються як аномальна поведінка реєстру. Багато атак використовують перевагу залежності Windows від реєстру. Дійсно, багато атак самі покладаються на реєстр Windows, щоб нормально функціонувати. Багато pmgrams зберігають важливу інформацію в реєстрі, не зважаючи на те, що інші програми можуть попередньо отримати доступ до інформатики. Хоча деякі версії Windows включають дозволи безпеки та реєстрацію реєстру, обидві функції використовуються рідко (через обчислювальні накладні витрати та складність варіантів конфігурації).

Більшість програм Windows отримують доступ до певного набору ключів реєстру під час звичайного виконання. Більше того, більшість користувачів регулярно використовують певний набір програм. Запуск своїх машин. Це може бути набір усіх програм, встановлених на машині, або, як правило, невелика підмножина цих програм. Ще однією важливою характеристикою діяльності Реєстру є те, що вона, як правило, регулярна з часом. Більшість програм або

отримують доступ до реєстру лише під час запуску та вимкнення, або отримують доступ до реєстру через певні проміжки часу. Ця регулярність робить реєстр чудовим місцем для пошуку нерегулярних, аномальних дій, оскільки. шкідлива програма може суттєво відхилятися від звичайної діяльності і може бути виявлена.

Багато атак пов’язані із запуском програм, які ніколи раніше не запускались, і зміною ключів, які не змінювалися з моменту першої встановлення операційної системи виробником. Якщо модель нормальної поведінки реєстру обчислюється за чистими даними, то такі види операцій реєстру не відображатиметься в моделі. Крім того, шкідливим програмам може знадобитися запитувати частини реєстру, щоб отримати інформацію про вразливості. Шкідлива програма також може ввести нові ключі, які допоможуть створити вразливості на машині.

Деякі приклади шкідливих програм та способи їх створення аномальної діяльності реєстру описані нижче.

– Налаштування троянських програм: Ця програма при запуску додає повний доступ для спільного читання / запису у файловій системі хост-машини. Він використовує реєстр, створюючи структуру реєстру в мережевих секціях клавіш Windows. Структура походить від HKLM \ Software \ Microsoft \ Windows \ CurrentVersion \ Network \ LanMan. Потім він створює, як правило, вісім нових ключів для власного використання. Він також отримує доступ до HKLM \ Security \ Provider, щоб знайти інформацію про безпеку машини, яка допоможе визначити вразливі місця. Цей ключ не доступний жодним звичайним пртуграні під час навчання або тестування в наших експериментах, і його використання є явно підозрілим за своєю суттю.

– Попередня версія 2000: Ця програма відкриває вразливість на хост-машині, яка надає будь-кому. з. назад джерело. клієнтська програма завершена. контроль над. господар. машина. Ця програма робить. широке використання. реєстр, проте це. використовує ключ, який. дуже рідко

доступний в системі Windows. HKLM \ Softuare \ Microsoft \ VBA \ Monitors не отримував доступу до звичайних програми в навчальних або тестових даних, що дозволило нашому алгоритму визначити його як аномальний. Ця програма також запускає багато інших програм (LoadWC. Exe, Patch. Exe, runonce. Exe, b02k_1-o-intl. E) як частину атаки, яку всі зробили. аномальний доступ до. \ Реєстр Windows.

– Aimrecover: це програма, яка викрадає паролі у користувачів AOL. Насправді це дуже проста програма, яка просто зчитує ключі з реєстру, де програма AOL Instant Messenger зберігає імена користувачів та паролі. Причина того, що ці звернення аномальні, полягає в тому, що Aimrecover отримує доступ до ключа, до якого зазвичай отримує доступ інша програма, яка створила цей ключ.

– Disable Norton: Це дуже проста експлуатація реєстру, яка відключає Norton Antivirus. Ця атака перемикає один запис у реєстрі, ключ HKLM \ SOFTWARE \ INTEL \ LANDesk \ VirusProtect6 \ CurrentVersion \ Storages \ Files \ System \ RealTimeScan \ UnOff. Якщо для цього значення встановлено значення 0, тоді моніторинг системи реального часу Norton Antivirus вимикається. Знову ж це аномально через доступ до ключа, який був створений іншою програмою.

– LOphtCrack: Ця програма, мабуть, найпопулярніша програма злому паролів для машин Windows. Він отримує хешований файл SAM, що містить паролі для всіх користувачів, а потім використовує словник або підхід bruteforce для пошуку паролів. Ця програма також використовує недоліки схеми шифрування Windows, що дозволяє програмі виявляти деякі символи в паролі. Ця програма використовує реєстр, створюючи власний розділ у реєстрі. Він буде складатися з багатьох запитів створення ключів та встановлення значень, які будуть розміщені на ключах, які раніше не існували на хост-машині, і тому їх раніше не бачили.

Ще одна важлива інформація, яка. може бути. використовується для

виявлення атак, всі програми, що спостерігаються у нашому наборі даних, і, мабуть, усі програми загалом, змушують Провідник Windows отримувати доступ до певного ключа. Ключ HKLM\Software\Microsoft\Windows NT\ CurrentVersion\Image File Execition Option\processName де processName - це назва процесу, що виконується, - це ключ, до якого EXplorer отримує доступ кожного разу, коли запускається програма. Тому ми. мати контрольну точку для кожного конкретного запущеного додатка для визначення шкідливої діяльності. Крім того, багато програм додають себе до автозапуску секції реєстру Windows під HKLM\Software\Microsoft\ Windows\Current Version\Run

Хоча це не злоякісно за своєю суттю, це рідкісна подія, яка, безумовно, може бути використана як натяк на атаку на систему. Троянські програми, такі як Back Orice, використовують цю частину реєстру для автоматичного завантаження кожного завантаження. Детектори аномалій не шукають шкідливих дій (прямо. Вони шукають відхилення від нормальної діяльності. Саме з цієї причини будь-яке відхилення від нормальної діяльності буде оголошено системою атакою. Встановлення нової програми в системі буде розглядатися як аномальна діяльність. Програми часто створюють нові розділи реєстру та багато нових ключів при встановленні. Це призведе до помилкової тривоги, подібно до того, як додавання нової машини до мережі може відбутися викликати сигнал тривоги на детекторі аномалій, який аналізує мережевий трафік. Існує кілька можливих рішень цієї проблеми. Шкідливі програми часто є скритними та встановлюються безшумно, так що користувач не знає, що програма встановлюється. Це не стосується більшості встановлених користувачем (законних) установок програм, які дають про себе знати. Алгоритм можна змінити, щоб ігнорувати тривоги під час роботи програми встановлення щита, оскільки це означало б, що користувач знає, що встановлюється нова програма. Інший варіант - просто підказати користувачеві, коли відбувається виявлення, щоб користувач міг дозволити виявлення аномалій System. Далі виконується законна встановлена програма, і тому модель виявлення аномалій потребує оновлення за допомогою нещодавно

доступного навчального набору, зібраного в режимі реального часу. Це типова взаємодія з користувачем у багатьох установках додатків, де для отримання інформації про конфігурацію вимагається зворотний зв’язок із користувачем.

Система RAD має три основних компоненти: датчик аудиту, модельний генератор та детектор аномалій. Сенсор реєструє кожну діяльність реєстру або в базі даних, де вона зберігається для навчання, або в детекторі, який повинен бути. використовується для аналізу. Генератор моделей зчитує дані з бази даних і створює модель нормальної поведінки. Потім модель використовується детектором аномалій, щоб вирішити, чи слід вважати кожен новий доступ до реєстру аномальним.

Для того, щоб виявити аномальний доступ до реєстру, RAD генерує модель звичайної діяльності реєстру. Набір п’яти функцій витягується з кожного доступу до реєстру. Використовуючи ці значення функцій над звичайними даними, генерується модель поведінки звичайного реєстру. Ця модель нормальності складається з набору перевірок узгодженості, застосованих до ознак. При виявленні аномалій модель нормальності визначає, чи відповідають значення в функціях поточного доступу до реєстру нормальним даним чи ні. Якщо нова діяльність не узгоджується, алгмітм позначає доступ як аномальний.

## 3.4 Моніторинг мережі

Виявлення ознак роботи зловмисного програмного забезпечення за допомогою моніторингу мережевого зв'язку є одним з основних методів, що роками застосовується для боротьби з мережевими загрозами. Багато систем IDS / IPS аналізує пакунки різними способами з метою виявлення та блокування зв'язку, пов'язаної із загрозами. Нещодавні напади на системи відомих корпорацій [29,30] та державні [31] установи показують, що класичні зовнішні методи виявлення, засновані на аналізі підписів, статистичних аномалій або евристичних методів, спрямованих на захист ззовні, не працюють,

і є легко обійти новими поколіннями шкідливих програм.

Гірше того, методи, спрямовані на захист від нового зараження шкідливим програмним забезпеченням, часто не справляються з виявленням слідів існуючих шкідливих програм [32,33]. Все більш очевидною стає необхідність доповнювати існуючі методи захисту ефективним рішенням для виявлення слідів шкідливої програми в мережі установ? необхідність створення додаткової, наступної лінії оборони, спрямованої на виявлення та блокування того, що потрапило всередину та функціонує в мережевому середовищі.

Робота комп'ютерної системи в мережевому середовищі завжди проявляється у двох режимах: як сервер- пасивний суб'єкт, що працює у відповідь на зовнішній запит на виконання конкретних послуг і як клієнт-активна сторона, яка ініціює комунікаційні операції, відправляючи запити до інших систем мережі.

У класичних рішеннях безпеки увага зосереджується головним чином на серверній частині зв'язку, захищаючи доступ до системних служб від мережі (наприклад, за допомогою «перемотки»), ретельно контролюючи доступ до системних портів на різних рівнях деталізації. Загалом, не існує зв’язку, ініційованого сервером. Клієнтська сторона вважається безпечною з припущення, спілкування ініціюється програмами, встановленими в системі, за замовчуванням під контролем користувача.

Цей підхід дав модель надійного захисту системного зв'язку іззовні, з поблажливим ставленням до вихідного трафіку, який за замовчуванням вважається безпечним. Поява сучасного покоління шкідливих програм (хробаки, троянські програми, шпигунські програми, нарешті боти) [34,35] змусило змінити погляд на безпеку системної комунікації в мережі та спроби охопити моніторингом вихідної комунікації із системи так само.

Впровадження цього моніторингу стикається з певними проблемами, що вимагають іншого підходу до правил, що визначають дозволену та заборонену поведінку. Підключення із зовнішнього світу надходять до відомих номерів портів, зазвичай фіксованих для конкретної послуги - можна легко

сформулювати правила контролю доступу.

Вихідний зв’язок здійснюється з будь-яких вихідних портів до будь-якого пункту призначення - не існує простих правил, які визначають, які вихідні порти дозволені чи ні для зв’язку. Однак адміністратори можуть контролювати, яким горщикам у зовнішніх системах вони відкриваються, і тим самим обмежувати можливість користування певними послугами користувачами.

Деякі служби, такі як http, пошта, настільки поширені, що стають стандартами, необхідними для виконання повсякденних завдань, і зовнішній доступ до портів цих служб зазвичай відкритий. Зв'язок з іншими, нетиповими портами може легко бути і часто блокується.

З іншого боку, центральні системи моніторингу не мають доступу до іншої IP-інформації, яка є джерелом, тому вони не знають, хто який процес відповідає за ініціювання вихідного зв'язку до даного сервісного порту, і чи безпечний він повинен бути дозволений чи ні. У більшості випадків такий зв’язок вважається безпечним і дозволеним.

Ця асиметрія постави безпеки добре відома і також використовується у зловмисних намірах - багато шкідливих програм змінили свою поведінку з серверного на клієнтський режим і активно "дзвонять додому", щоб перевірити команду [36], замість цього пасивно очікуючи на вхідні завдання.

3.4.1 Активність системи

Моніторинг мережевого зв'язку вимагає реєстрації з'єднань, що відкриваються системними програмами до інших систем (активна сторона зв'язку), портів, що відкриваються за допомогою підтримуючих їх програм (сторона сервера), та реєстрації вхідних з'єднань з інших систем.

Така реєстрація може бути здійснена на комунікаційних інтерфейсах системи на рівні ядра. У Windows існує два рівні інтерфейсів - специфікація інтерфейсу мережевого драйвера (NDIS) і транспортний інтерфейс транспортного драйвера (TDI) на рівні мережі, що надає повну інформацію про системні підключення.

Групуючи записані комунікаційні події за задіяними програмами, можна

отримати набір типів підключення, в якому бере участь дана програма - її модель спілкування. Для аналізу мережевих загроз було визначено поняття комунікаційного профілю програми як сукупності номерів вихідних з'єднань із певними портами призначення, до яких дана програма встановлює з'єднання за певний проміжок часу. Колекція профілів активних програм створює профіль діяльності системи.

Результати аналізів профілю діяльності мережної активності Windows можна побачити у дослідженні [37].

3.4.2 Діяльність шкідливого ПЗ у профілях зв'язку

Поява шкідливої програми змінює мережевий профіль діяльності системи - з'являється новий відправник, який контактує зі своїми серверами C&C. Робота зловмисного програмного забезпечення в системі може спричинити кілька типів ефектів у комунікаційному профілі системи: можуть з’являтися нові процеси або програми, що контактують з відомими або новими портами призначення - легко видимі шкідливі програми; можуть бути зміни в поведінці відомих системних програм, наприклад, поки що не здійснюється зв’язок в мережі, програми починають надсилати пакети (спроби встановити зв’язок) або відбуватимуться зміни в поведінці відомих програм у комунікаційному профілі системи - це відповідає ситуації, яка називається ін'єкцією .dll - вводиться шкідливий код (додається до коду запуску в програмі оперативної пам'яті), змінюючи його поведінку.

Проблема полягає у визначенні всіх елементів профілю активності шкідливого програмного забезпечення. Під час тестування систем медоносних банок та запуску на них зареєстрованих копій шкідливого програмного забезпечення було помічено, що в системі на момент зараження запускаються кілька програм, що по-різному ведуть себе.

Часто запускається новий процес спілкування в мережі, але також з'являються модулі шкідливого програмного забезпечення, що працюють як частина інших компонентів Windows, які зазвичай не діють у мережі, таких як notepad.exe або Explorer.exe, або модифікують діяльність мережевих програм.

При аналізі ряду прикладів шкідливого програмного забезпечення було відмічено появу деяких інших суттєвих відмінностей між чистою та зараженою системою пролів, що може бути ефективним елементом для класифікації підозрілих програм. У роботі [37] записує як один з елементів опису підключення клас подій, таких як CONNECT, DATAGRAM для успішного підключення TCP або UDP, а також помилки (TIMEOUT, RESET, CANCELED, UNREACH, ERR: тощо). Такі помилки практично не трапляються в профілі активності "чистої" системи, але у профілі активності шкідливого програмного забезпечення є досить багато, часто стосовно нестандартних портів призначення зв'язку.

# 4 РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ

## 4.1 Мета та завдання

Мета полягає у створенні ПЗ, що з достатньою достовірністю відносило би тестовий файл до шкідливих або безпечних. Програма не повинна давати поради відносно наступних дій, проте повинна показувати, чому саме той чи інший зразок відноситься до шкідливого ПЗ.

## 4.2 Огляд існуючих рішень

### 4. 2.1 Maltrail

Це система виявлення зловмисного трафіку, що використовує загальнодоступні (чорні) списки, що містять шкідливі та / або загалом підозрілі маршрути, разом із статичними слідами, складеними з різних звітів AV та користувацьких списків, визначених користувачем, де стежка може бути будь-якою з доменних імен (наприклад, zvpprsensinaix.com для зловмисного програмного забезпечення Banjori), URL-адреси (наприклад, hXXp: //109.162.38.120/harsh02.exe для відомого шкідливого виконуваного файлу), IP-адреси (наприклад, 185.130.5.231 для відомого зловмисника) або значення заголовка HTTP User-Agent (наприклад, sqlmap для автоматичного SQL інструмент ін'єкції та поглинання бази даних). Крім того, він використовує (необов’язково) вдосконалені евристичні механізми, які можуть допомогти у виявленні невідомих загроз (наприклад, нових шкідливих програм).

Що стосується статичних записів, то для наступних шкідливих об’єктів (наприклад, зловмисних програм C&C чи пробоїн) були включені вручну (з різних звітів AV та особистих досліджень автора).

Maltrail базується на архітектурі клієнта Traffic -> Sensor <-> Server <->. Датчик (и) - це автономний компонент, що працює на вузлі моніторингу (наприклад, платформа Linux, підключена пасивно до порта SPAN/ дзеркального відображення або прозоро вбудована на мосту Linux), або на

автономній машині (наприклад, Honeypot), де вона «контролює» прохідний трафік для елементів/маршрутів, включених до чорного списку (тобто доменних імен, URL-адрес та/або IP-адрес). У разі позитивного збігу він надсилає деталі події на (центральний) Сервер, де вони зберігаються у відповідному каталозі журналювання (тобто LOG_DIR, описаний у розділі конфігурації). Якщо Sensor запускається на тій самій машині, що і Сервер (конфігурація за замовчуванням), журнали зберігаються безпосередньо в локальному каталозі журналювання. В іншому випадку вони надсилаються через повідомлення UDP на віддалений сервер (тобто LOG_SERVER, описаний у розділі конфігурації).

Основна роль сервера - зберігати деталі події та забезпечувати внутрішню підтримку веб-додатку, що звітує. У конфігурації за замовчуванням сервер і датчик працюватимуть на одній машині. Отже, для запобігання потенційним збоям у діяльності датчика, інтерфейсна частина звітування базується на архітектурі «Жирний клієнт» (тобто вся подальша обробка даних здійснюється всередині екземпляру веб-браузера клієнта). Події (тобто записи в журналі) за обраний (24 год) період передаються Клієнту, де веб-додаток, що звітує, несе повну відповідальність за частину презентації. Дані надсилаються клієнту стиснутими шматками, де вони обробляються послідовно. Підсумковий звіт створюється у стислій формі, практично дозволяючи презентувати практично необмежену кількість подій.

### 4.2.2 Aalisha

У цій роботі автори в першу чергу зосереджувались на виконуваних файлах Windows. Зразки зловмисного програмного забезпечення були зібрані з набору даних шкідливого програмного забезпечення git-hub zoo.

Виклики API представляють абстрактну функціональність програм у виконуваній програмі Windows. API Windows дзвінки також використовуються для отримання інформації на рівні системи. Шкідливий код використовує ці виклики для здійснення зловмисної поведінки. Послідовність інструкцій про перехід і виклик у шкідливих програмах може йому стати в нагоді розрізняти

доброякісні та шкідливі програми. Антивірусне програмне забезпечення, розроблене з використанням підходу bascd підпису, вразливе до затухання зловмисного програмного забезпечення. Поряд з цим, підхід на основі підписів має вищу часову та космічну складність. За авторським підходом вони розробляють модель класифікації, яка фіксує поведінку доброякісних та шкідливих програм через графік теоретичні особливості, отримані з графіків викликів API.

Графіки викликів функції можна генерувати з виконуваних програм за допомогою програмних засобів, що дозволяють реверс-інженерію. Інструмент, що використовується в цьому проекті, є інструментом з відкритим кодом. radare2. Це зворотна інженерна структура, яка підтримує виконувані файли для Windows, Linux, а також операційної системи Mac OS X. Він також підтримує неправильно сформовані двійкові файли, які зробив його відповідним інструментом для дослідження. Інструмент був використаний для розбирання виконуваного файлу та аналізу за допомогою графіків викликів функцій. Те саме можна досягти і за допомогою власного програмного забезпечення IDA Pro. Але IDA Pro обмежує свої функціональні можливості виконуваними програмами Windows.

На першому кроці програма створює графік викликів API доброякісних та шкідливих програм з їх виконуваних файлів за допомогою radare2. Після розбору всього бінарного файлу та запуску статистичного аналізу в radare2. Сформовані графіки викликів зберігались у форматі .dot. Вигляд розбирання представляє графік викликів у radare2 як базовий блок коду збірки, розділений інструкцією умовного переходу. Сформований графік для всіх функцій зберігається у форматі .dot graph графі.

## 4.3 Огляд використаних інструментів

### 4.3.1 Python-magic

Python-magic - це інтерфейс Python до бібліотеки ідентифікації типу

файлу libmagic. libmagic визначає типи файлів, перевіряючи їх заголовки відповідно до попередньо визначеного списку типів файлів. Ця функція виконується в командному рядку командою Unix.

4.3.2 Yara

YARA - це інструмент, спрямований (але не обмежуючись цим) на допомогу дослідникам зловмисного програмного забезпечення ідентифікувати та класифікувати зразки шкідливих програм. За допомогою YARA ви можете створювати описи сімейств шкідливих програм (або все, що ви хочете описати) на основі текстових або двійкових шаблонів. Кожен опис, тобто правило, складається з набору рядків і булевого виразу, які визначають його логіку.

4.3.3 Radare2

R2 - це переписаний radare з нуля, щоб забезпечити набір бібліотек та інструментів для роботи з бінарними файлами. Проект Radare розпочався як інструмент криміналістики - це шістнадцятковий редактор командного рядка, який можна було відкрити на диску, але згодом було додано підтримку для аналізу двійкових файлів, розбирання коду, дизасемблювання коду, налагодження програм, підключення до віддалених серверів GDB та багато іншого. Також версія підтримує функцію sandbox.

4.3.4 Pandas library

Pandas library - це модуль Python, який широко використовується у спільноті дослідників даних. Він надає функціонал для інкапсуляції маніпуляцій з даними, готуючи кадри даних до аналізу.

4.3.5 NumPy

NumPy це open-source модуль для Python, який надає загальні математичні і числові операції у вигляді напередскомпільованих швидких функцій. Вони об'єднуються в високорівневі пакети. Вони забезпечують функціонал, який можна порівняти з функціоналом MatLab. NumPy (Numeric Python) надає базові методи для маніпуляції з великими масивами і матрицями. SciPy (Scientific Python) розширює функціонал numpy величезною колекцією корисних алгоритмів, таких як мінімізація, перетворення Фур'є, регресія, і інші

прикладні математичні техніки.

4.3.6 Noriben

Noriben - це скрипт на основі Python, який працює разом із Sysinternals Procmon для автоматичного збору, аналізу та звітування про показники виконання шкідливого програмного забезпечення. У двох словах, це дозволяє запускати додатки, натискати клавішу та отримувати простий текстовий звіт про діяльність зразка.

4.4 Огляд використаних алгоритмів

4.4.1 Евристичний сканер

Після вилучення ознак з файлу, вони сортуються та порівнюються з записаною бібліотекою ознак ( опис в ДОДАТКУ А).

Якщо ознака співпадає, лічильник сканеру збільшується. Остаточний результат передається до оцінюючого модулю.

4.4.2 IFT

Алгоритм відстеження поведінки. Він працює за наступним алгоритмом. Нехай множина P = (a 1, a 2, ..., $a_n$) є профілем, створеним із програми шляхом вилучення її критичних викликів API, де і представляє і-й виклик критичного API, а n - загальна кількість критичних викликів API. Щоб отримати послідовність операторів для кожного критичного виклику API, ми обчислюємо траєкторію поведінки програми щодо набору P. Трасування - це підмножина програми, яка містить лише ті оператори, які оцінюють аргументи, що вводяться до відповідного критерію трасування. . У нашому випадку критерій відстеження містить набір критичних викликів API P. Розглядаються деякі наближення під час генерації таких слідів:

– Для всіх прямих циклів, як для операторів, do-while та ін., Що виникають під час зворотного трасування, ми обходимо цикл щонайбільше 4 рази. Таке наближення допомагає зближенню алгоритму трасування, не втрачаючи суттєво семантики трасування, як це

демонструють наші результати.

– Для непрямих циклів у графіку потоку керування ми розглядаємо алгоритм маркування, який не переглядає вже відвіданий вузол і, отже, не додає його до поточного трасування.

– Для умовних тверджень, наприклад if-else, ми припускаємо, що вони завжди оцінюються як істинні. Це призводить до вищого охоплення коду, оскільки більшість умовних операторів у шкідливих програмах мають простий характер.

– Ми також ігноруємо досить короткі сліди, щоб зменшити "шум" у створеній послідовності. Такі короткі послідовності, як правило, представляють "функції", додані різними незалежними розробниками шкідливих програм, щоб затушувати оригінальну програму зловмисного програмного забезпечення. Однак виявляється, що усунення таких послідовностей не впливає на точність виявлення нашого підходу.

У методиці викладеної у [32] ми не повинні генерувати трасування всіх критичних викликів API P, оскільки якщо траєкторія, сформована для критерію i, містить будь-який виклик API, наприклад, j в P, тоді нам не потрібно виконувати трасування на j, оскільки і вже містить усі операції трасування, які повинен містити j. Отже, тоді кількість слідів буде більшою, ніж, скажімо. Алгоритм генерації слідів поведінки показаний в алгоритмі 1. Це наближення відрізняється від нарізання програм, описаного Mark Weiser в [32]. Ці наближення потрібні для перетворення графічних блоків у лінійні послідовності, з деякою втратою семантики, але зберігаючи більшу частину програмної поведінки, необхідної для фіксації характеристик шкідливого програмного забезпечення.

**Algorithm 1**

Let $S$ be a set which contains $S_i$, where $S_i$ is the trace w.r.t tracing criterion $i$, and $API_g$ be the global API call monitoring.
Initially $API_g$ set contains a empty set
Set $API_g = \{\varnothing\}$
Traversing each API calls, from the set $P$
**For each** api $a$ in $P$ {
**if**$(a \neq API_g)$ {
$S_a$ = Backward_Trace(a)
$S = S \cup S_a$
$API_g = API_g \cup \{\text{API in } S_a\}$
}
**Repeat till** All the API calls are traversed from set $P$

Рисунок 4.1 - Алгоритм побудови поведінки

Сигнатура для класу шкідливого програмного забезпечення генерується шляхом обчислення LCS кожного загального (тобто критичного виклику API, знайденого в обох зразках) критичного трафіку виклику API з початкових зразків. Алгоритм генерації сигнатур наведено в рисунку 4.2.

**Algorithm 2**

Let $S$ be a set which contains $S_i$, where $S_i$ is the behavior trace w.r.t tracing criterion $i$, $S'$ be an another set which is similar to $S$, and $LCS_a$ be the longest substring for API call $a$. $S$ and $S'$ are set of two variants of the same malware class.
**For each** $S_a$ in $S$ {
**For each** $S'_a$ in $S'$ {
**if**$(S_a = S'_a)$ {
$LCS_a = \text{LCS}(S_a, S'_a)$
}
}
**Repeat till** All the trace components are traversed from set $S'$
**Repeat till** All the trace components are traversed from set $S$

Рисунок 4.2 - Алгоритм присвоєння сигнатур

### 4.4.3 Моніторинг мережі

Алгоритм виявлення шкідливого програмного забезпечення з використанням аналізу системної діяльності мережі може бути сформульований наступним чином:

– Ми контролюємо локально всі вихідні зв’язки із системи (ініційовані програмою чи користувачем).

– За допомогою Noriben ми створюємо профіль системної мережної діяльності, також враховуючи невдалі з'єднання.

– Наступні симптоми можна розглядати як ознаки роботи зловмисного програмного забезпечення в системі:

1. поява у профілі програм, відомих як такі, що не спілкуються в мережі,
2. поява у профілі нових нетипових портів призначення для добре відомих програм,
3. поява у профілі раніше невідомих програм,
4. виникнення у профілі численних помилок підключення, особливо для нетипових портів призначення.

## 4.5 Структура комплексу

Структура передбачає наявність кількох модулів: модуль вилучення даних, модуль аналізу даних, модуль динамічного тесту та головний модуль, у який вбудовано прийняття рішення. На рисунку 4.3 показано взаємодію модулів та процес виконання програми в загальному вигляді.

Процес детектування потенційно вірусного програмного забезпечення здійснюється за допомогою аналізу структури файла, що подається як вихідний для аналізу. Окремо, проводиться аналіз елементів формату PE-файлу та більш глибокий аналіз за допомогою набору методів статичного та динамічного аналізу поведінки  програмного забезпечення. Ці види аналізу реалізовано за допомогою окремих модулів на мові Python.

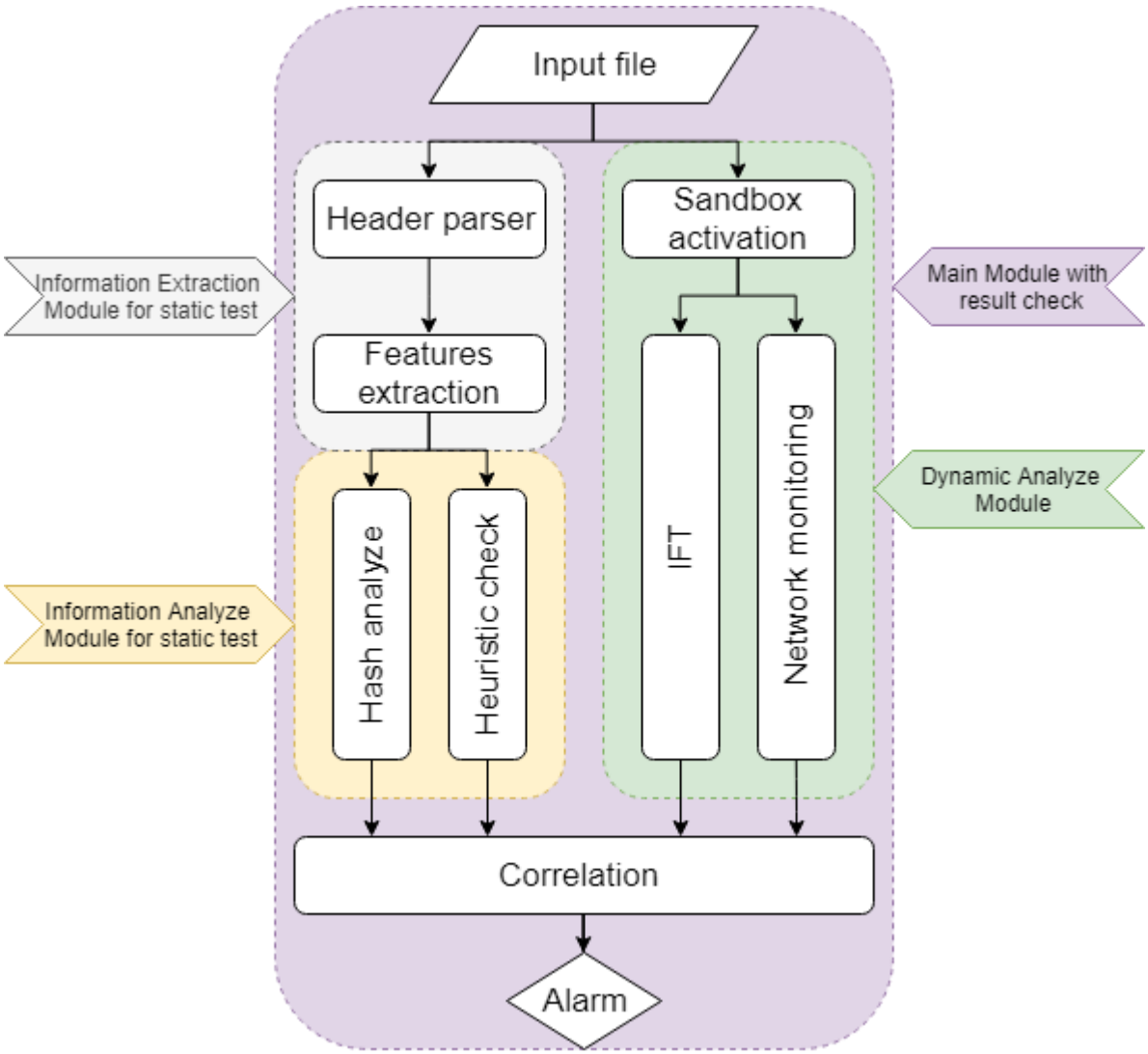


Рисунок 4.3 – Блок-схема взаємодії модулів програми.

## 4.6 Приклади роботи програми

Програма здатна виводити інформацію на всіх етапах роботи, проте кожний окремий блок потрібно вибирати власноруч. Для початку роботи потрібно – натиснути кнопку «Обрати файл», обрати файл, що знаходиться в тому самому каталогу, що і програма, після його під’єднання активуються додаткові елементи інтерфейсу для проведення аналізу.

У наступних екранних формах представлений вигляд графічного

інтерфейсу програми та приклади виконання аналізу (див. на рис. 4.4-4.7). У додатку Б наводиться повний огляд результатів роботи програми.


Рисунок 4. 4 – Приклад роботи програми при виведенні гешу.

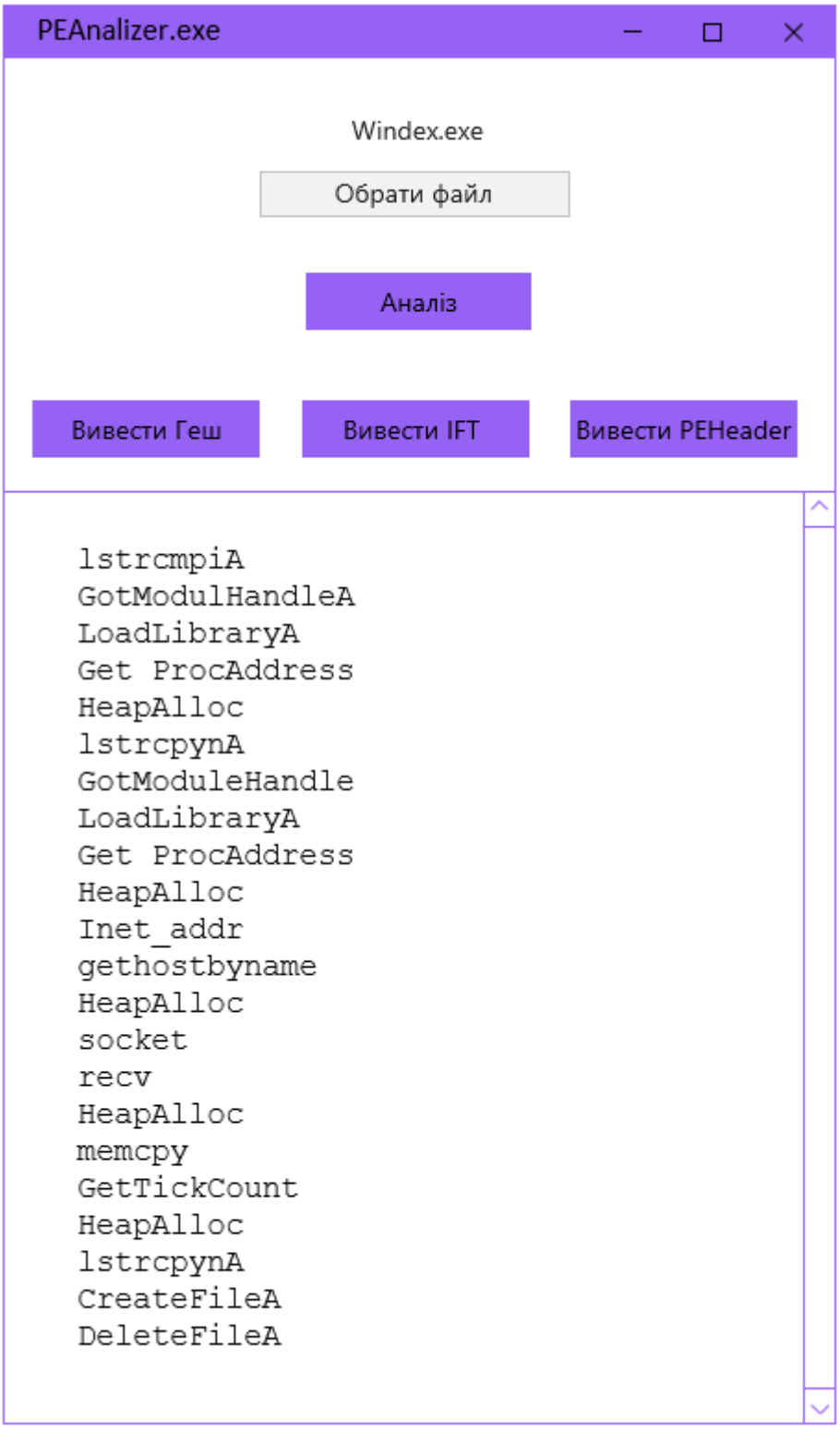


Рисунок 4.2 – Приклад роботи програми при виведенні IFT.

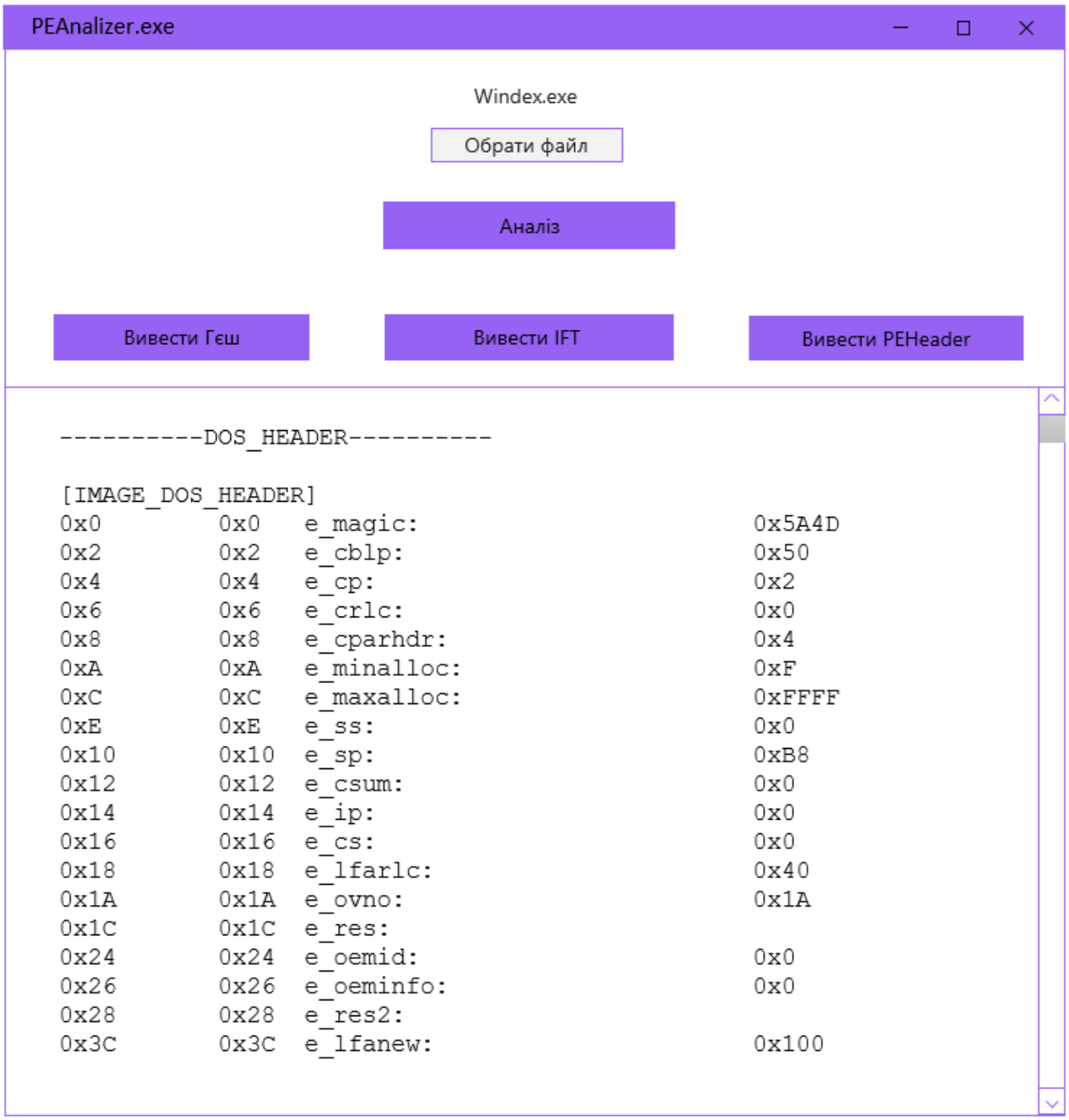


Рисунок 4.3 – Приклад роботи програми при активації пункту «Вивести PEHeader»

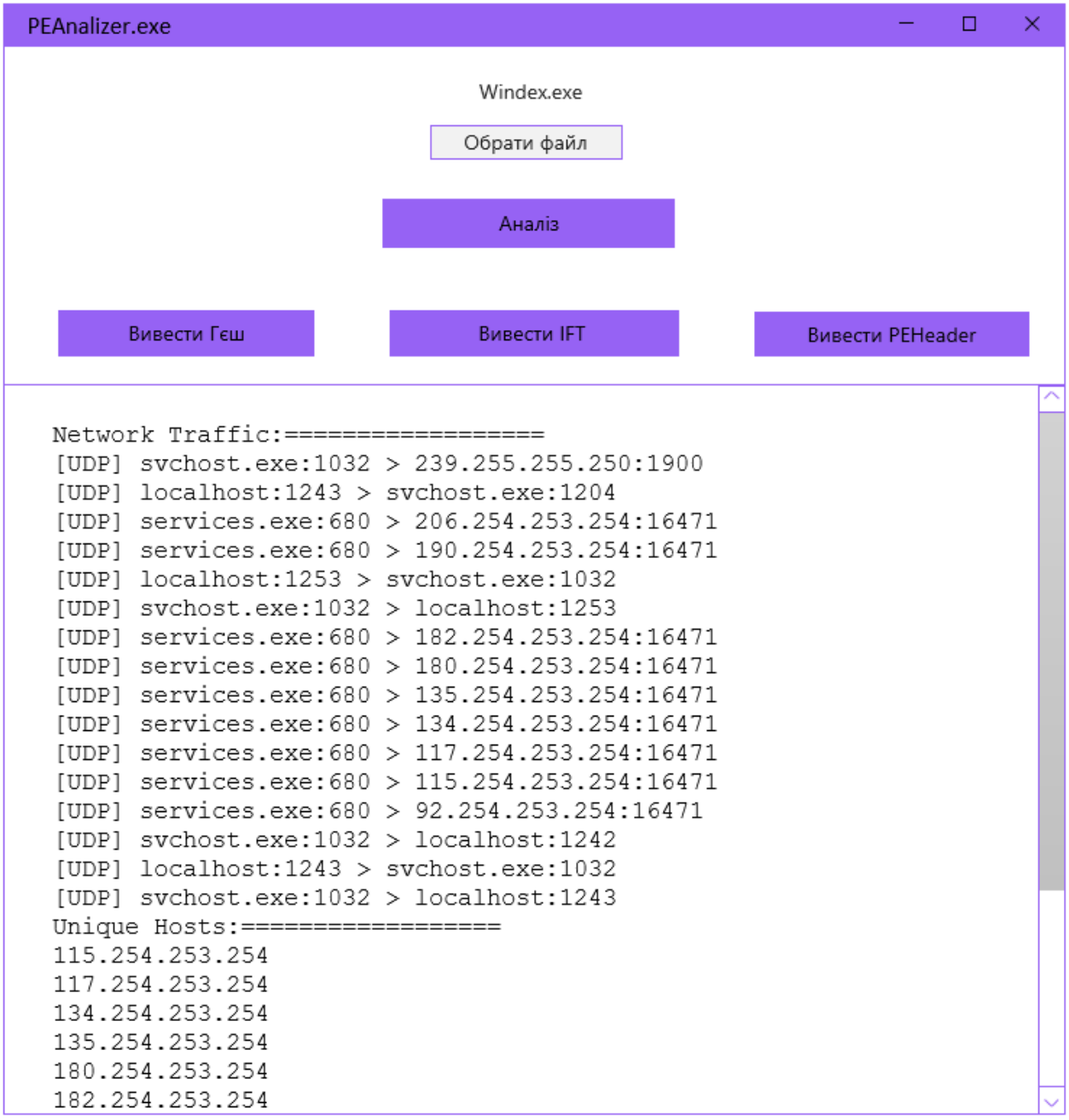


Рисунок 4.3 – Приклад роботи програми при аналізі мережного трафіку.

Таким чином, при розробці даного програмного забезпечення вдалося поєднати елементи різних видів аналізу в одному місці, що полегшує роботу аналітика у разі виникнення необхідності більш детального аналізу файлу за допомогою додаткових ознак, що були описані в попередніх розділах.

# 5 ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

## 5.1 Загальні відомості

При реалізації програмного продукту використовувались порівняльні ознаки шкідливого ПЗ, створений заголовний файл з функціями для витягу інформації заголовку, витягу ознак, також використовується додатковий файл для обробки. Також реалізовано графічний інтерфейс для взаємодії з користувачем.

В процесі роботи програма створює два файла: текстовий для представлення даних заголовку та файл з результатами оцінки. Усі компоненти написані на мові програмування Python. Розробка виконувалась за допомогою IDE JetBrains PyCharm Community Edition. Програма тестувалася на ПК з ОС Windows 10, процесор CPU 2020M 6*@3.80GHz 32 ГБ ОЗП.

## 5.2 Функціональне призначення та обмеження

Програма PEAnalizer.exe призначена для аналізу портативних виконуваних файлів, .

## 5.3 Технічні засоби, що використовуються

Учбова версія програми не накладає обмежень на технічні засоби, що використовуються. Технічним вимогам програми задовольняє ПК з мінімальними характеристиками.

## 5.4 Виклик та завантаження програми

Програма завантажується з жорстокого диску користувача. Для цього

небхідно перейти до виконавчого файлу з ім’ям PEAnalizer.exe і запустити його в роботу. Після завантаження всі дії виконуються за допомогою елементів управління на графічному інтерфейсі користувача.

## 5.5 Вхідні та вихідні дані

Вхідні дані – файл, що завантажується для проведення аналізу.

Вихідні дані – інформація про файл, віднесення або невіднесення його до шкідливого ПЗ.

## 5.6 Тестування

Щоб перевірити працездатність програми, було проаналізовано кілька зразків «чистих» файлів, взятих з офіційних сайтів та зразки шкідливого ПЗ з відкритої бази VirusTotal. Сервіс дозволяє отримати версії 126 (на сьогоднішній день) антивірусів для проведення дослідження. У додатку Б представлений приклад тестування роботи різних компонентів програми.

Результати тестів показали, що за наявності ознак наведених в додатку А, програма здатна чітко діагностувати небезпечну активність збоку потенційно шкідливого файлу, у разі співпадіння гешів з наявними з базі встановлювати вид шкідливого ПЗ. За відсутності прямого співпадіння результати програми є допомогою аналітику в прийнятті рішення щодо класифікації ПЗ.

# ВИСНОВКИ

В атестаційній роботі було розглянуто кілька методів пошуку шкідливого ПЗ через аналіз портативних виконуючих файлів, що є найпоширенішими джерелами враження комп'ютера. Також було досліджено формат PE-файлів, можливі сигнатури, що можуть вказати на шкідливість файлу, де-які сучасні статичні та динамічні методи виявлення ПЗ.

Статичний аналіз є першим кроком в аналізі шкідливих програм, він дозволяє отримати цінну інформацію з виконуваного файлу і допомагає при порівнянні та класифікації зразків шкідливого ПЗ. За допомогою розробленого програмного комплексу ми проаналізували різноманітні инструменти і методи, за допомогою яких можуть бути встановлені різні аспекти шкідливого коду без його виконання.

Крім того було проаналізовано, яку інформацію і як можна вилучити з файлу і чим вона може бути корисна. PE-файл має заголовок та інструкції виконання. З заголовку можливо дізнатись, які секції включає в себе файл, його атрибути та розмір та склад окремих директорій. Через інструкції ми відкриваємо порядок та набір функцій виконання, його поведінку в окремих випадках. В тому числі, графи, n-gram, де-які бінарні вказівки, аналіз через деобфускацію.

З методик побудови моделі програми найбільш оптимальною є граф. Проте він не дає вичерпну інформацію.

Найбільш перспективним серед існуючих методів є клас, що використовує машинне навчання, тобто система, яка сама себе вдосконалює, досліджуючи та аналізуючи нові файли контрольної групи, нові віруси.

Серед останніх можна виділити одновекторні та багатовекторні. Тобто, ті, що приймають рішення після прогону файлу через один класифікатор, та ті що мають кілька більш слабких класифікаторів.

В цій роботі ми розглянули найбільш прості методики: статистичного та

динамічного аналізу, які показали свою ефективність на практиці. Де перші досліджують конкретні атрибути та залежність їх значення від того, чи є файл шкідливим. Інші досліджують файл у процесі його виконання.

З боку динамічного аналізу було досліджено канали моніторингу: реєстр, журнал процесів, передача трафіку та інформаційні потоки, а також явні критерії або відправлення, що з великою ймовірністю вказують на шкідливість файлу. Крім цього розглянуто безпечне середовище та його налаштування для перевірки файлу.

Якщо програма несанкціоновано змінює ключі безпеки, або вимикає захисні механізми операційної системи, або надає самій собі забагато прав, то відповідь очевидна. При моніторингу файлової системи слід звертати увагу, розповсюдження та характер зміни. Через моніторинг трафіку можливо відстежувати зміни адрес спрямування та приховані канали передачі, що також може свідчити про злочинність програми.

Крім того, ми визначили алгоритм віднесення тієї чи іншого програми до злочинної через впровадження так званих флагів тревоги, тобто ми поділили всі ознаки на «бустери», що підвищують ймовірність шкідливості ПЗ, та «стоппери», що таку ймовірність знижують. Нейтральні показники ми не розглядали в силу їх незначності в контексті даної роботи.

Але, навіть використовуючи всі відомі нам способи та признаки, жодна програма не зможе бути захищеної від помилкової відповіді, іноді такі помилки можуть бути надто дорогими з точки боку витрачених ресурсів та пошкодженої пам’яті.

Нажаль, немає універсального рішення, є шанс віднести чистий файл до списку шкідливих, випадково виконати лікування та втратити важливу інформацію. Тож, у таких випадках рекомендується перевіряти файли самостійно.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Jinrong Bai, Junfeng Wang, Guozhong Zou. A Malware Detection Scheme Based on Mining Format Information / Jinrong Bai – The Scientific World Journal. 2014, Vol. 2014.

2. Eureka: A Framework for Enabling Static Malware Analysis, Sharif M. [et al.]. Recent Advances in Intrusion Detection, Lecture Notes in Computer Science., 2008, Vol. 5283, P. 481-500.

3. Microsoft Corporation, PE Format specification. [Електронний ресурс] Режим доступу:

https://msdn.microsoft.com/library/windows/desktop/ms680547(v=vs.85).aspx?id=19509 - 13.12.2020 - Загл. с экрана.

4. Касперски К. Путь воина – внедрение в pe/coff-файлы, 2004, [Електронний ресурс] Режим доступу: http://samag.ru/archive/article/297 13.12.2020 - Загл. с экрана.

5. Microsoft Corporation. What is a DLL. 2007, [Електронний ресурс] Режим доступу: https://support.microsoft.com/kb/815065/EN-US - 13.12.2020 - Загл. с экрана.

6. Manjunath. Malware images: Visualization and automatic classification, Nataraj L. [et al.], In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pp. 41-47, ACM, 2011.

7. Novel Feature Extraction, Selection and Fusion for Effective Malware Family lassification, Ahmadi M. [et al.], In Proceedings of the 6 ACM Conference on Data and Application Security and Privacy, Pp. 183-194, ACM, 2016.

8. File Signatures [Електронний ресурс] Режим доступу: https://filesignatures.net - 13.12.2020 - Загл. с экрана.

9. Алексей Пустыгин. Построение универсального представления графа потока управления для статического анализа исходного кода - Томск: Издательство ТУСУР, 2012. - 142 с. – С. 61-63.

10. Зубов М.В., Пустыгин А.Н., Старцев Е.В. Применение универсальных промежуточных представлений для статического анализа исходного программного кода. - Томск: Издательство ТУСУР, 2013. - 142 с. - С 64-69.

11. Christodorescu, S. Jha. Static Analysis of Executables to Detect Malicious Patterns. - In Proceedings of the 12th USENIX Security Symposium, 2003.

12. П. Кравченко. О возможностях применения системного программного обеспечения LLVM для научных и прикладных исследований в области информационной безопасности. - Известия Южного федерального университета - Технические науки №76. - 2012. - Т. 129, № 4. - С.70-73.

13. R. Cytron, J. Ferrante, B. Rosen and oth. Efficiently computing static single assignment form and the control dependence graph - ACM Transactions on Programming Languages and Systems (TOPLAS). - 1991. - V. 13 No 4. - P. 451-490.

14. Grim, J. Computer-aided evaluation of screening mammograms based on local texture models IEEE Transactions on Image Processing. - 2009. - Vol. 18, №. 4. - pp. 765-773.

15. Chen T., C. Guestrin C. XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. -2016. - P. 785-794.

16. A survey on Heuristic Malware Detection Techniques / Bazrafshan Z. [et al.] // 5th Conference on Information and Knowledge Technology. - 2013. - P. 113-120.

17. Ucci D., Aniello L., Baldoni R. Survey on the Usage of Machine Learning Techniques for Malware Analysis // arXiv preprint arXiv:1710.08189. — 2018.

18. A verifiable SSA program representation for aggressive compiler optimization V. Menon, N. Glew, B. Murphy and oth. // POPL '06 Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. - 2006. - P. 397-408.

19. Тарануха В.Ю. Модифікація n-грамної моделі, заснованої на

класах, для розпізнавання слов'янських мов /Тарануха В.Ю.// Вісник Київського національного університету імені Тараса Шевченко Серія: фізико-математичні науки. - 2014. - №1. - С.193-196.

20. Graphviz [Електронний ресурс] Режим доступу: http://www.graphviz.org 13.12.2020 - Загл. с экрана.

21. Logilab-Common [Електронний ресурс] Режим доступу: https://www.logilab.org/project/logilab-common - 13.12.2020 - Загл. с экрана.

22. Корабльов Н.М. Виявлення і аналіз шкідливих програм з використанням мультиагентного підходу – Харківський національний університет радіоелектроніки, Харків, Збірник наукових праць Харківського університету Повітряних Сил. - 2015. - №42. - С.34-38.

23. Войтович О. П. Особливості дослідження ознак шкідливого програмного забезпечення без наявності вихідних кодів/ О. П. Войтович, В. О. Вітюк, В. А.Каплун // Інформаційні технології. - №3. - С.4-7.

24. Blake Hartstein Matthew Richard Michael Ligh, Steven Adair. Malware Analyst's Cookbook: Tools and Techniques for Fighting Malicious Code. John Wiley & Sons. - 2010. P.450.

25. William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: an informationflow tracking system for realtime privacy monitoring on smartphones. In Proceedings of the 9th USENIX conference on Operating systems design and implementation, OSDI'10, USENIX Association. Berkeley, CA, USA, - 2010. - P. 1–6.

26. Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou, and Youfeng Wu. Lift: A low-overhead practical information flow tracking system for detecting security attacks. In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39, IEEE Computer Society, Washington, DC, USA, - 2006. - p.135-148.

27. Asia Slowinska and Herbert Bos. Pointless tainting. Evaluating the practicality of pointer tainting, 2009. - 345 p.

28. Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In Proceedings of the IEEE Symposium on Security and Privacy, - 2010. - p. 156-161.

29. Data Breach Investigations Report. Verizon [Електронний ресурс] Режим доступу: http://www.verizonenterprise.com/DBIR/2013/- 13.12.2020 - Загл. с экрана.

30. Fortinet 2013 Cybercrime Report. Fortinet. [Електронний ресурс] Режим доступу:http://www.fortinet.com/resource_center/whitepapers/cybercrime_report_on_botnets_network_security_strategies.html - 13.12.2020 - Загл. с экрана.

31. 2013 Information Security Breaches Survey. [Електронний ресурс] Режим доступу: https://www.gov.uk/government/publications/information-security-breaches-survey-2013-technical-report - 13.12.2020 - Загл. с экрана.

32. The Demise in Electiveness of Signature and Heuristic Based Antivirus. [Електронний ресурс] Режим доступу: http://docs.media.bitpipe.com/io_10x/io_102267/item_632588/2013-01-9_the_demise_of_signature_based_antivirus_final.pdf - 13.12.2020 - Загл. с экрана.

33. . Defeating Advanced Persistent Threat Malware. Infoblox. [Електронний ресурс] Режим доступу: http://securematics.com/sites/default/files/secure/default/files/pdfs/infoblox-whitepaper-defeating-apt-malware.pdf - 13.12.2020 - Загл. с экрана.

34. ENISA Threat Landscape 2013. Overview of current and emerging cyber-threats. [Електронний ресурс] Режим доступу: https://www.enisa.europa.eu/activities/risk-management/evolving-threat-environment/enisa-threat-landscape-2013-overview-of-current-and-emerging-cyber-threats - 13.12.2020 - Загл. с экрана.

35. IBM X-Force 2013 Mid-Year Trend and Risk Report. IBM [Електронний ресурс] Режим доступу: http://www-03.ibm.com/security/xforce/downloads.html - 13.12.2020 - Загл. с экрана.

36. The Advanced Cyber Attack Landscape. FireEye, Inc. [Електронний

ресурс] Режим доступу: http://www.security-inder.ch/fileadmin/dateien/pdf/studien-berichte/fireeye-advanced-cyber-attack-landscape-report.pdf - 13.12.2020 - Загл. с экрана.

37. Miroslaw Skrzewski. System Network Activity Monitoring for Malware Threats Detection. Silesian University of Technology Conf.- 2013. - P.35-44.

38. David Harley. Heuristic Analysis- Detecting Unknown Viruses. - ESET - Slovak Rebublic, 2009. - 24p.

39. Монаппа К. А. Анализ вредоносных программ / Монаппа К. А. - М.: ДМК, 2019. - 453 с.