МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
____________ С.В.Казмірчук
«_____»____________2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА
ЗА СПЕЦІАЛЬНІСТЮ 125 «КІБЕРБЕЗПЕКА»

Тема: «Засоби виявлення шкідливого коду з використанням механізмів машинного навчання»

Виконавець: студент групи БІ 201 Мз________________________Скальська В.І,

Керівник: д.т.н., професор___________________________________Толюпа С.В.

Нормоконтролер: ______________ (підпис) ________________ (П.І.Б.)

Київ 2020

ВСТУП

Інтернет став невід’ємною частиною нашого повсякденного життя. 59% населення світу станом на 2020 рік підключено до Інтернет [1]. Ми використовуємо його для банківської діяльності, спілкування, розваг, покупок та різної іншої комерційної та некомерційної діяльності. Хоча Інтернет зробив наше життя зручнішим, він також піддав нас ризику нападу. Незаконні користувачі використовують шкідливі програми для вчинення фінансових шахрайств або викрадення приватної/конфіденційної інформації у законних користувачів. Кількість повідомлених атак щороку збільшується. Av-Test повідомляє про виявлення більш ніж 350 000 зразків шкідливих програм щодня [2]. Те, що почалося як захоплення техно-ентузіастів та дослідників, зараз перетворилося на міжнародне співтовариство висококваліфікованих програмістів, мотивованих легкими прибутками. Зараз вона стала триліонною галуззю [3], де інструменти для злому продаються та купуються, як і легальне програмне забезпечення. Зловмисні розробники навіть пропонують технічну підтримку та обслуговування клієнтів.

Поширення зловмисного програмного забезпечення з постійно зростаючою швидкістю становить серйозну загрозу в світі з Інтернетом. Виявлення та класифікація зловмисних програм стала однією з найважливіших проблем у сфері кібербезпеки. З постійно зростаючим ризиком кібератак, увага лежить на дослідниках безпеки для розробки нових методів виявлення шкідливих програм та розробки нових механізмів захисту проти них. В останні роки спостерігається швидке зростання кількості файлів, поданих антивірусним компаніям на аналіз, тому аналізувати функціональність кожного файлу вручну стало дуже важко. Розробники зловмисних програм успішно розробляють зразки, що ухиляються від методів виявлення на основі підписів. Більшість переважаючих методів статичного аналізу містять інструмент для розбору файлу. Весь процес аналізу стає залежним від ефективності цього інструменту, і якщо він погано спрацьовує, задача виявлення становиться дуже важкою. Більшість методів динамічного аналізу передбачають двійковий файл, який запускають в віртуальному середовищі, щоб вивчити його поведінку. Зловмисники

легко обходять цю перевірку, приховуючи шкідливі дії файлу коли він запускається всередині віртуального середовища. У цій роботі було досліджено нову методику представлення шкідливих програм як зображень. Потім було використано існуючі методи навчання нейронних мереж для створення класифікатора для виявлення серед незнайомих файлів шкідливих програм. Так як файли представляються у вигляді зображення, процес аналізу стає незалежним від будь-якого інструменту, і це робить процес менш трудомістким. За допомогою модифікованої архітектури популярної нейромережі, у цій дипломній роботі вдалося отримати точність виявлення шкідливих програм 96,74% на тестовому наборі даних.

РОЗДІЛ 1
АНАЛІЗ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Зловмисне програмне забезпечення

Зловмисне програмне забезпечення – це спеціально написані програми, створені для виконання деяких шкідливих дій, таких як отримання доступу до машини жертви, налаштування реєстратора ключів, встановлення бекдорів або викрадення цінної приватної інформації. Англійською, malware – це скорочення від malicious software (шкідливе програмне забезпечення). Зловмисне програмне забезпечення можна визначити як "будь-яке програмне забезпечення, яке завдає шкоди користувачеві, комп'ютеру чи мережі" [4]. Зловмисне програмне забезпечення може бути у вигляді скрипту, виконуваного файлу або будь-якої іншої програми. Загалом, шкідливі програми можуть бути класифіковані як черв'яки, віруси, трояни, Ransomwares, Adwares, Spywares, боти, PUPs 1, Rootkits, Scarewares та інші шкідливі програми. Майже кожна атака, про яку ми чуємо, як, наприклад, атака зловмисного програмного забезпечення Mirai [5] та атака WannaCry Ransomware [6], передбачає якесь зловмисне програмне забезпечення. Зі зростанням цифрового сліду складність цих атак також зростає.

- *Черв`яки*: це мережеві віруси, які мають здатність поширюватися по мережі шляхом реплікації [7]. Вони не змінюють файли користувача або системних файлів, але вони перебувають у головній пам'яті і продовжують реплікацію та розповсюдження себе. Зважаючи на всі ці небажані дії, вони в кінцевому підсумку використовують усі наявні ресурси, завдяки чому система та мережа не реагують [8]. Хоча вони не є безпосередньо шкідливими, вони, як правило, використовуються для введення корисних навантажень, які можуть бути іншими шкідливими програмами, такими як троянські коні, віруси, бекдори тощо. Деякі відомі напади хробаків: черв'як Морріс, черв'як “любовний лист” і червоний кодекс.

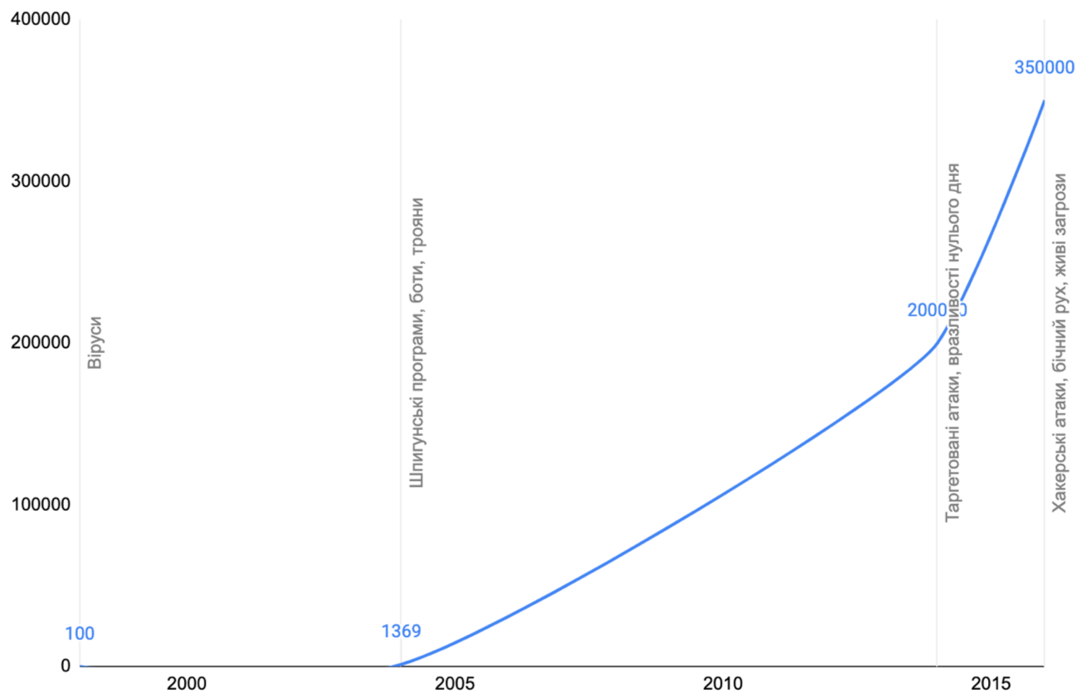

Рисунок 1.1 – Зростаюча складність шкідливого програмного забезпечення [61]

- *Віруси:* Вірус – це програмне забезпечення, яке модифікує інші програми та приєднує себе до їх коду. Як і справжні віруси, вони не можуть запускатись самостійно, і виконують шкідливу діяльність лише тоді, коли виконується заражена програма [9]. Вірус поширюється, заражаючи файли на хост-машині, а також спільні файли по мережі. У процесі зараження користувацьких файлів віруси, як правило, пошкоджують їх, таким чином, користувач може зазнати втрати даних. Одні з найвідоміших комп'ютерних вірусів - вірус Кріпера та Мелісса.
- *Трояни*: Трояни, як випливає з назви, є шкідливими програмами, які змушують користувача встановлювати їх, роблячи вигляд, що вони є корисним або доброякісним програмним забезпеченням. Після виконання вони починають виконувати свою шкідливу діяльність на задньому плані. Зазвичай вони розповсюджуються за допомогою деяких методів соціальної інженерії, таких як вкладення електронної пошти, підроблені спливаючі вікна або спам-посилання. Після встановлення на комп'ютер жертви вони можуть завантажувати інші шкідливі

програми, створювати на задньому плані або заражати системні файли. Деякі з відомих троянів - це Зевс, Темна комета та Shedun.

- *Програми-вимагачі (ransomware):* Нещодавно напади програм-вимагачів різко зросли, як приклад: недавні напади TorrentLocker, WannaCry та Petya [10]. Вони або шифрують усі користувацькі файли, або обмежують доступ користувача до своїх файлів. Обмежуючи користувачів, зловмисник змушує їх платити, щоб отримати доступ до своїх файлів. Зловмисне програмне забезпечення, як правило, здійснюється за допомогою троянського пристрою, який змушує користувача завантажити корисне навантаження, що містить програму-вимагача, у свою систему.
- *Потенційно небажані програми (PUP):* це програмне забезпечення, яке користувач може сприймати як небажане. Таке програмне забезпечення може використовувати реалізацію, яка може порушити конфіденційність або послабити безпеку комп'ютера. Компанії часто поєднують потрібну програму із програмою-обгорткою та можуть запропонувати встановити небажану програму, в багатьох випадках не надаючи чіткого способу відмови.
- *Рекламні програми (adware):* рекламні програми показують тареготовану рекламу зараженому хосту, що генерує для зловмисника дохід. Зазвичай вони встановлюються разом із PUP. Вони не можуть завдати шкоди жертві, але у рекламного ПЗ можуть бути вразливісті, яка можуть використовуватись зловмисником.
- *Шпигунські програми (spyware):* Шпигунські програми, як випливає з назви, зазвичай використовуються для шпигування за користувачами. Вони є шкідливими програмами, які спрямовані на збір приватної інформації із зараженого хоста, а потім надсилання цієї інформації зловмиснику. Вони використовуються для відстеження діяльності користувача, крадіжки паролів, реквізитів банківського рахунку тощо.
- *Боти*: Боти - це програмне забезпечення, яке дозволяє зловмиснику віддалено отримувати доступ до хост-машини. Скупчення ботів, кероване одним сервером, називається Botnet. Ботнет може використовуватися для запуску DDoS-атаки, запуску спам-кампаній тощо.

● *Руткіти (Rootkits):* Руткіти використовуються для приховування себе чи інших шкідливих програм від жертви [4]. Їх виявлення є важким, оскільки вони, як правило, працюють з привілеями root, що дозволяє йому підривати системні журнали та програмні засоби виявлення вторгнень.

● *Програми-лякалки (Scarewares):* програми-лякалки намагаються обманути користувачів придбати щось непотрібне, лякаючи їх. Вони використовують методи соціальної інженерії, такі як показ помилкових попереджувальних повідомлень, помилкові попередження про порушення безпеки, щоб створити помилкове відчуття загрози.

● *Бекдори (backdoors):* бекдори дозволяють отримати доступ до системи через віддалений термінал. Застосовується як в зловмисних, так і в нешкідливих цілях. Популярним інструментом Backdoor є RAT, термінал віддаленого доступу.

Шпигунські програми, рекламне програмне забезпечення та PUP іноді класифікуються як сірі програмні засоби. Сірі програми — це небажані програми, які не завдають шкоди зараженому хосту, але можуть споживати системні ресурси, що погіршують продуктивність, і можуть містити деяку вразливість, яка потенційно може стати точкою входу для зловмисника.

Хоча всі антивірусні компанії дотримуються суворих конвенцій щодо іменування шкідливих програм, не існує єдиного стандартну для іменування шкідливих програм між різними компаніями. Тож цілком можливо, що різні антивірусні системи можуть присвоювати одному і тому ж файлу зловмисних програм різні імена. У цій дипломній роботи буде для прикладу розглянута конвенція іменування, яку дотримується антивірус Microsoft. Антивірус Microsoft використовує схему іменування зловмисного програмного забезпечення, запропоновану організацією Computer Antivirus Researchers (CARO).

Наприклад, Trojan:JS/BlacoleRef.W, Worm:Win32/Allaple.A і т.п. Згідно з умовами іменування, зображеними на рисунку 1.2. З імені Trojan: JS/BlacoleRef.W, ми можемо легко зрозуміти, що Trojan – це тип, JS – платформа, BlacoleRef – сімейство шкідливих програм, а W – варіант.

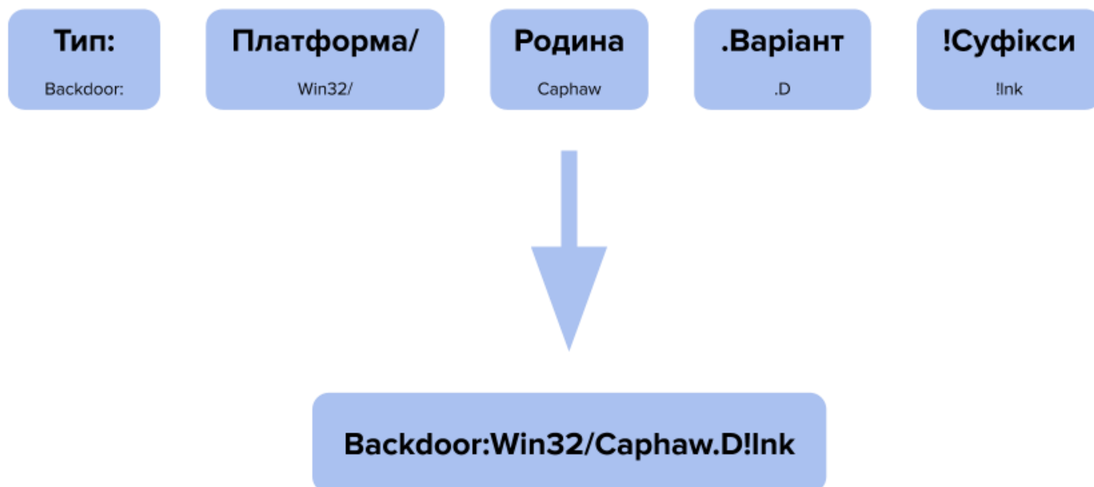

Рисунок 1.2 – Схема іменування шкідливих програм, що застосовується Microsoft

«Тип» в схемі іменування вказує тип зловмисного програмного забезпечення, тобто основну діяльність, яку виконує зловмисне програмне забезпечення. Основні типи зловмисних програм були обговорені в попередньому розділі, такі як Trojan, Backdoor, Virus тощо.

«Платформа» означає платформу, на якій зловмисне програмне забезпечення призначене для виконання. Платформа може включати операційну систему, таку як Windows, MacOS, Android тощо. Вона може також включати мову програмування, як HTML, JS тощо.

«Сімейство» шкідливих програм визначається на основі структурної подібності між різними шкідливими програмами. Зловмисне програмне забезпечення, що належить до однієї родини, має схожість коду, що допомагає у створенні загальних методів виявлення та видалення. Це найважливіше поле в конвенції про іменування.

«.Варіант» використовується для розмежування різних версій однієї сім'ї.

«!Інформація» використовується для визначення деякої додаткової інформації про шкідливе програмне забезпечення, наприклад, пошкоджене, стиснене, упаковане тощо. Це може також містити будь-яку іншу інформацію, яку аналітик вважає важливою характеристикою цієї шкідливої програми. Наприклад, lnk може бути доданий, щоб вказати, що зловмисне програмне забезпечення є файлом ярлика.

Ім’я зловмисного програмного забезпечення повинно містити родину та варіант, тоді як інші поля можуть бути пропущені. Якщо відомий лише один варіант певної родини, поле варіанту також дозволяється пропускати.

1.2 Поширені методи аналізу шкідливого програмного забезпечення

Аналіз зловмисного програмного забезпечення – це процес аналізу бінарного файлу, щоб зрозуміти його функціонування, а потім розробити методи ідентифікації цього та інших подібних файлів [4].

Аналіз зловмисного програмного забезпечення має на меті з'ясувати дії, що виконуються шкідливим програмним забезпеченням, а потім розробити методи нейтралізації цих дій та запобігання подальшим зараженням. Це не тільки відповідальність антивірусних компаній, але й адміністраторів безпеки різних організацій, яким потрібно виконувати аналіз шкідливих програм. Системним адміністраторам необхідно проводити аналіз зловмисного програмного забезпечення, щоб з’ясовувати ступінь шкоди для системи. Адміністраторам мережі потрібно проводити аналіз, щоб з'ясувати, в якій мірі була пошкоджена мережа, а потім спробувати виправити її. Академічний дослідник аналізує зразки зловмисних програм, щоб зрозуміти, як вони поводяться, та вивчає методи, які використовуються розробниками зловмисних програм, для підвищення безпеки існуючої інфраструктури.

Аналіз зловмисного програмного забезпечення використовується як для виявлення шкідливих програм, так і для класифікації шкідливих програм. Виявлення зловмисного програмного забезпечення - це процес маркування двійкового виконуваного файлу як доброякісного або зловмисного. Таким чином, це перший крок аналізу шкідливих програм. Лише після того, як спеціалісти дізнаються, що файл є шкідливим програмним забезпеченням, вони переходять до подальшого аналізу. Інше схоже поняття, класифікація зловмисного програмного забезпечення – це процес з'ясування класу або родини, до якого належить дане зловмисне програмне забезпечення. Аналіз зловмисного програмного забезпечення може проводитися

двома способами: статичним та динамічним. Статичний аналіз відноситься до методики, яка спрямована на аналіз файлів без її виконання, тоді як в динамічному аналізі ми виконуємо файли для визначення їх поведінки.

Статичний аналіз зловмисного програмного забезпечення включає огляд бінарного файлу повністю, не виконуючи його. Цей метод може бути застосований до різних типів представлених файлів. Якщо фактичний код присутній, то статичний аналіз становиться простою задачею. Навіть якщо код недоступний, ми можемо вивчити необроблений шістнадцятковий код або спершу розібрати двійковий файл, а потім вивчити асемблерний код. Статичний аналіз може передбачати вивчення друкованих рядків в програмі, вивчення заголовка файлу, розбирання програми, пошук послідовностей байтів тощо.

Поширені методи статичного аналізу:

1. Використати будь-який доступний антивірус, щоб перевірити, чи файл вже був позначений як шкідливий. Це можна зробити, взявши відбиток файлу. MD5 або SHA-256 хеш може служити відбитком для відрізнення файлу від інших файлів [4].

2. Отримати всі друковані символи з двійкового файлу за допомогою утиліти strings. Це може включати деяку інформацію про стан або повідомлення про помилки, за допомогою яких ми можемо зробити висновок про функціональність бінарного файлу [4].

3. Вивчити метадані, наявні у заголовку файлу. Метадані можуть включати інформацію, таку як магічне число, час компіляції, імпорт системних бібліотек, використовувані іконки тощо. Магічне число може допомогти нам визначити тип файлу, тип імпортованих бібліотек може допомогти нам передбачити поведінку зловмисного програмного забезпечення [4].

4. З'ясувати, чи файл упакований чи ні, використовуючи інструмент ідентифікації пакера, наприклад PEiD. PEiD управляє базою даних підписів популярних пакувальників, і якщо файл було запаковано за допомогою одного з них, він може повідомити нам. Якщо файл упакований стандартним пакетом, таким як UPX Packer, то спочатку слід розпакувати упакований бінарний файл, перш ніж далі аналізувати його [4].

5. Провести поглиблений аналіз функціональності зловмисного програмного забезпечення, спершу розібравши машинний код на асемблерний код, а потім вивчивши опкоди. IDA Pro або подібний інструмент може використовуватися для генерації коду асемблера (процесу розбірки/дизасемблінга), а потім OllyDbg або подібний інструмент можна використовувати для його відладки [4].

6. Спроба декомпіляції, щоб отримати код високорівневої мови з машинного коду. Він не відновлює вихідний код, оскільки при компіляції є втрата інформації [4], але ми можемо отримати деяке уявлення про двійковий код із відновленого коду.

Основна перевага статичного аналізу полягає в тому, що він не несе ризику поширення інфекції під час перевірки зловмисного програмного забезпечення. Статичний аналіз є всеосяжним, оскільки він, як правило, охоплює всі можливі шляхи виконання. Єдиний недолік у ньому полягає в тому, що це може забирати багато часу і вимагає проведення аналізу експертом. Статичний аналіз може зірватися, якщо двійковий файл зашифрований, стиснутий, обфускований або упакований. Серед популярних методів ухилення від статичного аналізу можна виділити:

● *Поліморфізм* загалом означає "змінювати зовнішній вигляд". У цій техніці розробник зловмисного програмного забезпечення змінює двійковий файл таким чином, що він може ухилятися від методів виявлення, які використовують підпис-відбиток. Для виготовлення поліморфних копій бінарних файлів використовують пакувальники або криптери. Це програмне забезпечення спочатку пакує двійковий файл, а потім додає розпорядок розпакування вгорі упакованого коду. Цей новий двійковий файл не можна виявити, використовуючи відбиток, оскільки його підпис буде різним. Коли двійковий файл буде виконаний, програма упаковки спочатку розпакує запакований бінарний файл лише в пам'яті, а потім передасть послідовність виконання до початку розпакованого коду. Таким чином, навіть якщо файл було змінено, він буде продовжувати виконувати ту саму функцію. Найпростіший спосіб виявити поліморфний вірус - шукати підписи в пам'яті, оскільки оригінальне зловмисне програмне забезпечення після розпакування зберігається в пам'яті.

● *Метаморфізм* шкідливих програм означає самомодифікацію бінарних виконуваних файлів. Зловмисне програмне забезпечення змінює власний двійковий

код кожного разу, коли він розповсюджується або запускається. Це відрізняється від поліморфізму, оскільки поліморфне зловмисне програмне забезпечення не може переписати власний код. Отриманий двійковий файл буде виконувати ті ж дії, що і оригінальний зловмисне програмне забезпечення, але його двійкове представлення буде зовсім іншим. Прості методи метаморфізму включають додавання різної тривалості No Operation (NOP) інструкції, зміну потоку управління за допомогою стрибків, перестановку використовуваних регістрів тощо.

Динамічний аналіз шкідливого програмного забезпечення включає виконання двійкового файлу та спостереження за його поведінкою, щоб визначити, чи є це зловмисне програмне забезпечення чи ні. Зловмисне програмне забезпечення виконується у пісочниці (sandbox) або віртуальній машині, так що наслідки зловмисного програмного забезпечення можуть бути стримані. Віртуальні машини дають нам можливість робити знімки стану перед виконанням зловмисного програмного забезпечення, і коли аналіз закінчиться, ми можемо легко повернутися до збереженого стану.

Поширені методи динамічного аналізу:

- *Моніторинг системних викликів* – системні виклики допомагають програмі взаємодіяти з операційною системою. Функція, як правило, пов'язана з певним завданням, наприклад, функція сортування використовується для сортування. Таким чином, моніторинг системних викликів, використовуваних виконуваним файлом, може дати нам інформацію про його функції.
- *Простеження інструкцій* – іншим способом відстеження діяльності зловмисного програмного забезпечення є простеження інструкцій. Він записує послідовність, у якій інструкції виконувалися зловмисними програмами.
- *Відстеження діяльності зловмисних програм* – дії, які виконуються під час виконання процесу на інших процесах і файлах, можна відстежувати за допомогою інструментів, таких як Process Monitor. Ми можемо відслідковувати створення нових процесів та список файлів, змінених шкідливим програмним забезпеченням. Інший інструмент, Process Explorer, допомагає нам відстежувати зміни в реєстрі, файлові дії та нові дочірні процеси, створені під час виконання.

- *Програма відстеження реєстру* – зловмисні програми часто намагаються змінити системний реєстр, щоб змінити поведінку системи. Отже, щоб відстежувати будь-які зміни в реєстрі, ми повинні зробити його знімок перед виконанням бінарного файлу, а потім порівняти початковий знімок з реєстром після закінчення виконання.
- *Відстеження мережевої активності* – ми можемо імітувати підроблену мережу для контролю за мережевою активністю. Деякі зловмисні програми показують шкідливу активність під час їх підключення до мережі. Таким чином, такі інструменти, як INetSim, можна використовувати для імітації підключення до Інтернету, оскільки надання реального доступу до Інтернету в пісочниці може бути ризикованим.
- *Відстеження модифікацій файлів* – файли-приманки (goat files) спеціально створені для відстеження модифікацій файлів, які виконуються зловмисними програмами. Зазвичай вони містять лише інструкцію NOP, тому, якщо зловмисне програмне забезпечення намагається заразити його, то зміни у файлі можна легко визначити [7].
- *Налагодження (Debugging)* – за допомогою налагоджувача ми можемо відстежувати зміни, внесені на кожному кроці виконання програми. Також налагоджувач може допомогти нам визначити альтернативні шляхи виконання, які можуть залишатися невивченими при простому виконанні.

Переваги та недоліки динамічного аналізу

Оскільки під час динамічного аналізу виконується зловмисне програмне забезпечення, це допомагає протидіяти методам обходу захисту, що перемагають статичний аналіз, таких як упаковка та обфускація.

Основними обмеженнями динамічного аналізу є те, що зазвичай він може відстежувати лише один шлях виконання, тому він страждає від неповного покриття коду. Крім того, якщо зловмисне програмне забезпечення здатне виявити пісочницю, воно може змінити свою поведінку, приховавши шкідливі дії, тим самим ухиляючись від аналізу [11]. Ще одне обмеження полягає в тому, що якщо в середовищі пісочниці є якась помилка або вразливість, то зловмисне програмне забезпечення може заразити хост-комп'ютер або інші комп'ютери в мережі.

Традиційний інструмент виявлення зловмисного програмного забезпечення, антивірусне програмне забезпечення, використовують підписи для ідентифікації шкідливих програм. Аналітики компаній, що займаються інформаційною безпекою, аналізують отриманий ними зразок і розробляють унікальні підписи, що ідентифікують цей зразком. Потім антивірусна база даних оновлюється цими підписами. Після цього в процесі сканування системи на кожній користувацькій машині порівнюється підпис кожного файлу на машині з базою даних підписів. Якщо підпис певного файлу збігається з будь-яким із підписів антивірусної бази, то цей файл позначається як зловмисне програмне забезпечення. Завдяки високій швидкості перевірки та високій точності пошуку відомих загроз методи, засновані на підписах, є ефективними, але ці методи не справляються із обфускацією коду та не в змозі ефективно визначити новітні загрози. Щодня на аналіз VirusTotal [12] надсилається понад півтора мільйона файлів [12]. Оскільки обсяг зразків, отриманих антивірусними компаніями для аналізу, дуже великий, робити аналіз вручну неможливо. Тому процес аналізу потребує автоматизації, що допоможе зменшити кількість файлів, які потребують ручного аналізу.

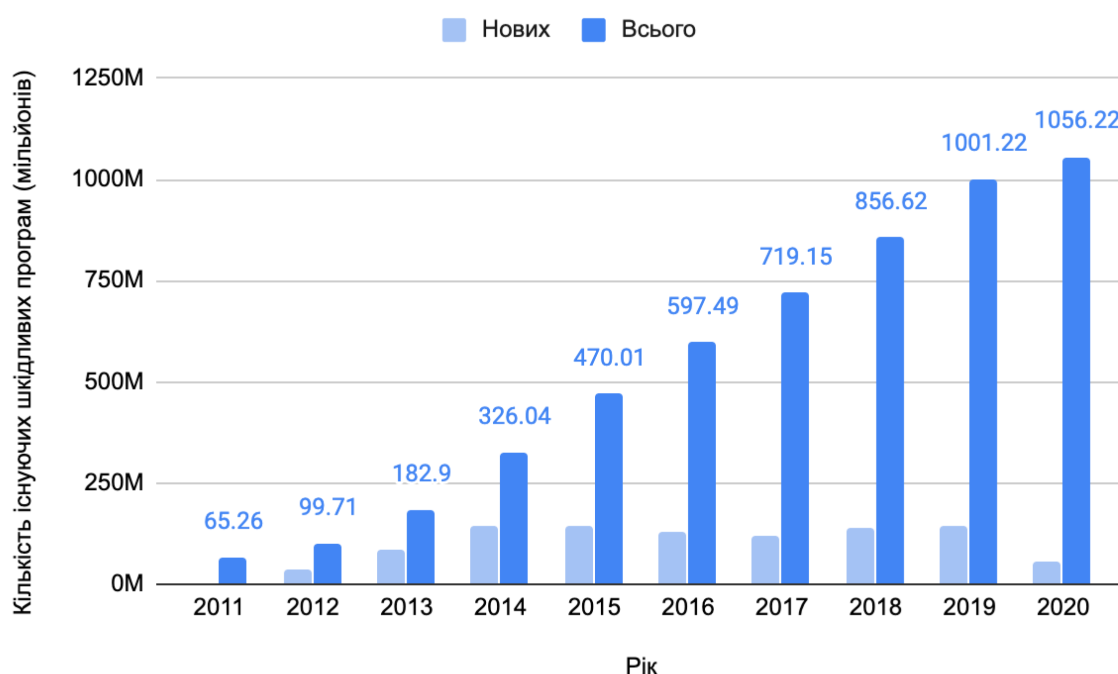

Рисунок 1.3 – Кількість нового та всього існуючого зловмисного програмного забезпечення за роками

Деякі дослідники висловлюють погляд, що автоматичне визначення того, чи є дана програма шкідливим програмним забезпеченням, є теоретично нерозв'язною проблемою [13]. Тому, автоматизовані методи виявлення та аналізу обмежені цим теоретичним результатом і можуть вийти з ладу в деяких випадках. Тож для таких випадків потрібне ручне втручання, щоб вивчати нові атаки та аналізувати методи обходу захисту. Нові методи, отримані в результаті такого ручного аналізу, потім включаються в автоматизоване програмне забезпечення, підвищуючи його ефективність. Незважаючи на це, розробка нових алгоритмів для автоматичного виявлення шкідливого програмного забезпечення є перспективною, бо вона може стати одним із додаткових шарів безпеки, збільшуючи імовірність виявлення та підсилюючи загальний рівень безпеки систем та мереж.

1.3. Огляд евристичних методів виявлення шкідливого коду

Методи виявлення шкідливого програмного забезпечення класифікуються за різними категоріями з різних точок зору. У цій роботі ми припускаємо три категорії методів виявлення шкідливого програмного забезпечення.

А. Методи підпису

На сьогодні узгодження шаблонів є найпоширенішим методом виявлення зловмисного програмного забезпечення, а виявлення на основі підписів є найпопулярнішим методом у цій галузі. Підпис - це унікальна функція для кожного файлу, щось на зразок відбитка пальця виконуваного файлу. Методи, що базуються на підписах, використовують зразки, витягнуті з різних шкідливих програм, для їх ідентифікації, вони ефективніші та швидші за будь-які інші методи. Ці підписи часто витягуються з особливою чутливістю, оскільки вони є унікальними, тому ті методи виявлення, які використовують цей підпис, мають невеликий рівень помилок. Де цей невеликий рівень помилок є основною причиною того, що більшість поширених комерційних антивірусів використовують цю техніку.

Ці методи не можуть виявити невідомі варіанти шкідливого програмного забезпечення, а також вимагають великої кількості робочої сили, часу та грошей для отримання унікальних підписів. Це основні недоліки цих методів. Крім того, ще одним недоліком є неможливість протистояти шкідливим програмам, які мутують свої коди в кожній інфекції, такій як поліморфна та метаморфічна. Для вирішення цих проблем дослідницькі товариства пропонують абсолютно нову родину виявлення шкідливих програм.

B. Методи, засновані на поведінці

Методи виявлення зловмисного програмного забезпечення на основі поведінки спостерігають за поведінкою програми, щоб зробити висновок, чи є вона шкідливою чи ні. Оскільки техніки, засновані на поведінці, спостерігають за тим, що робить виконуваний файл, вони не сприйнятливі до недоліків підписних. Простіше кажучи, детектор на основі поведінки робить висновок про те, чи є програма шкідливою, перевіряючи, що вона робить, а не те, що вона говорить. У цих методах збираються

програми з однаковою поведінкою. Таким чином, один підпис поведінки може ідентифікувати різні зразки шкідливих програм. Ці типи механізмів виявлення допомагають виявляти зловмисне програмне забезпечення, яке продовжує генерувати нові мутанти, оскільки вони завжди будуть використовувати системні ресурси та послуги подібним чином. Детектор на основі поведінки в основному складається з наступних компонентів:

• Колектор даних: Цей компонент збирає динамічну/статичну інформацію про виконуваний файл.

• Інтерпретатор: Цей компонент перетворює необроблену інформацію, зібрану модулем збору даних, у проміжні уявлення.

• Збіг: Використовується для порівняння цього подання з підписами поведінки.

Одним із прикладів підходу на основі поведінки є технологія виявлення шкідливого коду на основі гістограми, запатентована Symantec.

Головною перевагою методів виявлення шкідливого програмного забезпечення на основі поведінки є можливість виявити тип шкідливих програм, які базові методи підпису не в змозі виявити, такі як невідомі та поліморфні варіанти шкідливих програм. З іншого боку, відсутність перспективних коефіцієнтів помилково позитивного співвідношення (FPR), а також велика кількість часу сканування є основними недоліками цих методів виявлення шкідливих програм на основі поведінки.

Коли розробники шкідливих програм здогадуються, що їх шкідливе програмне забезпечення буде виявлено, вони намагаються уникнути стратегій боротьби з шкідливим програмним забезпеченням, застосовуючи різні методи приховування. Далі представлено деякі найбільш відомі стратегії приховування.

• Заплутування: у цій техніці дії розробника встановлюють дії, які дозволять запобігти методам виявлення підписів для виявлення їх шкідливого програмного забезпечення. Ці дії включають додавання команд сміття, непотрібні стрибки тощо.

• Шифрування коду: шкідливі програми використовують цей захисний механізм, шифруючи себе або свою зловмисну діяльність. Зашифроване шкідливе програмне забезпечення - це комплекс, що складається з алгоритму дешифрування,

алгоритму шифрування, ключів шифрування та зашифрованого шкідливого коду. Коли шкідливе програмне забезпечення запускається, ключ та алгоритм розшифровки вже використовувались для дешифрування його шкідливої частини. Шкідливе програмне забезпечення копіює себе та генерує новий ключ, використовуючи новий згенерований ключ та алгоритм шифрування. Створюється нова зашифрована версія. Ця версія містить алгоритм шифрування та новий ключ. Отже, навіть ключ шифрування та зашифрований код постійно змінюються, але їх можна виявити завдяки фіксованому алгоритму декодування.

• Олігоморфна стратегія: зловмисні програми використовують цю стратегію, використовуючи шифрування як захисний механізм для захисту себе і можуть змінювати алгоритм шифрування лише протягом обмеженого часу. Наприклад, вірус, який має невелику, кінцеву кількість різних дешифрувальних петель.

• Поліморфна стратегія: цей тип шкідливих програм зазвичай шифрується за допомогою алгоритму шифрування. Отже, при будь-якій інфекції буде використовуватися інший ключ дешифрування. Крім того, поліморфне шкідливе програмне забезпечення може використовувати необмежену кількість алгоритмів шифрування, щоб уникнути виявлення. У кожному виконанні частина коду розшифровки буде змінюватися. В залежності від типу шкідливого програмного забезпечення, шкідливі дії чи інші дії, що виконуються шкідливим програмним забезпеченням, можуть бути розміщені під операціями шифрування. Зазвичай у зашифроване зловмисне програмне забезпечення вбудований механізм трансформації, який при будь-яких змінах генерує алгоритм випадкового шифрування. Потім движок і шкідливе програмне забезпечення шифруються за допомогою створеного алгоритму, і до них підключається новий ключ розшифровки.

• Метаморфічна стратегія: метаморфічні шкідливі програми - найскладніший тип шкідливих програм. Це шкідливе програмне забезпечення змінюється самостійно, так що новий екземпляр не має схожості з початковим. Шкідливе програмне забезпечення не має жодного механізму кодування, і в кожній передачі відбуваються автоматичні зміни у вихідному коді шкідливого програмного забезпечення.

Як ми вже згадували, методи виявлення зловмисного програмного забезпечення на основі підписів та поведінки мають деякі недоліки. Отже, для подолання цих недоліків пропонуються евристичні методи виявлення шкідливого програмного забезпечення. Евристичні методи виявлення шкідливого програмного забезпечення використовують методи видобування даних та машинного навчання для вивчення поведінки виконуваного файлу. Наприклад, як перша спроба, Наїв Байєс та Мульти Наїв Байєс були використані Шульцем та ін. для класифікації шкідливих програм та доброякісних файлів.

Очевидно, що ці методи мають бути класифіковані:

A. API / системні виклики

Майже всі програми використовують виклики інтерфейсу прикладного програмування (API) для надсилання своїх запитів до Операційної системи, послідовності викликів API - це один із найпривабливіших способів, що відображає поведінку такого фрагмента коду, як шкідливе програмне забезпечення.

B. OpCode

OpCode (скорочення від Operational Code) - це підрозділ інструкції машинної мови, який ідентифікує виконувану операцію. Більш конкретно, програма визначається як серія впорядкованих інструкцій зі складання. Інструкція - це пара, що складається з операційного коду та операнда або списку операндів.

Найбільш значні дослідження OpCodes були проведені Біларом [20]. Він продемонстрував можливість використання окремих кодів OpCodes як функції при виявленні шкідливого програмного забезпечення. З цією метою він статистично проаналізував можливості окремих кодів OpCodes і продемонстрував їх високу надійність для визначення шкідливості виконуваного файлу та довів, що OpCodes можна використовувати як потужне представлення виконуваних файлів.

C. N-грами

N-грами - це всі підряди більшого рядка довжиною N. Наприклад, рядок “VIRUS” може бути сегментований на кілька 3-грамів: “VIR”, “IRU”, “RUS” тощо. За останнє десятиліття було проведено кілька досліджень щодо виявлення невідомих шкідливих програм на основі вмісту двійкового коду.

D. Контрольний графік потоку

Графік управління потоком (CFG) - це графік, який представляє управління потоком програм і широко використовується при аналізі програмного забезпечення і вивчався протягом багатьох років. CFG - це спрямований графік, де кожен вузол являє собою оператор програми, а кожен фронт представляє керуючий потік між операторами (тобто що після чого відбувається). Виписки можуть бути дорученнями, копіюваннями заяв, гілками тощо.

E. Гібридні особливості

На ефективність класифікаторів машинного навчання впливають два основні фактори: особливості та алгоритми. Деякі дослідники прагнуть покращити точність машинного навчання завдяки особливостям. Вони поєднують функції, щоб отримати кращу точність.

1.4. Аналіз шкідливих програм методом візуалізації двійкових файлів

Візуалізований метод аналізу шкідливого програмного забезпечення, який розглядається у даному розділі, складається з трьох етапів, як показано на рисунку 1.4. На кроці 1 двійкова інформація витягується з двійкових файлів зразків, а матриці зображень, в яких двійкова інформація записується як кольорові пікселі RGB, генеруються на кроці 2. На кроці 3 розраховується подібність між матрицями зображень.

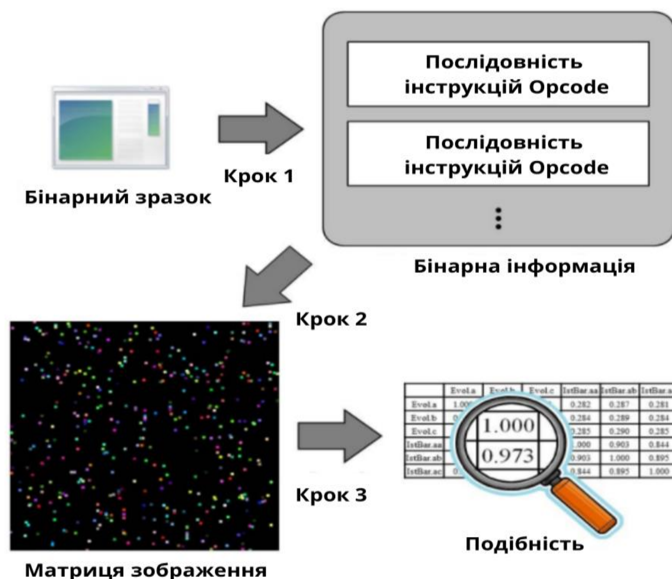

Рисунок 1.4 – Огляд запропонованого методу

На рисунку 1.5 показано процес вилучення двійкової інформації з двійкових файлів-зразків на кроці 1.

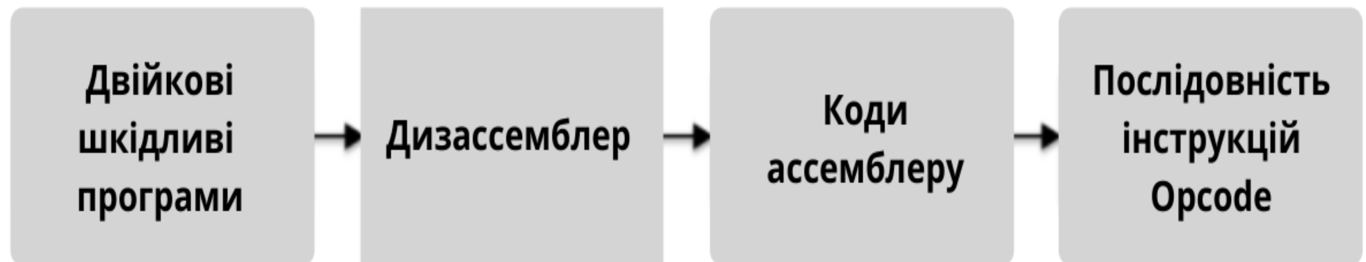

Рисунок 1.5 - Процедура вилучення двійкової інформації

Для вилучення двійкової інформації файли двійкових зразків спочатку розбираються за допомогою інструментів розбирання, таких як IDA Pro або OllyDbg. Після вилучення збірних кодів за допомогою інструменту послідовність збірних кодів поділяється на блоки відповідно до деяких інструкцій, які використовуються як роздільники, як показано на рисунку 1.6.

Послідовність кодів операцій, що входять до окремих блоків, використовується як двійкова інформація. З кожного операційного коду для створення інформації для блоку використовуються лише перші три символи. Наприклад, чотири символьні інструкції коду роботи, такі як push, зводяться до трьохсимвольних інструкцій. Потім ці трисимвольні інструкції об’єднуються разом, і рядок символів використовується для представлення блоку коду операцій на наступному кроці для створення матриці зображення.

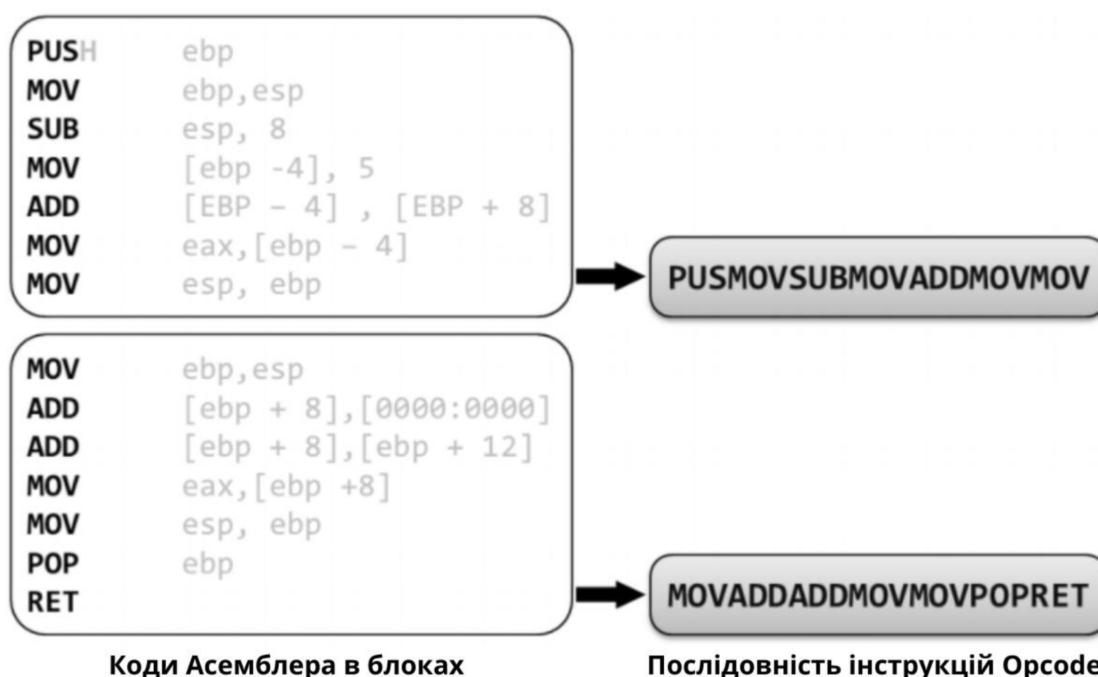

Рисунок 1.6 - Opcode інструкції, що використовуються як бінарна інформація

На рисунку 1.7 показана процедура кроку 2, яка перетворює послідовності команд операційного коду в матрицю зображення. Дві хеш-функції використовуються для визначення координатної інформації та інформації про колір RGB, як показано на рисунку 1.7.

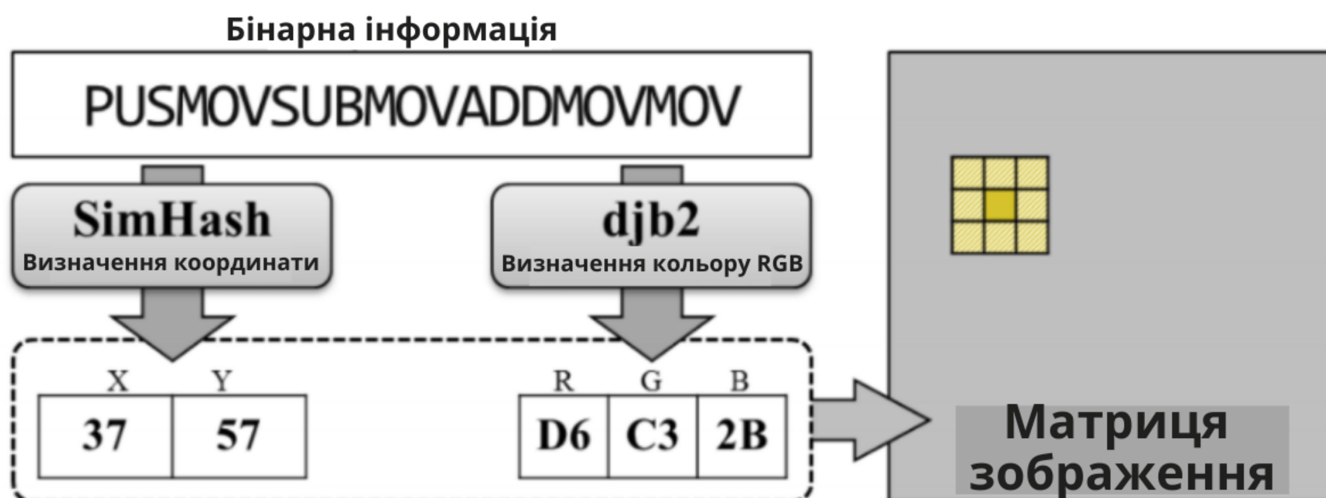

Рисунок 1.7 - Генерація зображень за допомогою двійкової інформації

Для того, щоб візуалізувати двійковий файл як матрицю зображення, як довжина, так і ширина матриці зображення ініціалізуються до 2n, де n вибирається

користувачами. Щоб зменшити ймовірність зіткнення хеш-функцій, n має бути досить великим. У наших експериментах ми обрали n як 8, щоб уникнути зіткнень.

Модуль, що визначає координати, і модуль, що визначає колір RGB, використовуються для створення матриць зображень. Спочатку модуль, що визначає координати, визначає координати (x, y) на матрицях зображень для двійкової інформації кожного блоку коду. SimHash застосовується до двійкової інформації, вилученої на кроці 1. SimHash є локально-чутлива хеш-функція, яка передбачає, що якщо вхідні значення подібні, вихідні значення також будуть подібними.

Отже, якщо рядки символів двійкової інформації схожі, виходи будуть схожими, і це відобразиться у подібні координати в матриці зображення.

По-друге, модуль визначення кольору RGB визначає значення кольорів зображень на матриці зображення застосовується djb2 до двійкової інформації для визначення кольорів зображень для двійкової інформації. Кольори RGB визначаються шляхом обчислення значень по 8 біт для червоного, зеленого та синього кольорів.

Після визначення координат та кольорів RGB окремих зображень кольорові зображення RGB записуються на окремі координати матриць зображень. Щоб забезпечити людських аналітиків більш зручним візуальним аналізом, пікселі навколо визначених координат реєструються одночасно. Як показано на рисунку 1.8,

записуються дев'ять пікселів від (x – 1, y – 1) до (x + 1, y + 1) навколо координати (x, y), визначеної в послідовності команд операційного коду для блоку.

x − 1, y − 1	x, y − 1	x + 1, y − 1
x − 1, y	**x,y**	x + 1, y
x − 1, y + 1	x, y + 1	x + 1, y + 1

Рисунок 1.8 - Запис дев’яти пікселів для послідовності однієї Opcode інструкції

Якщо зображення перекриваються, оскільки координати, визначені для декількох послідовностей команд операцій, сусідні, як показано на рисунку 1.9, суми кольорів RGB стають новими піксельними кольорами. Якщо результат підсумовування кольорів перевищує 255 (0xFF), результат буде встановлений на 255. Наприклад, якщо RGB1 дорівнює (255,0,0) і RGB2 дорівнює (0,176,50), новий колір стане (255,176,50).

$$R_3 = \min(R_1+R_2,\ FF) = \min(FF+00,\ FF) = FF$$
$$G_3 = \min(G_1+G_2,\ FF) = \min(00+B0,\ FF) = B0$$
$$B_3 = \min(B_1+B_2,\ FF) = \min(00+50,\ FF) = 50$$
$$\therefore \mathbf{RGB_3 = (FF,B0,50)}$$

Рисунок 1.9 - Метод запису накладених пікселів

Оскільки кількість пікселів, записаних на матриці зображення, різниться відповідно до розміру файлу та номера інструкції opcode послідовності, а кількість пікселів, що перекриваються, зростатиме як кількість зображень збільшується. Якщо занадто багато перекриваються зображення, розмір матриці зображення слід збільшити.

Для обчислення подібності ми використовували “вибіркове узгодження площ” між матрицями зображень. Для вибіркового узгодження областей зображення матрицю слід розділити на N частин, де N можна встановити в 4 рази (x = 1,2,3, ...), наприклад 4, 16 та 64. На рисунку 1.10 показано зображення матриці, в яких ділянки

були розділені відповідно до різних N значення. Потім із зображення випадковим чином вибирається n штук матриця для порівняння. Наприклад, як показано на рисунку 1.10, матрицю зображення можна розділити на 16 (N = 16) областей і чотири (n = 4) області можна вибрати випадковим чином.

Тепер відповідні пікселі ідентифікуються у кожній вибраній області та використовуються для розрахунків подібності. В цьому розділі для обчислення подібності між матрицями зображень використовується векторний кутовий алгоритм вимірювання відстані, який визначає подібність за допомогою векторного значення для кожного пікселя. Розраховується схожість між n частинами областей, а загальна подібність обчислюється як середнє значення подібності для відповідних пікселів на кожній області.

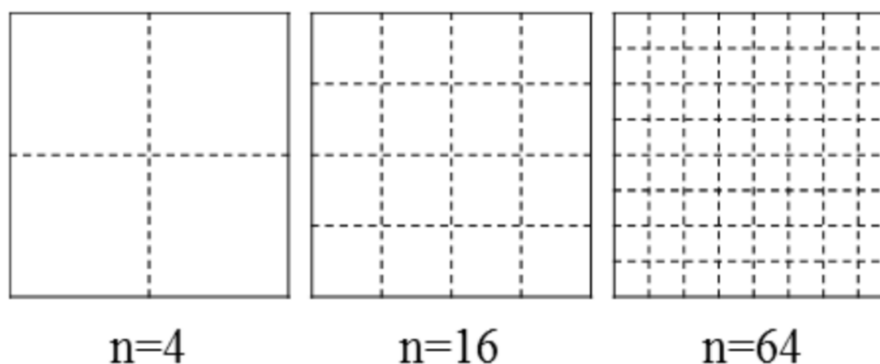

Рисунок 1.10 - Площі матриць зображення розділені відповідно до значень N

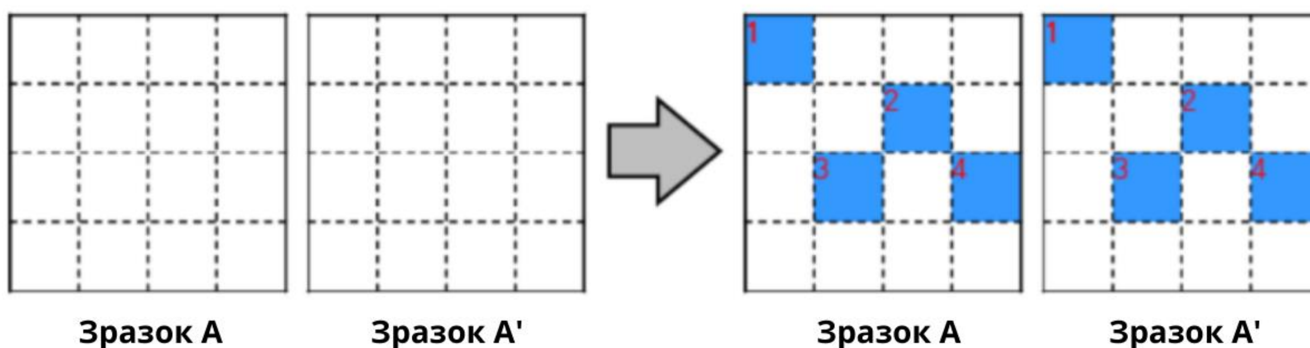

Рисунок 1.11 - Приклади випадково вибраних областей (N = 16, n = 4) 4

Висновки за розділом 1

Шкідливе програмне забезпечення стало основною проблемою кіберпростору в час повсюдного доступу до Інтернету. Для боротьби зі зловмисними програмами, дослідники, аналітики та компанії, що створюють антивіруси, застосовують два основних методів аналізу шкідливого програмного забезпечення: статичний, коли досліджувані програми ніколи не запускаються, та динамічні – коли програми, що досліджуються, запускаються та тестуються в віртуальному середовищі на серверах дослідників або в реальному часі на користувацькому пристрої.

Зростаюча складність та кількість шкідливого програмного забезпечення робить неможливим масовий ручний аналіз зразків і вимагає створення високоточних автоматизованих процесів перевірки виконуваних файлів. Сучасні підходи до автоматизованої перевірки досі здебільшого покладаються на перевірку хеш-підписів, що не є ефективною проти вразливостей нульового дня та мінімально змінених вже відомих шкідливих зразків.

Водночас, більш розповсюдженими стають динамічні підходи, що перевіряють поведінку зразка в віртуальному середовищу, та «розумні» статичні підходи, що оцінюють структуру та міру схожості програм з іншими відомими. Обидва підходи керовані даними, тобто можуть навчатися та підвищувати свою ефективність лише завдяки збільшенню кількості відомих зразків. Це стало можливим завдяки швидкому розвитку сфери машинного навчання, глибокого навчання, що застосовує для класифікації глибокі штучні нейронні мережі. В наступному розділі будуть розглянуті підходи машинного навчання, а також проаналізована література з застосування машинного навчання до проблеми аналізу шкідливого програмного забезпечення.

РОЗДІЛ 2
МАШИННЕ НАВЧАННЯ ТА НЕЙРОННІ МЕРЕЖІ

2.1 Автоматична класифікація та виявлення шкідливих програм за допомогою машинного навчання та нейронних мереж

Зловмисне програмне забезпечення означає будь-яке програмне забезпечення, яке певним чином завдає шкоди користувачам, комп'ютерам або мережам. Шкідливе програмне забезпечення містить віруси, хробаків, бекдори, троянських коней або інші шкідливі програми.

В даний час шкідливе програмне забезпечення є важливою проблемою в галузі інформаційної безпеки . Згідно з доповіддю "Лабораторії Касперського", за минулий рік було атаковано 58% корпоративних комп'ютерів, а 29% компаній постраждали від мережевих атак. Хоча щодня виявляються сотні тисяч нових шкідливих програм, більшість із них походять із відомих сімей шкідливих програм.

Методи замаскування шкідливого програмного забезпечення включають переважно пакування, метаморфозу та віртуальні технології. Ці технології широко використовуються для уникнення виявлення антивірусного програмного забезпечення . Більшість систем виявлення шкідливих програм, прийнятих виробниками антивірусів, базуються на виявленні підписів та аномалій. Хоча техніка підписів має високу точність, вона не може виявляти нові шкідливі програми і повинна оновлювати свою бібліотеку функцій у режимі реального часу вручну. Нове шкідливе програмне забезпечення можна знайти шляхом виявлення аномалій; проте частота помилкових тривог висока.

Залежно від стану аналізованого шкідливого програмного забезпечення, аналіз шкідливого програмного забезпечення можна розділити на статичний та динамічний аналіз. Статичний аналіз відноситься до аналізу виконуваних файлів без запуску програми. Перевагами статичного аналізу є його здатність знаходити авторський стиль та профілювати потік коду.

Недоліком є те, що його легко зірвати методами затуманення. З іншого боку, динамічний аналіз дозволяє спостерігати за робочим станом програми в безпечному та контрольованому середовищі. Цей підхід здатний точно відображати поведінкові характеристики програми. На це не впливає шифрування, стиснення, метаморфоза тощо. Однак цей метод витрачає час не лише на налагодження програми, а й на відстеження та перезапис запущеного процесу програми. Тому динамічний аналіз, як правило, більш неефективний, ніж статичний. Крім того, на нього поширюються деякі обмеження в робочому середовищі. Шкідливе програмне забезпечення може містити умови активації; таким чином, деякі зловмисні дії можуть спостерігатися неправильно.

Завдяки успішному застосуванню машинного навчання в галузі обробки зображень, розпізнавання мови та програмної інженерії, машинне навчання стало важливим методом аналізу шкідливих програм за останні 10 років. Як правило, за допомогою машинного навчання процес виявлення шкідливих програм можна розділити на три етапи.

По-перше, функції виконавчого файлу, які витягуються за допомогою статичного та динамічного аналізу, відображаються у вхідних даних машинного навчання.

По-друге, модель прогнозування тренується з використанням цих особливостей, що означає отримання типу підпису вищого рівня. Нарешті, передбачається невідоме програмне забезпечення. Машинне навчання здатне автоматично перевіряти та виявляти властивості програмного забезпечення; таким чином, він може розрізняти доброякісне програмне забезпечення та шкідливе програмне забезпечення. Машинне навчання також може бути використано для присвоєння невідомої шкідливої програми відомому сімейству. Однак автори шкідливих програм можуть легко створити велику кількість та різноманітність варіантів шкідливих програм за допомогою автоматичних інструментів. Тому нам рекомендується використовувати машинне навчання для вирішення таких проблем:

1. Як швидко та ефективно призначати варіанти відповідним сім’ям?
2. Як виявити нове шкідливе програмне забезпечення?

2.2 Основи машинного навчання та нейронних мереж

Цей розділ пояснить основні поняття, що стоять за методами, які використовуються в цій дипломній роботі. Ця теорія допоможе в розумінні понять, які використовуються в наступному розділі, де ця теорія використовується на практиці.

Основним завданням цієї роботи є класифікація шкідливих програм на шкідливі та нешкідливі. Це завдання може виконувати людина, яка має глибоке розуміння шкідливих програм. Оскільки кількість шкідливих програм може бути теоретично нескінченною, цю задачу необхідно автоматизувати. Для цього нам потрібен класифікатор, що зможе виконувати те ж завдання. Класифікатори можуть бути багатьох типів. Класифікатор можна зробити за допомогою евристики (набору правил), що створюється людиною при вивченні різних типів шкідливих програм. Наприклад, якщо у файлі знайдені певні типи заголовків, ми можемо з високою ймовірністю знати, що це шкідливе програмне забезпечення. Але знову ж таки може бути багато різних способів створення зловмисного програмного забезпечення та складне завдання — створити евристику для всіх типів. Крім того, нам може знадобитися спочатку вивчити шкідливе програмне забезпечення, щоб оновити нашу евристику, і до цього часу багато машин вже могли бути заражені. Тому є необхідність у класифікаторах, які автоматично вивчають структуру файлів та основні функції шкідливого програмного забезпечення та добре узагальнюють вивчене. Для вирішення цього питання в цій роботі використовуються штучні нейронні мережі.

Штучні нейронні мережі дещо мотивовані тим, як працює власне біологічний мозок, моделюючи його за допомогою статистики та прикладної математики. Мозок людини містить велику кількість взаємопов’язаних клітин, які називаються нейронами. Інформація передається від одного нейрона до іншого у вигляді електричних сигналів. Мозок отримує багато сенсорних входів, і нейронна павутина маніпулює ним у свідомій частині мозку. Кожен нейрон може бути змодельований як

нелінійна функція $f(x)$, яка приймає вхідні сигнали з усіх нейронів, до яких він підключений, у вигляді електричних сигналів. Сила цих сигналів змінюється під час подорожі від одного нейрона до іншого, залежно від хімічного складу з'єднання. Потім, якщо сумарний вхідний сигнал перевищує певний поріг, нейрон видає сигнал y, який потім приймається на вхід іншими нейронами, як показано на рис. 2.1. Це продовжується у дуже складною зв'язаній структурі нейронів, що призводить до дуже складної функції. Вхід нейрона X можна записати як $w_1x_2 + w_2x_2 + \cdots + w_nx_n$, де w_i є вагами, що змінюють силу сигналу. Тоді нейрон має безперервну функцію активації, $\varphi(X)$. Коли ми переживаємо нові речі та вчимося, старі зв'язки змінюються, та формуються нові зв'язки. Саме так, вважають, відбувається навчання.

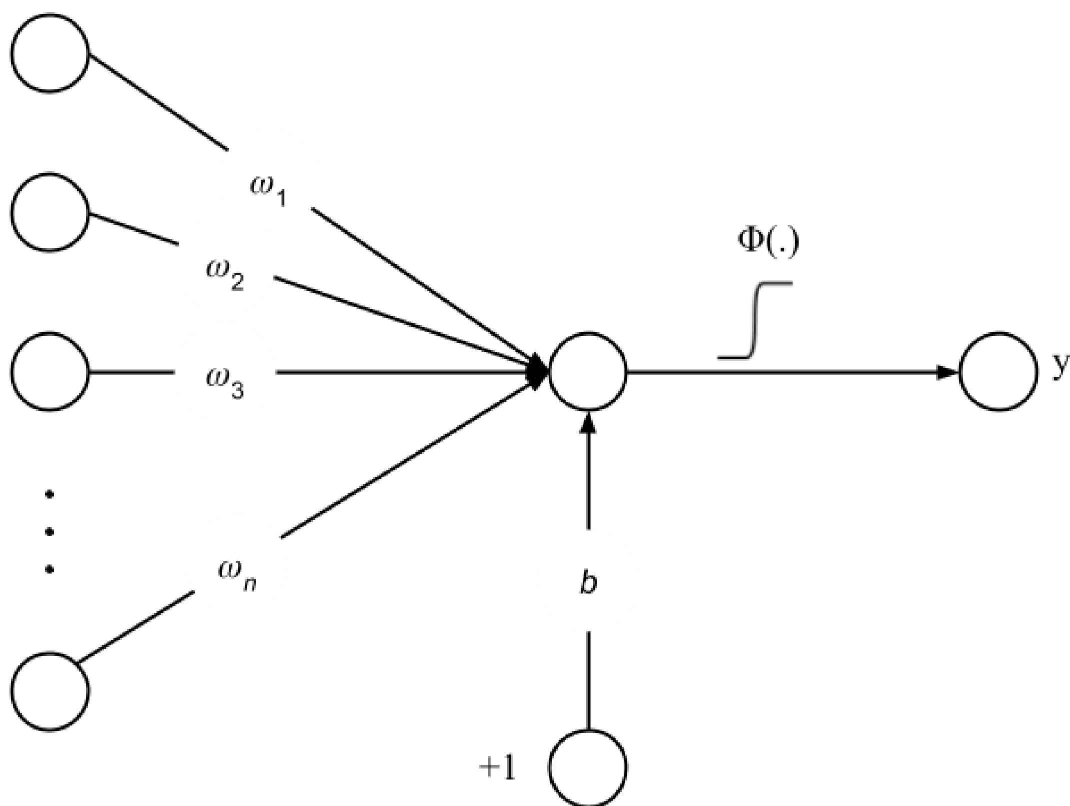

Рисунок 2.1 – Графічне зображення того, як працює нейрон. На зображенні: y - вихідний сигнал, φ - функція активації, x_i - входи інших сполучених нейронів, w_i - відповідні ваги. b додає поріг активації. [14]

Структура штучних нейронних мереж (ШНМ) є аналогічною до описаної вище моделі. ШНМ складаються зі з'єднаних одиниць, званих штучними нейронами. Вони організовані пошарово. Дані подорожують від вхідного до вихідного шару, проходячи через багато шарів. Кожне з'єднання між двома нейронами має вагу w_i, як правило, реальне число між 0 і 1, що регулює, як змінюється вхідний сигнал перш ніж передаватися наступному нейрону. На додаток до цього, на кожному вузлі, як правило, є порогова або обмежувальна (активаційна) функція. Ці шари можуть бути повністю з’єднані (кожен нейрон з'єднаний з усіма нейронами в наступному шарі) або частково з'єднані (вибрані випадковим чином). Отже кожен нейрон може бути представлений функцією:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

де y – вихід;

φ – функція активації;

n – кількість вхідних нейронів, що надходять у нейрон;

w_i – величина ваги і-го з'єднання;

x_i – вихід нейронів, що подаються в нейрон;

b – поріг.

Можна вважати b нейроном з постійним виходом значенням -1. Насправді, значення b варіюється від одного нейрона до іншого, що дозволяє мережі обрати різні пороги активації для різних нейронів. На рис. 2.2 показана повна нейронна мережа. Існує загалом три типи шарів, вхідний шар, прихований шар і вихідний шар. Вхідний шар розглядає необроблену інформацію, подану як вхід до мережі. Приховані шари отримують деяке інше зображення шару, визначене вагами та функціями активації попередніх шарів. Це важливо, тому що приховані шари можуть вільно вивчити власне представлення вхідних даних і таким чином спростити проблему класифікації для наступних шарів. Всю мережу можна розглядати як набір шарів, де кожен шар засвоює різне представлення вхідних даних (видаляючи непотрібну інформацію та

зберігаючи лише корисну інформацію для виконання завдання). Ці штучні нейрони також називаються перцептроном, і коли є кілька шарів нейронів, вся нейронна мережа називається багатошаровим перцептроном. Нейронні мережі, в яких виходи з кожного шару подаються як входи до наступного шару, також називаються Feed-Forward Networks (нейронна мережа прямого поширення).

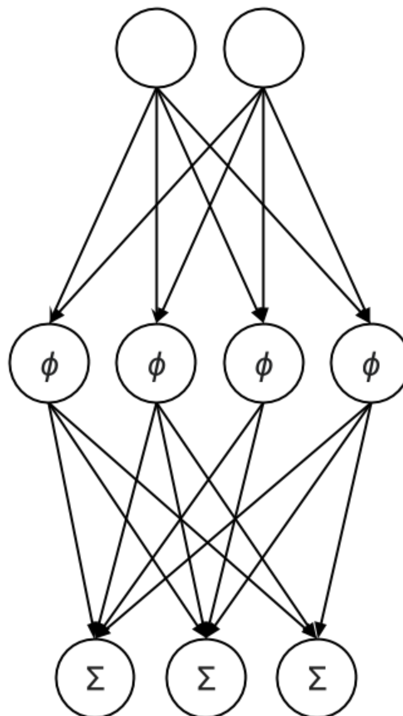

Рисунок 2.2 – Графічне зображення штучної нейронної мережі із одним прихованим шаром [14]

Як згадувалося раніше, класифікатор має *вивчати функції автоматично*. Штучні нейронні мережі роблять це за допомогою коригування їх ваг таким чином, щоб їх вихід наближався до потрібного результату для кожного запропонованого нами прикладу. Існує два основні типи навчальних процесів: один – контрольований, або «під наглядом» (supervised), інший – неконтрольований, або «без нагляду» (unsupervised). Під час контрольованого навчання ми знаємо, який результат повинна видати мережа для кожного навчального прикладу, і ми використовуємо цю інформацію під час навчання. При неконтрольованому навчанні, навпаки, ми не знаємо правильних результатів, і ми просто хочемо об'єднати схожі дані/розділити

різні дані або вивчити деякий розподіл, з якого вибираються вхідні дані. Контрольоване навчання можна використовувати, коли існують розмічені дані, як це було в цій дипломній роботі. На етапі навчання по черзі подаються приклади з набору даних та щоразу змінюються ваги на основі помилки, зробленої моделлю. Це можна розглядати як метод навчання через помилки. Цей процес повторюється до тих пір, поки різниця між виходами від мережі та бажаними виходами не стане дуже маленькою, якщо це можливо. Існує багато способів досягти цього. Один із найпопулярніших способів розпочати цей процес — це випадково ініціалізувати ваги нейромережі, використовуючи деякий розподіл. Вхідні дані, X, подаються в мережу, а вихідні, $\hat{y}$, отримуються. Далі, обчислюється помилка, описуючи, наскільки далеко знаходиться $\hat{y}$ від фактично бажаного виводу y. Наша мета – мінімізувати цю помилку наскільки це можливо. Як саме обчислюється ця помилка, дуже важливо, оскільки вона визначає проблему оптимізації. Один із застосовуваних прийомів – це обчислення середньої квадратичної помилки за допомогою методу найменших квадратів. На основі цієї помилки ваги потім оновлюються за допомогою методу, званого зворотним поширення помилки (backpropagation). Зворотне поширення ґрунтується на тому, що оптимальною зміною ваги (для максимального зменшення помилки) була б така, коли помилка зменшується на невеликий крок дельта. З лінійної алгебри ми знаємо, що помилка буде зменшуватися найшвидше в напрямку, де негативний нахил функції помилки є максимальним. Також, щоб не пропустити мінімум, доводиться робити багато дуже маленьких кроків в цьому напрямку, поки значення помилки не досягне мінімального. Тренування відбувається пакетами (batches). Один пакет включає всі приклади в наборі даних або деяку фіксовану кількість прикладів, випадковим чином обраних із набору даних. Пакети продовжують подаватися до тих пір, поки не буде досягнено достатньо низької помилки або помилка перестане змінюватися.

Розпізнавання зображень – одне з найважливіших застосувань, де нейронні мережі успішно впроваджуються та показують справді чудові результати. Штучні нейронні мережі вже використовуються як популярна практика в галузях промисловості, охорони здоров'я, управління ризиками тощо. Як уже говорилося

раніше, структура штучних нейронних мереж включає вхідний шар, один або більше прихованих шарів та вихідний шар. Багато завдань з розпізнавання зображень можуть стати дуже складними, і тому їх неможливо засвоїти за допомогою одного прихованого шару. Якщо розглянути кожен шар як функцію f_i на вході, можна вивчати складніші і складніші функції, використовуючи багато шарів ... $f_4(f_3(f_2(f_1)))$ На рис. 2.3 показаний приклад того, як різні шари можуть сприяти вирішенню загальної задачі у випадку розпізнавання зображень. Таким чином, збільшення кількості шарів дозволяє нам вивчити складніші функції, або іншими словами навчитися виконувати більш складні завдання. Кількість шарів в нейронній мережі представляє її глибину. Таким чином, моделі, що використовують багато прихованих шарів, називають глибокими моделями, і таке навчання називають глибоким навчанням. Використовуючи глибоке навчання в розпізнаванні зображень, можна отримати набагато більшу точність, ніж застосовуючи будь-які попередні методи. Далі будуть детально розглянуті глибокі мережі.

Івахненко та Лапа в 1965 р. Першими запропонували працюючий алгоритм навчання глибоких, багатошарових перцептронів. З тих пір протягом багатьох років було проведено багато досліджень на різних моделях, але перші серйозні зміни відбулися в 2006 році, оскільки комп'ютери тепер могли виконувати величезну кількість обчислень. Вперше було показано, як багатошарова нейромережа може бути попередньо натренована та використана для виконання різних завдань після фінального навчання. З тих пір нейромережі стали найпопулярнішою технікою у багатьох практичних сферах. Щоб зрозуміти можливості нейронних мереж, спершу трохи повернемося назад і подивимось, як глибоке навчання відрізняється від інших методик класифікації та як воно долає їх обмеження.

Найпростішими класифікаторами є лінійні моделі. Лінійні моделі можуть бути застосовані до задач, де завдання класифікації можна вирішити, застосувавши до вхідних даних деяке лінійне перетворення. Один із популярних прикладів — лінійна регресія. У лінійній регресії маємо скалярну залежну змінну y (вихід) та одну чи більше описових змінних, позначених X (вхід). Використовуючи багато точок даних, ми намагаємось вивчити модель β так, що:

$$y_i = X_i^T \beta + \varepsilon_i \#(2.2)$$

де X_i^T – і-тий елемент транспонованого вектора описових змінних;

β – лінійна модель;

ε_i – випадкова змінна, що моделює шум під час збору даних;

n – кількість описових змінних (розмір вектора описових змінних);

i – лічильник від 1 до n.

Лінійна регресія використовується у багатьох статистичних моделях. Наприклад, моделювання констант у деяких поліноміальних відносинах у фізиці з використанням точок даних, сформованих в результаті експериментів. Як випливає з назви, проблема з лінійною регресією полягає в тому, що вона може вирішувати лише лінійні залежності. А що робити, якщо взаємозв'язок має форму сигмоїдної функції, якогось многочлену високого ступеня, або комбінації цих двух форм? Для моделювання таких відносин в модель потрібно включити нелінійну функцію. Це розуміння призвело до розробки багатьох методик класифікації нелінійного машинного навчання. Найпоширенішою технікою розпізнавання та класифікації зображень були SVM (Support Vector Machines, укр. Опорно-Векторні Машини). SVM спочатку перетворює вхідні дані, використовуючи деяку нелінійну функцію $\varphi(X)$, відому як ядро, а потім застосовує лінійну модель до перетворених вхідних даних. Найкращим вибором функції φ може бути будь-що і повністю залежить від набору даних. Якщо ми підберемо дуже загальну φ (наприклад, нескінченномірну φ), ми завжди можемо вивчити будь-який навчальний набір даних, але це призводить до поганого узагальнення. Таким чином, інженери повинні вручну конструювати φ, і саме цим користувалися люди до появи глибокого навчання.

У глибокому навчанні функція φ *вивчається автоматично*. Це чудово з двох причин. По-перше, модель може вивчити будь-яку нелінійну функцію. По-друге, не потрібно турбуватися про ручне конструювання складної φ. Далі розглянемо типову модель глибокого навчання.

У процесі глибокого навчання результат $\hat{y}$ може моделюватися як $f(x;\ \theta, w)$, де θ – параметр для вивчення φ, а w – ваги, які перетворюють значення φ до y. Таким чином, $y = \varphi(x;\ \theta)w$.

Щоб знайти гарне представлення $\hat{y}$, використовуються алгоритми оптимізації. Зворотне поширення помилки – один з таких алгоритмів, який використовується для вирішення проблеми оптимізації. Як і в багатьох інших моделях, існує багато параметрів, які потрібно визначити перед тренуванням нейронної мережі. Вони обираються експериментально в процесі навчання, залежать від сфери застосування та розподілу даних, і називаються гіперпараметрами. Серед таких гіперпараметрів можна виділити необхідність вибору:

- функції втрат (помилки);
- форми вихідних нейронів;
- функції активації;
- глибини мережі;
- ширини кожного прихованого шару;
- сполучуваності шарів тощо.

На даний момент, найпопулярнішим вибором для функції активації є функція ReLU (Rectified Linear Unit, випрямляч):

$$f(x) = x^{+} \ = (0, x)\ \#(2.3)$$

Ця функція застосовується в більшості архітектур нейронних мереж для всіх шарів окрім останнього. ReLU замінив попередньо-вживані фукнції $tanh$ та $sigmoid$, бо він допомогає краще вирішувати проблему затухання градієнтів – для більшості можливих значень вхідних нейронів зазначені функції видають значення близьке до нуля, що робить процес навчання дуже повільним. В ReLU всі значення нейрона більше нуля передаються далі, і це дозволяє прискорити навчання мережі.

Для останнього шару, що видає результат обчислень мережі, як правило, застосовуються дві основних функції активації, вибір між якими залежить від кількості класів або вихідних одиниць: Sigmoid та Softmax.

Функція Sigmoid (сигмоїда) – це неперервно диференційована монотонна нелінійна S-подібна функція, що перетворює значення вхідної величини у значення в діапазоні [0, 1]. Коли вона використовується як функція активації, вона дозволяє отримати значення p ймовірності віднесення прикладу до одного з двох класів, або ймовірність віднесення до протилежного класу $1 - p$. Формула для розрахунку цієї функції виглядає наступним чином:

$$sigmoid(x) = \frac{1}{(1 + e^{-x})} \#(2.4)$$

де x – вхідне значення;

e – число Ейлера.

Для вирішення задач багатокласової класифікації або декількох вихідних одиниць використовується більш складна функція Softmax, яка є узагальненням логістичної функції**.**

Функція Softmax (нормована експоненційна функція) – це узагальнення логістичної функції, що "стискує" K-вимірний вектор z із довільними значеннями компонент до K-вимірного вектора $\sigma(z)$ з дійсними значеннями компонент в області [0, 1] що в сумі дають одиницю — $\sigma : R^K \longrightarrow [0,1]^K$. Це дозволяє отримати значення ймовірностей віднесення прикладу до кожного із класів. Функція задається наступним чином:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} \quad e^{z_k}} \#(2.5)$$

де z – вхідний K-вимірний вектор;

K – розмірність вхідного вектору;

e – число Ейлера;

j – лічильник компонент вхідного вектора, $j = 1, \dots, K$;

k – другий лічильник компонент вхідного вектора.

В якості функції втрат (або функції помилки) популярним вибором слугує крос-ентропія, що для двох класів визначена як:

$$J = -\sum \quad y_i log\, \hat{y}_i = -y\, log\, log\, \hat{y} - (1 - y)\, log\, log\, (1 - \hat{y})\ \#2.6$$

де J – це функція втрат (кросс-ентропія);

y – реальне значення класу прикладу;

$\hat{y}$ – передбачене значення ймовірності класу (результат обчислення моделі).

Усі ці параметри залежать від проблеми, набору даних, обчислювальної спроможності тощо. Через велику кількість прихованих шарів методи глибокого навчання страждають від перенавчання (overfitting). Щоб уникнути перенавчання, під час навчання можна застосовуватися різні методики, такі як зменшення ваг або виключення (dropout) [15].

Звичайні штучні нейронні мережі добре працюють з розпізнаванням та класифікацією зображень, але страждають від прокляття багаторозмірності і тому можуть використовуватися лише для зображень низького розміру. Наприклад, зображення розміром 32x32 пікселів вимагає 3072 ваг на вході. Збільшення розміру призводить до величезної кількості ваг і, таким чином, робить навчання обчислювально важким. Крім того, через їх плоску структуру звичайні нейронні мережі не в змозі зрозуміти зв'язок між сусідніми пікселями і, таким чином, не зможуть повністю скористатися 2D структурою зображень. Ці обмеження були подолані згортковими нейронними мережами (ЗНМ, англ. – Convolutional Neural Network (CNN)).

2.3 Згорткові нейронні мережі (CNNs)

Згорткові нейронні мережі були натхнені біологічними структурами з зорової кори котів. Вона складається із складного розташування клітин, які вивчали Губель та Вайзель у своїй роботі [16], що називається рецептивним полем. Ці клітини поводяться як фільтри та можуть використовувати локальну просторову кореляцію у зображеннях. В процесі роботи Губеля та Вайзеля було виявлено два типи клітин. Один тип в основному розпізнає контури, а інший тип – це клітини зі складними фільтрами. Згорткові нейронні мережі розроблені так, щоб їх шари виконували аналогічні функції.

Структура згорткових нейронних мереж містить кілька наборів шарів згортки з подальшим об'єднанням і в кінці повністю пов'язаних шарів для класифікації. На рис. 2.3 показаний типова архітектура LeNet для класифікації зображень. Перший крок роботи мережи – операція згортання (convolution). Ряд фільтрів, спочатку рандомізованих і пізніше виведених мережею під час навчання, застосовуються до зображення за допомогою операції згортки. Цей крок розпізнає важливі елементи структури у зображенні. Наприклад, один фільтр може розпізнавати всі контури, а інші можуть розпізнавати контраст. Виходи з кожного фільтра укладаються один за одним для формування 3D-мережі. Для введення нелінійності та обмеження вихідних значень застосовується нелінійна функція активації (як правило, ReLU). Після цього застосовується просторове об'єднання для зменшення розмірності та збереження лише важливих особливостей зображення. Об'єднання також робить поведінку моделі більш інваріантною щодо невеликих перетворень, зміщень та шумів. І врешті-решт, мережа отримує розміро-інваріантне представлення зображення, що дуже важливо для гарної роботи класифікатора. Існують різні типи об'єднань, наприклад, сумарне, середнє та максимальне. Цей процес можна повторювати декілька разів, як показано на рисунку 2.3. Останнім шаром є повнозв'язаний шар нейронів із функцією активації Softmax для багатокласової класифікації. Усі фільтри та ваги моделі вивчаються лише за допомогою методу зворотнього поширення помилки. Оскільки

той же фільтр застосовується до всього зображення, класифікатор стає інваріантним до положення об'єкта.

Згорткові нейронні мережі найкраще підходять для класифікації в сфері розпізнавання зображень. Але в міру збільшення кількості класів завдання стає складніше. Розмір мережі збільшується, що призводить до різних проблем, таких як проблема затухання градієнтів, коли переднім шарам стає складніше тренуватися. Щоб вирішити цю проблему, у 2015 році Microsoft представила ResNet (Redsidual Network). Мережа, що використовувалась Microsoft, мала 152 шари та найкраще виступила у змаганнях з класифікації зображень ILSVRC 2015 [17].

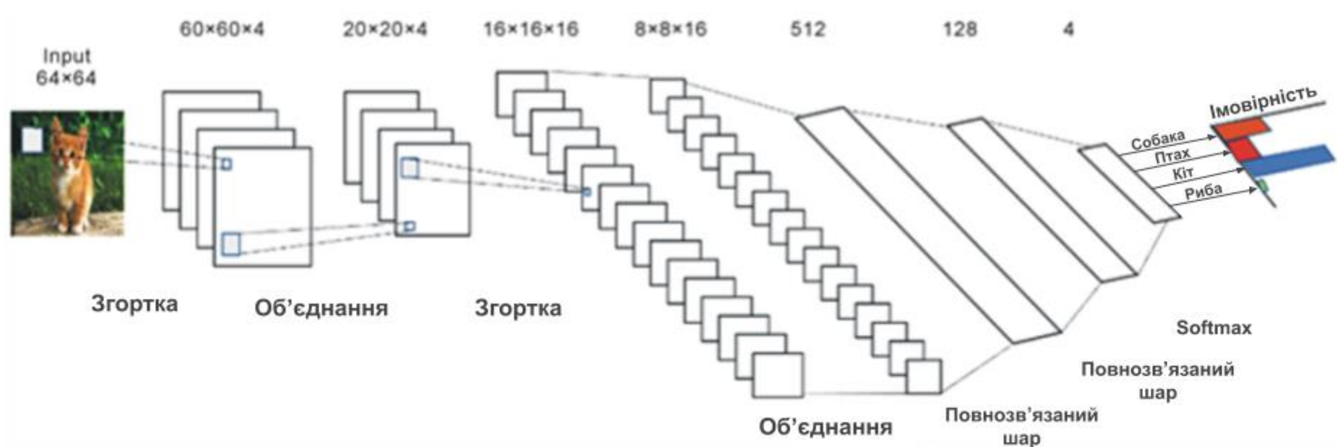

Рисунок 2.3 – Приклад згорткової мережі з двома згортковими шарами та двома повністю сполученими шарами, що здійснює класифікацію між собакою, котом, птахом та рибою [18]

Навчання згорткових нейронних мереж є обчислювально ефективнішим ніж навчання повно-зв'язаних нейронних мереж, і передбачає ті ж кроки. Тож ті самі прийоми та значна частина коду використовується з невеликими модифікаціями. Процес починається з ініціалізації всіх ваг і параметрів до випадкових значень. Далі вводяться вхідні дані і обчислюється вихідний вектор імовірностей, елементи якого відповідають ймовірності кожного класу. Далі обчислюється помилка як сума різниці ймовірностей із реальним значенням класу прикладу. Далі проводиться зворотне поширення помилки для оновлення ваг мережі. Ваги оновлюються пропорційно їх внеску в помилку. Різниця із звичайними повно-зв'язними нейронними мережами

полягає в тому, коли ваги фільтрів потрібно оновлювати. Потрібно зауважити, що єдина відмінність між згортковим шаром і повністю зв'язаним шаром полягає в тому, що нейрони підключені лише до невеликої області попереднього шару і однакові з'єднання використовуються для всіх областей. Таким чином, для обчислення оновлення ваг для всіх вхідних прикладів, що використовують спільні ваги, можна взяти середнє значення градієнтів помилки.

2.4 Глибокі залишкові мережі (ResNets)

Як було сказано раніше, глибші мережі можуть вивчати більш складні та корисні функції, ніж менш глибокі моделі. У майбутньому, глибокі мережі будуть ще глибшими. У міру поглиблення стандартних повністю пов'язаних багатошарових перцептронів починають виникати такі проблеми як вибух та затухання градієнтів [20], що порушує збіжність функції та перешкоджає навчанню моделі. Проблема затухання градієнтів також починає виникати зі збільшенням глибини, точність спочатку поступово затухає, а потім швидко деградує. У *залишкових мережах* це вирішується подачею входів шару на інші шари вперед – це допомагає шарам за необхідності вивчити тотожне відображення. Тож замість того, щоб просто сподіватися, що глибокі мережі зможуть вивчити тотожне відображення попередніх шарів, архітектура мережі спеціально робить так, щоб стек шарів вивчив необхідне відображення. На рис. 2.4 показаний блок такої мережі. Як показано, введення скорочено до наступних шарів. Ці стеки шарів зазвичай містять згорткові мережі. Стек шарів вивчає відображення $H(x)$ в нейронній мережі, коли x – це вхідні дані, що вводяться в перший шар. Використовуючи гіпотезу про те, що кілька нелінійних шарів можуть асимптотично наближати складні функції (ця гіпотеза залишається відкритим питанням) [21], можна сказати, що однаковий стек шарів може набувати форми $H(x) - x$ (виходячи з припущення про сумісний вимір).

Таким чином, шари наближаються до значення нової функції, $F(x) = H(x) - x$, а вхід до наступного стеку шарів залишається таким самим $F(x) + x$, тобто $H(x)$.

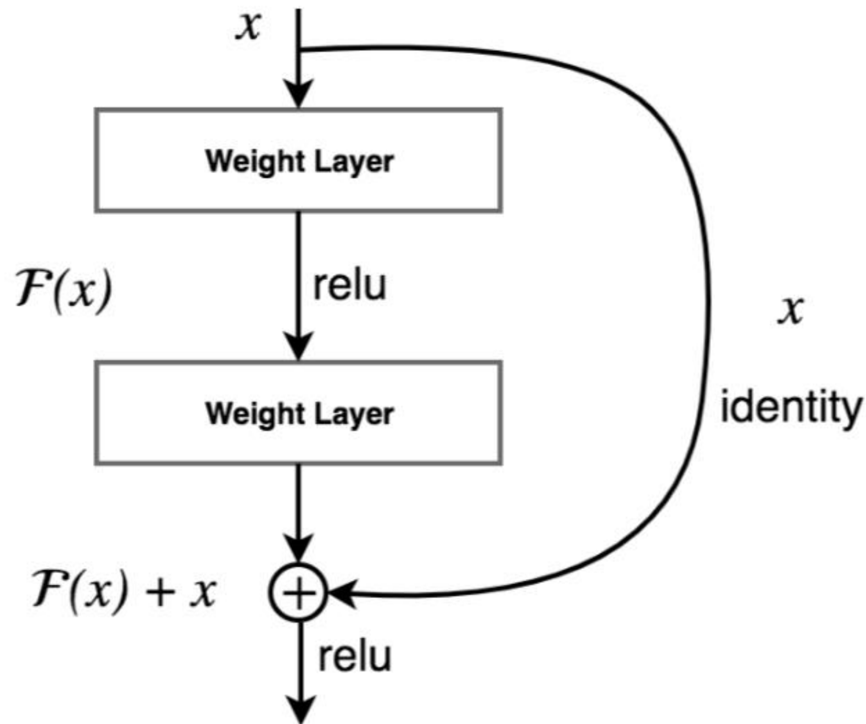

Рисунок 2.4 – Блок, з якого складаються залишкові мережі (ResNets). Кількість шарів всередині блока та кількість таких блоків в мережі можуть відрізнятися [17].

Ця передача вхідних шарів наперед базується виключно на прямому вирішенні проблеми затухання градієнтів. Як можна побачити, мережа може вивчити обидві функції, але різниця полягає в простоті навчання. Отже, залишкові мережі змогли зменшити проблему затухання градієнтів і, таким чином, дозволили створювати більш глибокі мережі, аж до 150 шарів.

2.5 Попередня робота з застосування машинного навчання у класифікації шкідливого програмного забезпечення

Зловмисне програмне забезпечення використовується для нападу на критичну інфраструктуру, для шпигунства проти держав, для крадіжки приватної інформації або проведення фінансових шахрайств. Майже всі атаки використовують мережу як носій. При цьому, майже всі системи виявлення зловмисних програм у галузі використовують або підхід на основі підпису, або підхід на основі виявлення

аномалій. Підпис – це унікальна послідовність байтів, яка присутня у шкідливому бінарному файлі та у файлах, які були пошкоджені цією шкідливою програмою [22]. Методи на основі підписів використовують унікальні підписи, розроблені антивірусними компаніями, використовуючи відомі зловмисні програми. Цей підхід швидкий і має високу точність, але він не в змозі виявити раніше небачені зловмисні програми. Тож, як правило, тільки після зараження новим зловмисним програмним забезпеченням численних систем аналітикам вдається створити підпис і додати його в базу. Крім того, база даних підписів повинна бути підготовлена вручну, що є трудомістким процесом [23]. За підходу, заснованого на виявленні аномалій, антивірусні компанії формують базу даних про дії, які вважаються безпечними. Якщо процес порушує будь-яке з цих заздалегідь визначених правил, він позначається як шкідливий [24]. Хоча за допомогою методу, що базується на виявленні аномалій, можна виявити нові небачені зразки зловмисного програмного забезпечення, кількість хибно позитивних результатів є дуже високою.

Інший метод, що часто використовується, це метод на основі евристики [25]. В цьому підході аналітики використовують методи машинного навчання для створення класифікатора шкідливих програм. Статичні, динамічні, візуальні представлення особливостей програм або їх комбінація використовуються для навчання класифікатора на наборі даних, що складається зі злоякісних і доброякісних двійкових файлів. Для класифікації та виявлення зразків зловмисного програмного забезпечення були запропоновані різні методи машинного навчання, такі як опорно-векторні машини, випадкові ліси, дерева рішень, Naive Bayes, кластеризація за допомогою K-середнього, Gradient Boost та Ada-boost. Деякі з цих прийомів, що застосовуються в літературі, обговорюються в цьому підрозділі. У решті підрозділу розглядається минула робота над підходами, заснованими на статичному, динамічному аналізу та методу заснованому на візуалізації.

Статичний аналіз включає вилучення статичних ознак з двійкового файлу за допомогою інструментів бінарного аналізу. Метью та ін. [26] використовували детальну інформацію, таку як список функцій DLL, викликаних виконуваним файлом, кількість унікальних системних викликів, що використовуються в межах

кожної DLL, список DLL, використовуваних виконуваним файлом, символи для друку або рядки, закодовані у двійковому файлі, та послідовності байтів із використанням hex-дампу як ознаки для класифікації. Вони використовували алгоритм Multinomial Naive Bayes, щоб класифікувати набір шкідливих програм з 3265 шкідливих та 1001 доброякісних зразків і повідомили про точність 97,11%. Вони одними з перших спробували виконати аналіз зловмисного програмного забезпечення з використанням методів майнінгу даних.

Колтер та ін. [27] вивчали точність класифікації різних методик машинного навчання, таких як Naive-Bayes, опорно-векторі машини, дерева рішень та їх розширені версії, у задачі класифікації шкідливих програм в різні сімейства, використовуючи ознаки, запропоновані Schultz et al. [26] та показав, що прискорені дерева рішень давали найкращу точність для класифікації. Zhang та Reeves [28] запропонували автоматизований метод статичного аналізу для ідентифікації метаморфних зловмисних зразків програмного забезпечення, які, як правило, уникають звичайних методів виявлення на основі підписів. Вони обчислили ступінь подібності між розбірками двох виконуваних файлів, використовуючи список викликів функцій бібліотеки, здійснених двома бінарними файлами, і показали хорошу точність у виявленні метаморфічних варіантів.

Контрольовані алгоритми навчання (з наглядом) вимагають великої кількості мічених двійкових файлів як злоякісного, так і доброякісного класів для успішного виявлення зловмисних програм. Але отримати такий мічений набір даних доволі важко, тому Santos et al. [22] запропонували напівконтрольований метод, щоб виявляти раніше небачені зразки зловмисного програмного забезпечення. Напівконтрольований підхід до навчання дуже корисний, коли кількість доступних мічених даних невелика. Він намагається навчити класифікатор, використовуючи доступні мічені дані, і намагається передбачити мітки для немічених даних протягом декількох ітерацій, і в кожній ітерації деякі нерозмічені дані, прогнози класів яких перевищують встановлений поріг впевненості, переміщуються в категорію розмічених даних. Для представлення виконуваних файлів використовувався підхід розподілу байтових n-грам. Автори використовували LLGC (Learning with Local and

Global Consistency) [29], напівконтрольований алгоритм, у своїй роботі та повідомили про точність 88,25% у виявленні зловмисних програм. Хоча точність їх моделі менша порівняно з іншими моделями, про які повідомлялося раніше в літературі, їм вдалося зменшити кількість мічених зразків, необхідних для навчання, всього до 2 тисяч, зберігаючи доволі високу точність класифікації.

Siddiqui та ін. [30] представили новий підхід використання послідовностей інструкцій змінної довжини для виявлення черв'яків серед доброякісних двійкових файлів за допомогою машинного навчання. Вони використовували випадкові ліси та дерева рішень, щоб класифікувати набір даних, що складається з 1444 хробаків та 1330 доброякісних файлів та повідомили про точність класифікації 96%. Кан та ін. [31] застосували машинне навчання на послідовностях n-опкодів за допомогою класифікатора SVM (опорно-векторної машини) та отримали точність 98% у виявленні зловмисних програм. Оп-коди, мнемоніка машинного коду та такі методи, як косинусна схожість та критичні послідовності інструкцій, були використані в процесі їх виявлення. Сун та ін. [32] досліджували використання послідовності API викликів для виявлення зловмисних програм. Вони показали, що всі версії одного і того ж зловмисного програмного забезпечення мають спільний профіль поведінки, який можна вирахувати через послідовності API викликів.

Мозер та ін. [33] запропонували схему, засновану на техніці обфускування, щоб вивчити недоліки підходів статичного аналізу. Їхні експерименти показали, що лише статичного аналізу недостатньо для ефективного аналізу зловмисного програмного забезпечення. Оскільки статичного аналізу можна легко уникнути, якщо зловмисне програмне забезпечення обфусковано чи запаковано, для аналізу потрібні додаткові надійні поведінкові ознаки. Це розуміння приводить до різних підходів динамічного аналізу.

Спільним для всіх підходів, заснованих на *динамічному аналізі*, є виконання бінарних зразків у контрольованому середовищі для аналізу поведінкових особливостей всередині віртуальної машини. Zolkipli та ін. [34] запропонували комплексний підхід до аналізу зловмисних програм на основі поведінки та класифікації зловмисного програмного забезпечення на нові групи з використанням

методів штучного інтелекту. Вони використовували пастку (honeypot) [41] та системи виявлення вторгнень, такі як HoneyClients та Amun, для збору зразків шкідливих програм для аналізу. Звіт про поведінку для кожного зібраного зразка формувався за допомогою віртуальних машин, таких як CWSandbox [35] та Anubis [36], і кожен звіт аналізувався вручну. Потім за допомогою методів штучного інтелекту зразки зловмисного програмного забезпечення були згруповані у черв'яків та троянів. Єдиним обмеженням було те, що вони не автоматизували аналіз звітів, тож враховуючи величезний обсяг зловмисного програмного забезпечення, що генерується в наш час, людський аналіз звітів неможливий.

Christodorescu та ін. [37] автоматизували трудомісткий процес ручного аналізу. Вони запропонували методику збору зловмисної поведінки шкідливого програмного забезпечення, яку не проявляють доброякісні програми. Для цього вони порівняли поведінку зразків зловмисного програмного забезпечення з поведінкою виконання набору доброякісних зразків. Їхній підхід створює стисле представлення шкідливої поведінки, і може сильно покращити розуміння дослідників безпеки щодо шкідливих програм взагалі.

Rieck та ін. [38] запропонували новий метод автоматизованого знаходження нових класів шкідливих програм із подібним типом поведінки (кластеризація) та класифікацію раніше небачених шкідливих програм до цих виявлених класів (класифікації) за допомогою машинного навчання. Використовуючи і кластеризацію, і класифікацію, вони застосували інкрементальний (поступовий) підхід для обробки поведінки великої кількості шкідливих програм. Поступовий підхід суттєво зменшив час роботи поточних методів аналізу. Вони записували ознаки зміни стану, такі як відкриття файлу, блокування м’ютекса, активність мережі, інфікування запущених процесів або встановлення ключів в реєстрі і, таким чином, описували простір поведінки зловмисного програмного забезпечення векторним простором. У своєму експерименті вони використали понад 10 000 зразків шкідливих програм, що належать до 14 різних сімей. Ці зразки зловмисних програм були зібрані за допомогою пасток. Вони повідомили про точність 88% класифікації сімейств, використовуючи простий класифікатор на основі опорно-векторних машин.

Обмеженням їхньої роботи є те, що вони розглядали лише один шлях виконання двійкового коду в своєму аналізі, який зупинявся, якщо програма сама виходила із виконання.

Кі та ін. [39] розробили нову методику, застосувавши алгоритми вирівнювання послідовності ДНК (MSA та LCS) до послідовності API визовів, що генерується зловмисним програмним забезпеченням, та вилучили деякі загальні шаблони послідовностей API визовів, знайдені в різних типах шкідливих програм. Алгоритм вирівнювання послідовності ДНК допомагав їм у знаходженні структури у послідовностях викликів API, яку в іншому випадку було б важко зафіксувати. Використання структури послідовностей викликів API та критичних послідовностей викликів API у своїй моделі допомогло їм досягти високої точності у виявленні раніше невідомих зловмисних програм. Оскільки послідовність викликів API може бути вилучена майже з усіх систем, їх метод можна використовувати для повсюдного виявлення зловмисного програмного забезпечення. Вони показали певні функції або шаблони послідовностей викликів API, які існували серед шкідливих програм навіть у різних категоріях, які можна використовувати для виявлення нових невідомих зловмисних програм. Обмеженням їх роботи є те, що алгоритми вирівнювання послідовності ДНК є високозатратними та трудомісткими алгоритмами, які сповільнюють процес виявлення.

Anderson та ін. [40] представили новий підхід, моделюючи графи ланцюгів Маркова з трасування, згенерованого виконанням двійкового файлу на Ether, платформі для аналізу шкідливих програм. Вершини на графах (насправді, ланцюгів Маркова) – це одиниці трасування (інструкції), а ймовірність переходу оцінюється з даних, що містяться у звіті про трасування. Ядра графу відображають схожість між графами трасування як на глобальному, так і на локальному рівнях і таким чином створюють матрицю схожості або матрицю ядер між графами. Локальна схожість між графами вимірюється за допомогою ядра Гаусса, тоді як глобальна схожість вимірюється за допомогою спектрального ядра. Ці матриці подібності подаються як вхідні дані для опорно-векторної машини для виконання класифікації. У своєму експерименті, автори використали набір даних, що включає 615 доброякісних зразків

та 1615 злоякісних зразків та повідомили про точність класифікації 96,41%. Цей підхід, хоча і показує хороші результати, обмежений дуже високим часом на обчислення, що робить його непридатним для реальних програм.

```
// open the source-file as a memory-mapped file
HANDLE src = NtOpenFile("C:\sample.exe");
HANDLE sectionHandle = NtCreateSection(src);
void *base = NtMapViewOfSection(sectionHandle);

// don't overwrite the target
if (NtQueryAttributesFile("C:\Windows\sample.exe") !=
    STATUS_OBJECT_NAME_NOT_FOUND)
    exit(1);

// open the target
target = NtCreateFile("C:\Windows\sample.exe");

void *p = base;
while(p < base + fileLen) {
    NtWriteFile(target, p++);
}
```

Фрагмент псевдо-коду

```
File|C:\sample.exe
            open:1
Section|C:\sample.exe
            open:1, map:1, mem_read: 1
File|C:\Windows\sample.exe
            query_file:0, create:1, write:1
Section|C:\sample.exe -> File|C:\Windows\sample.exe
            mem_read - write: (fileLen)
```

Профіль поведінки

Рисунок 2.5 – Приклад профілю поведінки [41]

Bayer та ін. [41] запропонували систему, яку можна легко масштабувати, щоб автоматично кластеризувати велику кількість зразків шкідливих програм у групи/класи на основі їх поведінки. Вони розширили систему Anubis [36], щоб включити відстеження забруднень та додатковий аналіз мережі та створили автоматизовані трасувальні звіти для всіх зразків шкідливих програм, що

використовують цю розширену систему. Профілі поведінки, зображені на рисунку 2.6, створюються шляхом абстрагування системних викликів, їх залежностей та мережевих дій до узагальненого вигляду, що складається з об'єктів операційної системи та операцій, які виконувались над цими об'єктами для кожного звіту про трасування. Ці звіти про поведінку можуть більш абстрактно характеризувати діяльність програми, і слугують вхідними даними до алгоритму кластеризації, LSH (Locality Sensitive Hashing), [42] – сублінійного методу, що апроксимує розв'язання задачі знаходження найближчого сусіда. Щоб протестувати масштабованість запропонованого способу, автори згрупували набір даних, що складається з 75 000 шкідливих файлів менш ніж за 3 години.

Nari та ін. [43] представили основу для автоматизованої класифікації зразків шкідливих програм з використанням їх мережевої діяльності. Мережевий слід, що генерується при виконанні бінарного файлу, зберігається у вигляді файлу pcap. Файл pcap – це файл-дамп мережевої активності, який використовується такими додатками, як WireShark, і подається як вхід до моделі. З файлів pcap витягується інформація про комунікацію, таку як IP-адреса, номер порту та використовуваний протокол, що разом в цілому показують мережеві потоки. З цих даних будується поведінковий граф, щоб абстрактно представити мережеву поведінку шкідливих програм та залежності між мережевими потоками. Для класифікації шкідливих програм застосовуються багато ознак, що беруться із графа поведінки, включаючи розмір графа, фактор розгалуження кореня, середній фактор розгалуження, максимальний фактор розгалуження та кількість конкретних вузлів. Автори використовували набір даних із 9610 зразків зловмисних програм та мітили їх за допомогою 11 антивірусів. Зразки зловмисних програм були відскановані за допомогою всіх 11 антивірусів, а мітка, яка виявилася вище встановленого порогу, була присвоєна вибірці. Для класифікації використовувались алгоритми класифікації, що надаються в Java-бібліотеці WEKA, [44], і було показано, що дерево рішень J48 дає точність 95,23%, що перевершує інші алгоритми класифікації.

З вищенаведеного обговорення видно, що або статичного аналізу, або лише динамічного аналізу недостатньо для точної та ефективної класифікації зразків

шкідливих програм. При використанні лише одного з цих підходів, аналізу можна легко уникнути, використовуючи різні методи ухилення від аналізу, такі як обфускування коду або різні методи зупинення виконання. Також, лише за допомогою динамічного аналізу, ми не в змозі дослідити всі шляхи виконання виконуваного файлу. Отож, в області аналізу шкідливих програм розгорнулася окрема сфера дослідження – сфера гібридного аналізу, який використовує одночасно поєднання статичних та динамічних методів для підвищення точності виявлення та класифікації шкідливих програм.

Santos та ін. [45] запропонували новий підхід до використання разом як статичних, так і динамічних ознак для навчання класифікатора шкідливих програм під назвою OPEM. Вони поєднали частоту виникнення операційних кодів, *статичну ознаку*, із слідом виконання (трасуванням) виконуваного файлу, отриманим за допомогою моніторингу виконуваних операцій, зроблених системних викликів та винятків, *динамічною ознакою*, для тренування своєї моделі та показали, що гібридний підхід працює краще, ніж обидва підходи, що працюють окремо. Вони підтвердили ефективність свого підходу за допомогою двох різних наборів даних, а також безлічі машинних алгоритмів, Дерева рішень, KNN, Байєсової мережі та SVM.

Islam та ін. [46] також запропонували гібридну модель класифікації двійкових файлів на доброякісні та шкідливі файли, використовуючи як динамічні, так і статичні ознаки. Вони використовували такі статичні ознаки як друковані рядки, і частоту довжин функцій та такі динамічні функції як параметри API та назви функцій API. Для тестування своєї моделі вони використали 2939 зразків злоякісних виробів та 541 доброякісних зразків. Вони використовували інтегровані метакласифікатори, такі як SVM, IB1, DT та RF, для класифікації зразків зловмисних програм і повідомляли про точність 97.055%. Їх результати були поліпшенням порівняно із попередніми результатами в літературі.

Гібридні підходи є дуже перспективними, оскільки вони суттєве вдосконалюють ефективність класифікації порівняно з тільки статичними або тільки динамічними підходами. Далі, розглянемо підхід із застосуванням представлення файлів у вигляді зображень.

Для візуалізації та редагування двійкових файлів доступно багато утиліт, що називаються Hex-редакторами, але вони просто відображають файл у шістнадцятковому та ASCII форматі і не передають аналітику будь-яку інформацію про загальну структуру файлу. Різні дослідники намагалися візуалізувати бінарні файли таким чином, щоб можна було відразу побачити файл як єдине ціле.

Helfman [47] застосував техніку візуалізації даних на основі точкових графіків для візуалізаці програм і показав, що візуалізація може бути корисною для виявлення структури програмного забезпечення. Точковий графік – це техніка візуалізації структури у збігах рядків (тексту) і може давати наочний огляд структури величезної системи. Такі візуалізації структури корисні для проектування програмних систем за допомогою Successive Abstraction – шаблона проектування програмного забезпечення, який допомагає усунути надмірність. Послідовність розбивається на лексеми, а лексеми розміщені навколо двох осей з крапкою, де лексеми збігаються та пропуском там, де вони не збігаються.

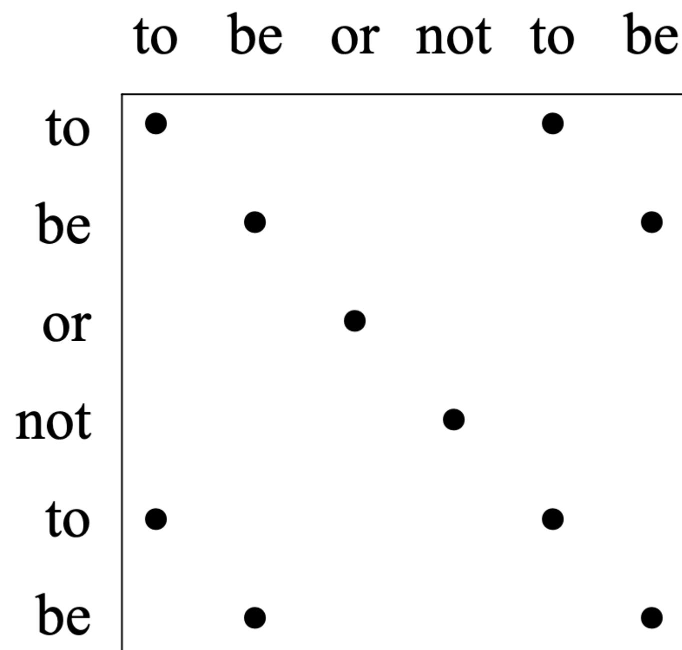

Рисунок 2.6 – Шість слів Шекспіра, зображені, використовуючи метод візуалізації точкового графіка [47]

Бінарні файли – це різноманітні примітивні типи файлів, такі як аудіо, зображення, відео, текст і виконуваний код. Conti та ін. [48] запропонували автоматизовану техніку візуалізації бінарних файлів з використанням візуалізації діаграми байтів, методу відображення або класифікації регіонів у файлі. Діаграма байтів генерується, розглядаючи кожен байт у файлі як значення пікселя, 0 – як чорний, а FF (255 у десятковому виді) – як білий. Використовуючи діаграму байтів, вони автоматизували проблему пошуку початків та кінців кожної окремої області в двійковому файлі, а також передбачали її примітивний тип даних. Примітивний тип – це однорідна ділянка двійкового файлу, яка має споріднену структуру, наприклад послідовність випадкових чисел або подібні групи даних. Метод візуалізації байтових діаграм допоміг у пошуку виразних структур, навіть таких перетворень як кодування та шифрування. Їх робота обмежувалася лише визначенням примітивних шаблонів у двійковому файлі. З рисунка 2.4 можна побачити, що різні розділи двійкового зображення мають різну текстуру.

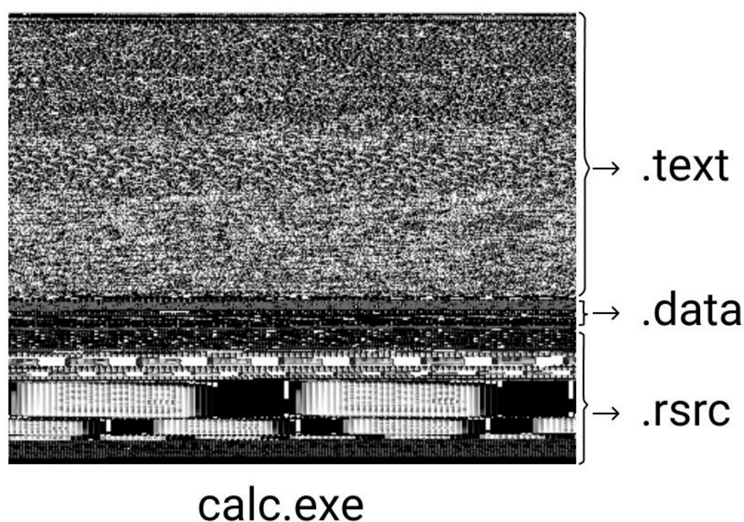

Рисунок 2.7 – Візуалізація бінарного файла за допомогою методу візуаліції байтових діаграм, із зазначеними примітивними типами [48]

Yoo та ін. [49] намагалися виявити та візуалізувати віруси, вбудовані у виконуваний файл, використовуючи самоорганізуючі карти (SOM) [50], не використовуючи жодної інформації про підписи. SOM – це тип штучних нейронних

мереж, що використовується для візуалізації багатовимірних даних. SOM поєднує векторне квантування та векторну проекцію та створює топологічно впорядковане відображення даних. Вони показали, що так само, як і кожна людина має унікальну ДНК, кожна сім'я вірусів має вірусну маску, яку видно, коли двійковий файл візуалізується за допомогою SOM. Вони використовували ці унікальні ознаки, ідентифіковані за допомогою SOM, для прогнозування сімейства вірусів. Вони показали, що за допомогою вірусної маски, яку виробляє SOM, також можна ідентифікувати варіант кожного вірусу.

Nataraj та ін. [51] першими вивчали використання візуалізації байтових графіків для автоматичної класифікації шкідливих програм. Вони перетворили всі зразки зловмисного програмного забезпечення на зображення байтових діаграм сірого кольору та використовували текстуру зображення для класифікації зловмисного програмного забезпечення. Вони використовували абстрактну техніку представлення даних, GIST, для обчислення текстурних особливостей із зображень. Їх набір даних складався із 9458 зразків шкідливих програм, що належать до 25 різних класів, зібраних із системи Anubis [36]. Вони використовували глобальні ознаки на основі зображень для навчання моделі K-найближчих сусідів, використовуючи евклідову дистанцію між об'єктами як міру відстані, щоб класифікувати зразки зловмисних програм у відповідні класи та отримали точність 97,18%. Отримані результати порівнянні з результатами динамічного або гібридного аналізу та показали, що класифікація зловмисного програмного забезпечення за допомогою методу на основі обробки зображень може класифікувати шкідливі програми швидше, ніж існуючі динамічні підходи.

Han та ін. [52] запропонували новий спосіб візуалізації зловмисного програмного забезпечення з використанням послідовностей оп-кодів для виявлення та класифікації зразків шкідливих програм. Вони використовували матриці зображень для візуального представлення зловмисного програмного забезпечення, які допомагали виявляти особливості зловмисного програмного забезпечення, а також швидко знаходити схожість між різними зразками. Спочатку бінарний файл розбирався за допомогою IDA Pro або OllyDgb, а послідовність оп-коду ділилася на

блоки. Потім вони використовували дві хеш-функції і для кожного блоку послідовності оп-кодів обчислювали координату та значення RGB, використовуючи дві хеш-функції. Потім вони перетворювали всі значення RGB до відповідних координат у матриці розмірності 8 на 8, щоб отримати матрицю зображення. Вони використовували "вибіркове узгодження областей" для обчислення подібності між матрицями зображень та оцінювали їх модель на зразках шкідливих програм з 10 різних сімей. Їх результати показали, що матриці зображень зловмисних програм з однієї сім'ї мають більш високий показник схожості, ніж зі зловмисними програмами з іншої сім'ї, і матриці зображень можуть ефективно класифікувати сімейства шкідливих програм.

Макандар та ін. [53] перетворювали зловмисне програмне забезпечення у двовимірне зображення в відітнках сірого та класифікували зразки, використовуючи текстурні ознаки. Вони діставали глобальні ознаки на основі текстури із файлів, використовуючи вейвлет-перетворення Габора та GIST, та використали набір даних Mahenhur [53] для експериментів, який складається з 3131 зразків двійкових файлів із 24 унікальних сімейств шкідливих програм. Вони використовували штучні нейронні мережі для класифікації шкідливих програм і повідомляли про точність 96,35%.

Liu та ін. [54] запропонували поступовий підхід до автоматичного присвоєння зловмисних програм до відповідної родини та виявлення нових шкідливих програм. Вони використовували комбінацію ознак: сірий байтовий точковий графік, n-грами op-коду та імпорт функції. Модуль прийняття рішень використовує ці ознаки для класифікації зразків шкідливого програмного забезпечення до відповідних сімей та для виявлення нових невідомих зловмисних програм. Вони використовували алогоритм спільного найближчого сусід (SNN) як алгоритм кластеризації для виявлення нових сімей шкідливих програм. Їх модель оцінюється на наборі даних, що складається з 21740 зразків зловмисних програм із 9 різних сімей, і вони повідомляють про точність класифікації 98,9% та точність виявлення 86,7%.

Висновки за розділом 2

У цьому розділі було розглянуто теоретичні основи машинного навчання та нейронних мереж, включаючи сучасні архітектури згорткових нейронних мереж (CNN) та залишкових мереж (ResNet). Було описано застосування методів машинного навчання до проблем класифікації зловмисного програмного забезпечення за сімействами, а також виявлення шкідливого ПЗ серед доброякісного. Було окреслено, як машинне навчання допомогає у різних підходах до аналізу (статичному, динамічному та гібридному). Як можна побачити, існуючі статичні та динамічні методи виявлення зловмисного програмного забезпечення, що використовуються на практиці, недостатньо ефективні проти нових та мінімально змінених старих образців шкідливого коду:

1. Динамічний аналіз нових, до того невідомих шкідливих програм потребує використання значних обчислювальних ресурсів або часу (у разі динамічного аналізу системних викликів або профілю поведінки) або має низьку ефективність через перевірку лише одного шляху виконання коду (у випадку одноразового аналізу поведінки у віртуальному середовищі);

2. Статичний аналіз на основі підписів є абсолютно неефективним проти навіть мінімально модифікованих шкідливих програм, і тим паче абсолютно нових, тоді як статичні методи аналізу, що використовують машинне навчання, можуть знаходити структурну близькість та спрацьовують навіть для абсолютно нових одиниць шкідливого коду, і тому виглядають перспективними. Їх проблемою часто є висока складність та повільність роботи, що робить використання цих алгоритмів на практиці незручним.

У наступному розділі буде описано модель, побудовану в рамках цієї дипломної роботи, розглянуто набір даних для її тренування і проведено оцінку точності та повноти. Додатково буде оцінено швидкість перевірки файлів за допомогою розробленої моделі для порівняння із попередніми роботами, що відомі в літературі.

РОЗДІЛ 3
ПОБУДОВА МОДЕЛІ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Набір даних

Для цієї роботи було зібрано 65536 зразків зловмисного програмного забезпечення з репозиторію шкідливого програмного забезпечення VirusShare [55]. Портали-репозиторії типу VirusShare збирають програмне забезпечення за допомогою пасток, а також від користувачів зі всього світу, що самостійно надсилають файли для аналізу та обміну зразками шкідливих програм.

Так як для деяких типів доброякісних файлів можливе помилкове виявлення, для додаткової перевірки всі MD5 хеш-підписи зразків з VirusShare були відправлені на VirusTotal, сервіс що надає звіти про отриманий файл від десятків антивірусів. Тільки ті зразки, що були названі шкідливими більш ніж 50% антивірусів у звіті VirusTotal, помічалися як шкідливі.

Збір великої кількості доброякісних зразків програмного забезпечення, як виявилося, є нетривіальною задачею, і в процесі дослідження не було знайдено готових корпусів програм, що підішли би для цієї задачі. Тому було вирішено зібрати власний набір даних, скориставшись технікою, подібною до описаної Beisel та ін. [56], а саме зібрати набір даних з виконуваних файлів, що знаходяться на довірених комп’ютерних системах. Таким чином було зібрано 10451 унікальних одиниць доброякісних виконуваних програм з систем на базі Windows 7, Windows 10 та OS X.

Після відбору файлів відповідного розміру, набір даних був розділений на три частини: 18424 зразків для навчання (train dataset), 4448 зразків для вибору найкращої моделі (dev dataset), та 4013 зразків для об’єктивної оцінки точності обраної моделі (test dataset).

3.2 Представлення файлів у вигляді зображень

Загалом, всі файли можна розглядати як послідовність байтів, тобто десяткових чисел від нуля до 255. Отже, першим етапом візуалізації файлів було перетворення кожного двійкового файла у одновимірний масив байтів, що розглядається як масив значень пікселів у відтінках сірого (grayscale). Після цього одновимірний масив пікселів перетворюється у двовимірну матрицю, отримуючи таким чином представлення двійкового файлу у вигляді зображення. Для цієї роботи ширина матриці є фіксованою та становить 384 байти, тоді як висота зображення змінюється і залежить від розміру файлу, аналогічно до попередніх робіт (наприклад, Singh [57]). Для простоти навчання, було обрано лише зразки, розмір зображення яких можна змінити до 384x384 без істотного викривлення – загальним розміром від 76800 до 192000 байт. Останнім кроком в перетворенні, для більшої наглядності, зображення у відтінках сірого перетворюються у кольорове RGB зображення за допомогою кольорової карти viridis [58].

З зображень на рисунку 3.1 та 3.2 ми можемо легко помітити, що є деяка текстурна схожість серед шкідливих програм, і деякі відмінності між доброякісними та злоякісними програмами.

Ці структурні відмінності пов’язані з використанням шкідливими зразками спільних модулів, функцій та прийомів. Наприклад, на другому зображенні у другому рядку рисунка 3.2, велика однорідна зона синього кольору є послідовністю NOP-кодів для архітектури Intel x86, тобто байту 0x90. Послідовності NOP-кодів часто застосовуються зловмисними програмами, що використовують вразливість переповнення буфера.

Більшість створених нових зловмисних програм повторно використовує попередній код, а для ухилення від виявлення шляхом відповідності підписів вони використовують такі методи, як обфускація, упаковка чи шифрування.

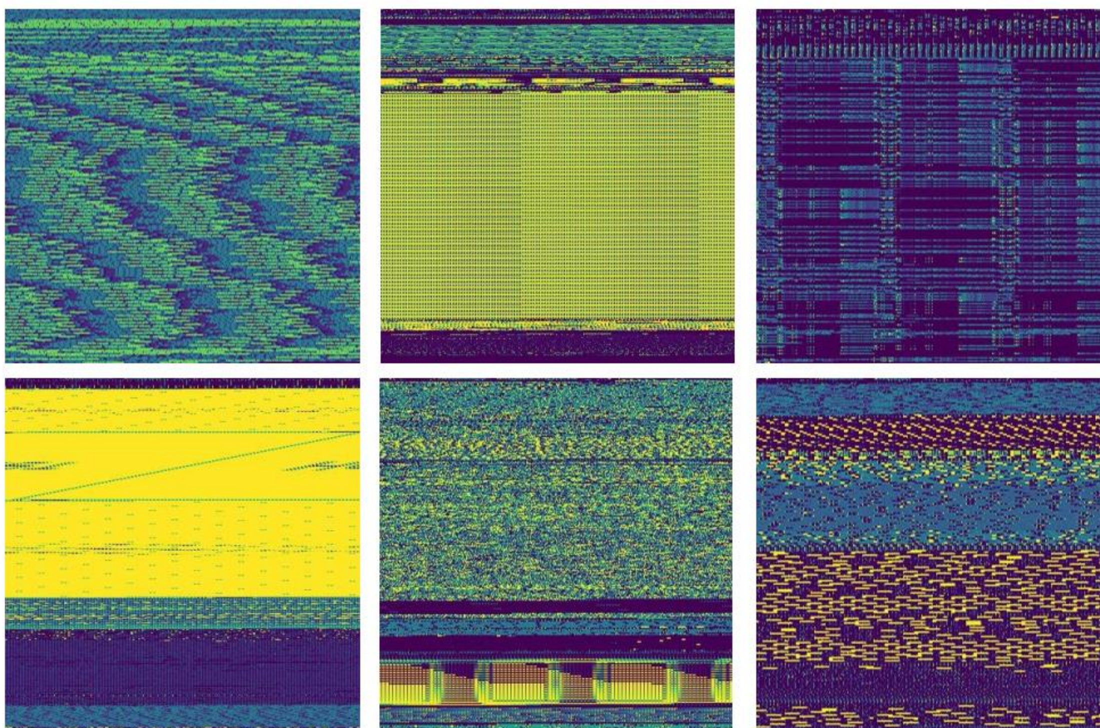

Рисунок 3.1– Зображення шести різних зразків доброякісного програмного забезпечення

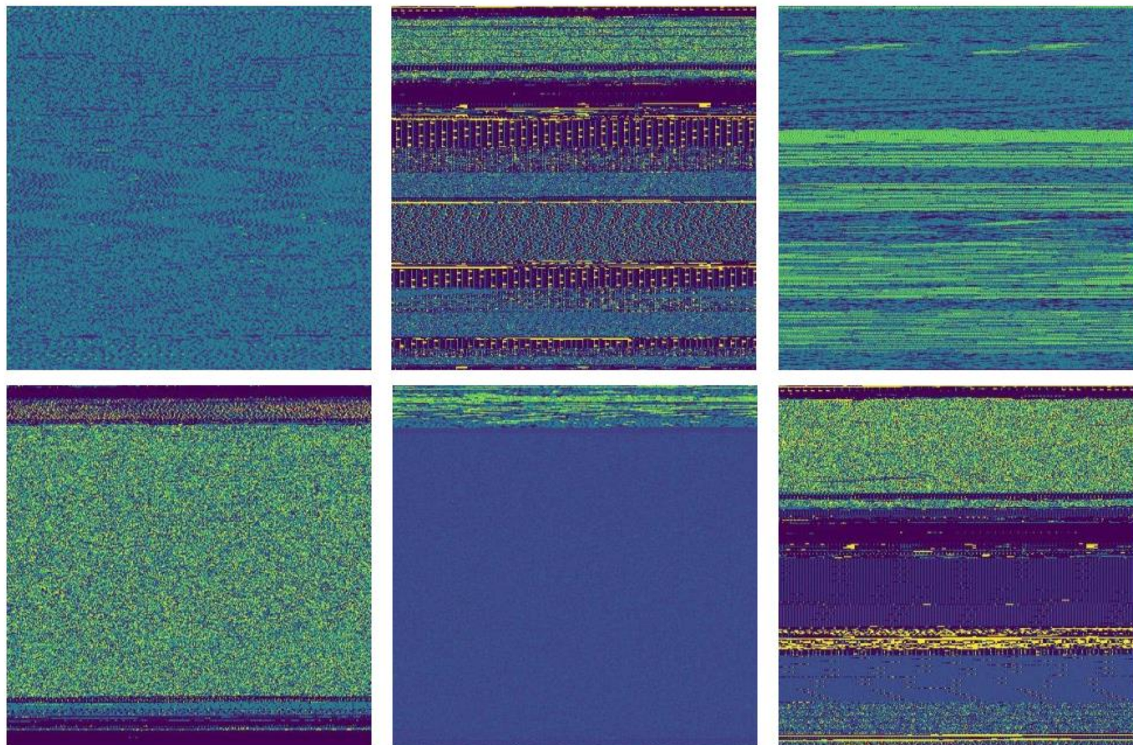

Рисунок 3.2 – Зображення шести різних зразків злоякісного програмного забезпечення

Через це, деякі зображення злоякісних зразків виглядають як шум, майже випадковий набір байтів, що є ознакою шифрування. Зображення доброякісних програм, навпаки, як правило, виглядають більш організовано, маючи більш визначену структуру.

Далі ми побачимо, як ми можемо використовувати ці подібності та відмінності для виявлення шкідливих програм серед невідомого програмного забезпечення.

3.3 Модель та аналіз результатів

Нейронні мережі є дуже ефективними у знаходженні сенсу та розпізнаванні структури у наборі зображень. У цій дипломній роботі використано 50-шарову залишкову нейронну мережу (ResNet50V2 [59]) для тренування класифікатора для виявлення зловмисних програм. Для кращих результатів та більш швидкого навчання, ініціалізація вагів моделі була виконана за допомогою ваг, натренованих на наборі даних ImageNet [60].

Для реалізації запропонованої системи була використана мова програмування Python та безліч різних бібліотек для Python. Весь код для підготовки даних та маркування за допомогою сервісу VirusTotal реалізований на Python. Для створення та зміни розміру зображень були використані бібліотеки numpy та PIL. Для навчання моделей був використаний Keras на базі TensorFlow. На рисунку 3.3 наведена функція, що виконує процедуру створення зображення із файлів.

Таблиця 3.1

Експериментальні результати

Модель	Набір зразків програмного забезпечення				Точність
	Всього зразків	Навчальний набір	Валідаційний набір	Тестовий набір	
ResNet50V2	26885	18424	4448	4013	92,74%

ResNet50V2 (з шаром виключення)					96,74%

```
def create_image_from_file(path_to_input, output_folder):
    # load file
    img = np.fromfile(path_to_input, dtype='uint8')

    # resize to (None, FIXED_WIDTH)
    num_pixels = img.shape[0]
    columns = FIXED_WIDTH
    rows = math.ceil(num_pixels / 384)
    total_size = columns * rows
    img.resize(total_size)
    img = img.reshape((rows, columns))

    # convert to RGB using a cmap
    cmap = plt.get_cmap('viridis')
    rgba_img = cmap(img)
    img = np.delete(rgba_img, 3, 2)

    # resize to (FINAL_SIZE, FINAL_SIZE)
    img = np.uint8(img * 255)
    pic = Image.fromarray(img)
    pic = pic.resize((FINAL_SIZE, FINAL_SIZE), Image.ANTIALIAS)

    # save image as a file
    original_file_name = os.path.basename(path_to_input)
    save_path = os.path.join(output_folder, original_file_name)
    pic.save(save_path + '.jpg')
```

Рисунок 3.3 – Створення кольорового зображення розміром 384x384 пікселів із файлу з використанням бібліотек NumPy, Matplotlib та Pillow

За допомогою оригінальної моделі ResNet50V2 була отримана точність виявлення 92.74% на тестовому наборі, що є досить високим значенням, але враховуючи точність цієї моделі в 97,68% на тренувальному наборі, зрозуміло, що модель «перевчилася», тобто почала описувати шум в даних і мала велику дисперсію. Через це було вирішено модифікувати модель, додавши шар виключення (dropout) [15]. Шар виключення слугує ефективним регуляризатором нейронних мереж, тобто

запобігає їх перенавчанню завдяки тому, що виключає випадкові нейрони при кожному спрацюванні мережі під час навчання, таким чином змушуючи її запам'ятовувати більш істотні характеристики вхідних даних. Після цієї модифікації була отримана точність 96,74% на тестовому наборі даних, що є значним покращенням порівняно із звичайною моделлю ResNet50V2. На таблиці 3.1 зображена інформація про точність обох моделей. На таблиці 3.2 зображені додаткові оцінки точності найкращої моделі. Потрібно зауважити, що точність у таблиці 3.2 є так званим позитивним прогностичним значенням, тобто мірою відношення знайдених дійсно шкідливих файлів до всіх файлів, що були названі моделлю шкідливими (іншими словами, це відповідь на питання: «Серед зразків, що були названі моделлю шкідливими, який відсоток зразків є насправді шкідливим?»). Повнота, або чутливість, є мірою відношення знайдених дійсно шкідливих файлів до всіх шкідливих файлів (відповідь на питання: «Серед всіх шкідливих зразків у наборі даних, який відсоток був виявлений?»). F1-оцінка представляє собою збалансовану єдину метрику, що включає в себе позитивну точність та повноту.

Таблиця 3.2

Аналіз точності розробленої моделі

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">Передбачуваний клас</td><td rowspan="3">Точність</td><td rowspan="3">Повнота</td><td rowspan="3">F1-оцінка</td></tr>
<tr><td>Шкідливий</td><td>Добро-якісний</td></tr>
<tr><td rowspan="2">Справжній клас</td><td>Шкідливий</td><td>3311</td><td>84</td></tr>
<tr><td>Добро-якісний</td><td>58</td><td>560</td><td>98,27%</td><td>97,52%</td><td>0,9790</td></tr>
</table>

Як видно із таблиць 3.1 та 3.2, розроблена модель має високу точність, яка відповідає кращим показникам точності інших ефективних підходів, відомих в

літературі. На таблиці 3.3 можна побачити порівняння точності класифікації різних відомих моделей. Треба зауважити, що пряме порівняння точностей моделей у таблиці не є коректним через використання різних наборів даних. Таблиця слугує лише приблизним показником, що розроблена модель працює на рівні найкращих відомих моделей на основі статичних, динамічних та гібридних підходів, і при цьому запропонований підхід має декілька переваг:

Таблиця 3.3

Порівняння розробленої моделі з іншими відомими

Модель	Набір зразків програмного забезпечення		Точність, %
	Всього зразків	Тестовий набір	
LLGC (на основі байтових n-грам) [29]	2000	400	88,25
Random Forest, послідовності інструкцій змінної довжини [30]	—	1774	96,00
SVM, послідовності n-опкодів та інструкцій [31]	2520	—	98,00
SVM, графи ланцюгів Маркова з кластеризацією [40]	—	2230	96,41
Random Forest, гібридний підхід [46]	2939	—	97,05
ResNet50 з представленням програм у вигляді зображень (ця робота)	26885	4013	96,74

- швидкість — перевірка одного виконуваного файлу на процесорі Intel i5-8259U (4 ядра по 2.3ГГц) займає приблизно 2,5с і може бути сильно пришвидшеною з використанням GPU;
- безпечність — файли не виконуються та не розбираються;
- зрозумілість — аналітик може передивлятися структуру файлів, так само як і штучна нейронна мережа, бо файли переводяться у зображення, де гарно видно структуру;

● новизна — підхід використовує сучасні техніки обробки зображень та комп'ютерного бачення на основі глибоких нейронних мереж, що швидко розвиваються. Це підвищує потенціал підходу, так як із постійним покращенням технік обробки зображень, цей підхід становитиметься все кращим.

Висновки за розділом 3

Аналіз шкідливого програмного забезпечення на сьогодні є однією з важливих задач світової кібербезпеки. Із постійним збільшенням кількості нових зловмисних зразків, нам все більше стають потрібні способи швидкого автоматичного аналізу великих об'ємів нових програм.

У цьому розділі було описано створення моделі для автоматичного виявлення шкідливого програмного забезпечення з представленням двійкових файлів у вигляді зображень, за допомогою найсучасніших підходів глибокого машинного навчання. Як можна побачити з зображень, що наведені для прикладу, файли, зображені за допомогою запропонованого підходу, мають яскраво виражену текстуру, що відрізняється між програмами різних типів, та між сімействами зловмисного програмного забезпечення. Різниця та схожість цієї структури дозволяє створити класифікатор, в ролі якого в цій роботі виступає сучасна глибока нейронна мережа, ResNet50V2. Після модифікації моделі додаванням шару виключення, модель показала дуже високу точність, порівняну із найкращами моделями, відомими в літературі.

Підхід до автоматичного виявлення шкідливих зразків програмного забезпечення, запропонований та реалізований в цьому розділі, має декілька переваг над іншими високоточними статичними, динамічними та гібридними методами, особливо – висока швидкість роботи, відносна простота реалізації та безпечність.

ВИСНОВОК

Більшість методів класифікації зловмисних програм вимагають виконання зловмисного програмного забезпечення для фіксації його поведінки або використання методів розбору коду для прогнозування його поведінки. Як виконання зловмисного програмного забезпечення, так і розбір коду забирають багато часу і вимагають високих вимог до обчислень. У цій дипломній роботі було показано, що за допомогою представлення файлів у вигляді зображень можна досягти порівнянних результатів та значних поліпшень у швидкості класифікації. Крім того, порівняно з попередніми роботами з використанням візуального представлення файлів, набір даних, що використовувався для навчання класифікатора у цій роботі, був значно більший.

Як правило, поведінковий або статичний аналіз залежать від платформи користувача, тому необхідно мати різні класифікатори для різних платформ. Підхід на основі зображень, розроблений в цій роботі, не залежить від платформи, оскільки він класифікує файли на основі подібностей та відмінностей структури бінарних зразків. Крім того, цей метод є більш безпечним у порівнянні з підходами, що базуються на динамічному аналізі поведінки програм, оскільки двійкові файли перетворюються у формат зображення і ніколи не виконуються.

Модель, розроблена в цій дипломній роботі, вдосконалює попередні роботи, а також прокладає шлях для застосування сучасних методів глибокого навчання в сфері аналізу шкідливих програм.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. DIGITAL 2020: APRIL GLOBAL STATSHOT. [Electronic resource] – Access: https://datareportal.com/reports/digital-2020-april-global-statshot

2. Malware. [Electronic resource] – Access: https://www.avtest.org/en/statistics/malware/

3. Microsoft Word - Into the Web of Profit FINAL. [Electronic resource] – Access: https://www.bromium.com/wp-content/uploads/2018/05/Into-the-Web-of-Profit_Bromium.pdf

4. Sikorski M., H. A. (2012). Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software.

5. Mirai (ботнет). [Електронний ресурс] – Режим доступу https://uk.wikipedia.org/wiki/Mirai_(ботнет)

6. WannaCry. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/WannaCry

7. Szor, P. (2005). The Art of Computer Virus Research and Defense.

8. S. Staniford, V. P. and Weaver, N. (2002). How to own the internet in your spare time. In Proceedings of the 11th USENIX Security Symposium.

9. Spafford, E. H. (1989). The Internet worm incident. In Proceedings of the 2nd European Software Engineering Conference.

10. Ransomware. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Ransomware

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

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

13. Cohen, F. (1987). Computer Viruses: Theory and Experiments. [Electronic resource] – Access: http://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html

14. Honkela, A. (2001). Nonlinear switching state-space models. [Electronic resource] – Access: https://www.cs.helsinki.fi/u/ahonkela/dippa/

15. Srivastava, N. et al. (2015). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. [Electronic resource] – Access: http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

16. Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London)

17. He, Kaiming; Zhang, X. R. S. S. J. (2015). Deep Residual Learning for Image Recognition. eprint arXiv:1512.03385, ARXIV

18. Williams, T. et al. (2018) An Ensemble of Convolutional Neural Networks Using Wavelets for Image Classification [Electronic resource] – Access: https://m.scirp.org/papers/82297

19. Shamir, O. (2018). Are ResNets Provably Better than Linear Predictors? [Electronic resource] – Access: https://arxiv.org/pdf/1804.06739.pdf

20. Bengio, Y., Simard, P., and Frasconi., P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks.

21. Montufa, G., Pascanu, R., Cho, K., and Bengio., Y. (2014). Learning long-term dependencies with gradient descent is difficult on the number of linear regions of deep neural networks. NIPS.

22. Santos, I., Nieves, J., and Bringas, P. (2011). Semi-Supervised Learning for Unknown Malware Detection. In Symposium on Distributed Computing and Artificial Intelligence Advances in Intelligent and Soft Computing.

23. Kong D, Y. G. (2013). Discriminant malware distance learning on structural infor- mation for automated malware classification. In Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM.

24. Vinod, P., Jaipur, R., Laxmi, V., and Gaur, M. (2009). Survey on malware detection methods. In Proceedings of the 3rd Hacker's Workshop on Computer and Internet Security (IITKHACK?09).

25. Bazrafshan, Z., Hashemi, H., Fard, S. M. H, and Hamzeh, A. (2013). A survey on heuristic malware detection techniques. In Information and Knowledge Technology.

26. Schultz, M. G., Eskin, E., and Zadok, F. (2001). Data Mining Methods for Detection of New Malicious Executables. In In Proc. of the 22nd IEEE Symposium on Security and Privacy.

27. Kolter, J. and Maloof, M. (2004). Learning to detect malicious executables in the wild. In In Proc. of the 10th ACM Int. Conf. on Knowledge Discovery and Data Mining.

28. Zhang, Q. and Reeves, D. (2007). Metaware: Identifying Metamorphic Malware. In Proceedings of the 23rd Annual Computer Security Applications Conference.

29. Zhou, D., Bousquet, O., Lal, T. N., Weston, J., and Scholkopf, B. (2003). Learning with local and global consistency. In Advances in Neural Information Processing Systems 16: Proceedings of the 2003.

30. Siddiqui, M. and Wang, M. C. (2009). Detecting Internet Worms Using Data Mining Techniques. In Journal of Systemics, Cybernetics and Informatics.

31. BooJoong Kang, Suleiman Y. Yerima, K. M. S. S. (2016). N-opcode Analysis for Android Malware Classification and Categorization.

32. A. Sung, J. Xu, P. C. and Mukkamala, S. (2004). Static Analyzer of Vicious Executables (SAVE). In Proceedings of the 20th Annual Computer Security Applications.

33. Moser, A., Kruegel, C., and Kirda, E. (2007). Limits of Static Analysis for Malware Detection. In IEEE Computer Society.

34. Zolkipli, M., Jantan, A. (2011). An approach for malware behavior identification and classification. [Electronic resource] – Access: https://ieeexplore.ieee.org/document/5764001

35. Willems, C., Holz, T., and Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using Cwsandbox. In IEEE Security and Privacy.

36. Kolbitsch, C. (2011). Anubis. [Electronic resource] – Access: https://wepawet.cs.ucsb.edu/

37. Christodorescu, M., Jha, S., and Kruegel, C. (2007). Mining specifications of malicious behavior. In Proceedings of the 6th Joint Meeting of the European Software Engineer- ing Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESECFSE).

38. Rieck, K., Holz, T., Willems, C., Dussel, P., and Laskov, P. (2008). Learning and classification of Malware behaviour. In Detection of Intrusions and Malware, and Vulnerability Assessment.

39. Ki, Y., Kim, E., and Kim, H. K. (2015). A Novel Approach to Detect Malware Based on API Call Sequence Analysis. International Journal of Distributed Sensor Networks.

40. Anderson, B.and Quist, D., Neil, J., Storlie, C., and Lane, T. (2011). Graph Based Malware Detection Using Dynamic Analysis. In Journal in Computer Virology.

41. Bayer, U., Comparetti, P., Hlauschek, C., and Kruegel, C. (2009). Scalable, Behavior- Based Malware Clustering. In Proceedings of the 16th Annual Network and Distributed System Security Symposium.

42. Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In Proceedings of 30th Annual ACM Symposium on Theory of Computing, Dallas.

43. Nari, S. and Ghorbani, A. (2013). Automated Malware Classification Based on Net- work Behaviour. In Proceedings of International Conference on Computing, Networking and Communications (ICNC), San Diego.

44. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. (2009). The WEKA Data Mining Software. In ACM SIGKDD Explorations Newsletter.

45. Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. (2012). OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection.

46. Islam, R., Tian, R., Batten, L. M., and Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. In Journal of Network and Computer Applications.

47. Helfman, J. (1995). Dotplot patterns: A literal look at pattern languages. TAPOS, 2:31–41.

48. Conti, G., Bratus, S., Sangster, B., Ragsdale, R., Supan, M., Lichtenberg, A., Perez- Alemany, R., and Shubina, A. (2010). Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classification. In The proceedings of The Digital Foren- sic Research Conference DFRWS.

49. Yoo, I. S. (2004). Visualizing windows executable virus using self-organizing maps. In Proceedings of ACM workshop on Visualization and data mining for computer security.

50. Kohonen, T. (1995). Self-Organizing Maps. Springer.

51. Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. (2011). Malware Images: Visualization and Automatic Classification. In Proceedings of International Symposium on Visualization for Cyber Security.

52. Han, K. S., Lim, J. H., and Im, E. G. (2013). Malware analysis method using visualization of binary files. In Proceedings of Research in Adaptive and Convergent Systems ACM.

53. Makandar, A. and Patrot, A. (2015). Malware Analysis and Classification using Artificial Neural Network. In Trends in Automation Communications and Computing Technology.

54. Liu, L., Bao-sheng WANG, YU, B., and Qiu-xi ZHONG (2016). Automatic Malware Classification and New Malware Detection using Machine Learning. In Frontiers of Information Technology and Electronic Engineering.

55. VirusShare.com – Because sharing is caring. [Electronic resource] – Access: https://virusshare.com/

56. Beisel, B., Katz, M., Yehuda, K. (2013). Benign Software Corpus. [Electronic resource] – Access: https://api.semanticscholar.org/CorpusID:17243964

57. Singh, A. (2017). Malware Classification using Image Representation. [Electronic resource] – Access: https://security.cse.iitk.ac.in/node/254

58. Garnier, S. (2018). Implementation of the Matplolib 'viridis' color map in R. [Electronic resource] – Access: https://github.com/sjmgarnier/viridis

59. ResNet and ResNetV2. Keras. [Electronic Resource] – Access: https://keras.io/api/applications/resnet/

60. ImageNet. [Electronic Resource] – Access: http://www.image-net.org/

61. Pandalabs (2017). Quaterly Report. [Electronic resource] – Access: http://www.pandasecurity.com/mediacenter/src/uploads/2017/05/Pandalabs-2017-T1-EN.pdf