Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра кібербезпеки
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(освітній рівень)

на тему: *"Виявлення шкідливого програмного забезпечення в Android пристроях"*

Виконав: студент
Спеціальності:
125 «Кібербезпека»
(шифр і назва напряму підготовки, спеціальності)
Баранюк В. В.
підпис (прізвище та ініціали)

Керівник *Стадник М. А.*
підпис (прізвище та ініціали)

Нормоконтроль
підпис (прізвище та ініціали)

Завідувач кафедри *Загородна Н.В.*
підпис (прізвище та ініціали)

Рецензент
підпис (прізвище та ініціали)

м. Тернопіль – 2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра кібербезпеки
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Загородна Н.В.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 125 Кібербезпека
(шифр і назва спеціальності)

Студенту Баранюку Володимиру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Виявлення шкідливого програмного забезпечення в Android пристроях

Керівник роботи Стадник М. А., к.т.н.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 23 » березня 2022 року № 4/7-178

2. Термін подання студентом завершеної роботи 17.06.2022

3. Вихідні дані до роботи Промаркований датасет з ознаками шкідливого ПЗ на Android

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Огляд літературних джерел. 1.1 Шкідливе програмне забезпечення (ПЗ). 1.2 Сучасні процеси виконання коду 1.3 Типовий процес атаки зловмисного ПЗ. 1.4 Архітектура Android 1.5 Постановка задачі. 2 Теоретичні основи виявлення шкідливого ПЗ на Android пристроях. 2.1 Популярність Android. 2.2 Загрози Android. 2.2.1 Атаки шкідливого ПЗ на Android . 2.2.2 Помилки користувачів і розробників додатків. 2.3 Методи Виявлення шкідливого ПЗ на базі Android з використанням МН. 2.3.1 Статистичні методи аналізу. 2.3.2. Динамічні методи аналізу. 3. Практична частина. Виявлення шкідливого ПЗ в Android пристроях. 3.1 Етапи виявлення шкідливого ПЗ з набору даних. 3.2 Попередня обробка даних та формування набору ознак. 3.2.1 Попередня обробка набору дозволів для програмних додатків. 3.2.2 Попередня обробка набору сигнатур ПЗ. 3.3 Класифікація з використанням методів LR, KNN, RF. 3.3.1 Виявлення шкідливого ПЗ з використанням методів LR, KNN, RF на наборі даних дозволів програмних додатків. 3.3.2 Виявлення шкідливого ПЗ з використанням методів LR, KNN, RF на наборі сигнатур. 3.4 Порівняння результатів. 4 Безпека життєдіяльності, основи охорони праці. 4.1 Проведення інструктажів з охорони праці. . 4.2 Значення адаптації у трудовому процесі. Висновки. Список літературних джерел.

5. Перелік графічного матеріалу: 1. Титулка 2. Мета та задачі. 3. Актуальність дослідження. 4. Типові етапи атаки шкідливого ПЗ. 5. Класифікація методі МН для виявлення шкідливого ПЗ на Android пристроях. 6. Етапи виявлення шкідливого ПЗ методами МН 7. Попередня обробка набору даних дозволів 8. Попередня обробка набору даних сигнатур. 9. Результати класифікації методами RF, KNN, LR на основі набору даних дозволів. 10. Результати класифікації методами RF, KNN, LR на основі набору даних сигнатур. 11. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці			

7. Дата видачі завдання 23.03.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.03 – 28.03	*Виконано*
2.	Підбір джерел про проблеми шкідливого ПЗ на Android пристроях	29.03 – 05.04	*Виконано*
3.	Опрацювання джерел в галузі дослідження	06.04 – 11.04	*Виконано*
4.	Виконання дослідження щодо виявлення шкідливого програмного забезпечення на Android пристроях	12.04 – 18.04	*Виконано*
5.	Розроблення програмного коду	19.04 – 25.04	*Виконано*
6.	Оформлення розділу “Огляд літературних джерел”	26.04 – 29.04	*Виконано*
7.	Оформлення розділу “Теоретичні основи”	02.05 – 05.05	*Виконано*
8.	Оформлення розділу “Практична частина. Виявлення шкідливого програмного забезпечення в Android пристроях”	06.05 – 11.05	*Виконано*
9.	Виконання завдання до підрозділу “Безпека життєдіяльності, основи хорони праці”	12.05 – 16.05	*Виконано*
10.	Оформлення кваліфікаційної роботи	17.05 – 08.06	*Виконано*
11.	Нормоконтроль	09.06 – 13.06	*Виконано*
12.	Перевірка на плагіат	14.06 – 15.06	*Виконано*
13.	Попередній захист кваліфікаційної роботи	16.06 – 17.06	*Виконано*
14.	Захист кваліфікаційної роботи	23.06	

Студент _____ *Баранюк В. В.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Стадник М. А.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Виявлення шкідливого програмного забезпечення в Android пристроях // Кваліфікаційна робота ОР «Бакалавр» // Баранюк Володимир Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра кібербезпеки, група СБс-42 // Тернопіль, 2022 // С. – 64 , рис. – 9 , табл. – 3 , кресл. – 0, додат. – 2 .

Ключові слова: БЕЗПЕКА ANDROID, ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ, УРАЗЛИВІСТЬ КОДУ, МАШИННЕ НАВЧАННЯ.

Кваліфікаційна робота присвячена дослідженню проблеми шкідливого програмного забезпечення для мобільних пристроїв Android та його виявленню за допомогою машинного навчання. Представлено архітектуру Android, її вбудовану систему безпеки, проаналізовано недоліки та потенційні загрози, що відображають передумови для подальшого розуміння предметної області. В роботі здійснено систематичний огляд існуючих методів виявлення шкідливого програмного забезпечення, а саме: статичного, динамічного та гібридного.

Розроблено алгоритм для виявлення шкідливого програмного забезпечення для Android мобільних пристроїв з використанням існуючих бібліотек машинного навчання Python та промаркованих наборів вхідних даних дозволів та сигнатур з ознаками шкідливого програмного забезпечення.

Представлено дослідження якості роботи класифікаторів RF, LR, KNN в залежності від його набору параметрів та вхідного набору даних. Визначено важливі параметри в наборі даних, які значним чином впливають на якість класифікації, та оптимальні значення параметрів класифікаторів.

ANNOTATION

Identification of Malicious Software in Android Devices// Qualification thesis of educational level “Bachelor” // Baranyk Volodymyr Volodymyrovich // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Cybersecurity, СБс-42 group // Ternopil, 2022 // P. – 64, fig. – 9, table – 3, drawing – 0, apendix –2.

Key words: ANDROID SECURITY, MALWARE DETECTION, CODE VULNERABILITY, MACHINE LEARNING.

The qualification thesis is devoted to the investigation of Android device malware and its identification by machine learning. The Android architecture and its built-in security system are presented, vulnerabilities and potential threats are analyzed, which reflect the prerequisites for further understanding of the subject area. The paper presents systematically review the existing methods of detecting malicious software, namely: static, dynamic, and hybrid.

An algorithm has been developed to detect malicious software on Android mobile devices using existing Python machine learning libraries and a labeled input permission and signature dataset with signs of malware.

The dependence of the quality of classifiers RF, LR, KNN depending on its set of parameters and the input data set is presented in the qualification thesis. Important parameters in the data set that significantly affect the quality of classification and the optimal values of classifier parameters are identified.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API	Application Programming Interface
APK	Android Application Packages
ART	Android Runtime
DL	Deep Learning
DS	Data Science
JVM	Java Virtual Machine
KNN	K Nearest Neighbor
LR	Logistic Regression
NB	Naive Bayes
RF	Random Forest
МН	Машинне навчання
ОС	Операційна система
ПЗ	Програмне забезпечення

ВСТУП

24 лютого 2022 року почалась війна України із своїм східним сусідом, який незважаючи на усі домовленості, Будапештській меморандум, встановлені права людини, визнані світом границі кожної країни, цинічно напав на території української держави. Однією із особливостей цієї війни є окремий фронт, на якому українці тримають свої бойові позиції, а саме – цифровий. Сюди входить і протистояння спотвореним і неправдивим новинами, атаки у відповідь на DDOS державних сайтів та цифрових інфраструктур, захист баз даних державних реєстрів, захист звичайних переписок в месенджерах українців, інформування громадян України про першочерговий захист будь-якої інформації, створення додаткових програмних продуктів, такий як “Тривога”, чи нових функціональностей програм для отримання достовірної інформації.

Зважаючи на те, що практично кожен українець користується мобільним телефоном, який по суті є джерелом новин, способом зв’язку з рідними, міні-версією клієнта банку, то зростає потреба у захисті пристрою від несанкціонованого доступу. Станом на травень 2021 року частка Android пристроїв становила 72,2% [1], аналогічний розподіл користувачів і в Україні. Кількість програмних додатків для Android пристроїв в маркетплейсі Google Play становить понад 2,9 мільйони, звичайно це для усього світу. Така тенденція щодо зростання додатків породжується надзвичайно не складним завантаженням додатку без додаткових перевірок, це в порівнянні із App Store. Така популярність Android, легкість у публікації додатку у відкритому доступі та можливість отримання доступу до інформації українців робить пристрої Android надто привабливою мішенню для кіберзлочинців. Відповідно захист смартфону від шкідливих програм – це також одне із завдань кіберфоронту.

Хакери намагаються атакувати смартфони різними методами проте найбільш популярними визначено наступні: крадіжка облікових даних, спостереження та шкідлива реклама. Серед численних контрзаходів, методи на основі машинного навчання (MH, ML – machine learning) виявилися ефективними засобами виявлення цих атак, оскільки вони здатні отримати класифікатор з

набору навчальних прикладів, таким чином усуваючи необхідність у чіткому визначенні сигнатур при розробці детекторів шкідливих програм.

Метою кваліфікаційної роботи є автоматизація виявлення шкідливого програмного забезпечення на Android пристроях з використанням алгоритмів та методів машинного навчання.

Основне завдання було розділено на складові, отримані результати яких в сукупності дозволять досягнути встановленої мети. Перелік складових включає наступні:

- Проаналізувати архітектуру Android та можливі вразливості.
- Проаналізувати основі алгоритми роботи шкідливого ПЗ.
- Дослідити існуючі методи машинного навчання.
- Розробити алгоритм машинного навчання для ідентифікації шкідливого програмного забезпечення на Android.
- Отримати оптимальні параметри моделі класифікатора.
- Протестувати розроблений алгоритм на тестовому датасеті.

1 ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1.1 Шкідливе програмне забезпечення (ПЗ)

Шкідливе програмне забезпечення (ПЗ) – програмне забезпечення, що отримує несанкціонований доступ до конфіденційних даних, приватних мереж, комп'ютерних систем, та в загальному перешкоджає роботі електронного пристрою. В роботі під шкідливим ПЗ будемо розуміти зловмисне ПЗ, відкидаючи категорію дефективного ПЗ. Оскільки, останнє є легальним ПЗ з певними помилками та дефектами, що потребують додаткового тестування для виявлення причини та її усунення. До зловмисного ПЗ належать віруси, трояни, логери, шпигунське ПЗ, шкідливі додатки, комп'ютерні хробаки.

Перший та найбільш відомий комп'ютерний хробак Stuxnet використовує чотири вразливості операційної системи (ОС) Windows (одна з них є вразливістю нульового дня, zero-day). Успішність застосування Stuxnet демонструє зупинка ядерних центрифуг з повітряним зазором на іранському заводі по збагаченню урану в Натанзі в 2010 році. Хробак Stuxnet був одним із найбільш сенсаційних успіхів міжнародної кібервійни та демонстрацією, яка змінила гру, широкомасштабних руйнівних можливостей шкідливого комп'ютерного ПЗ. Ця частина шкідливого програмного забезпечення без розбору поширювалася по всьому світу серед користувачів ОС Windows. Stuxnet опинився на десятках тисяч машин працівників заводу у стані сплячого ПЗ, водночас призводячи до знищення однієї п'ятої ядерних центрифуг Ірану. Як наслідок, завдяки шкідливому ПЗ було перешкоджено державній програмі озброєння. Отож, комп'ютерний хробак Stuxnet отримав несанкціонований доступ до електронних пристроїв працівників з метою пошкодження інформаційних потоків між контролерами, що спричинило диверсію в автоматизованій системі керування заводу.

Аналіз шкідливого програмного забезпечення (ПЗ) – це дослідження функціональності, призначення, походження та можливого впливу шкідливого ПЗ. Зазвичай виявлення шкідливого ПЗ є задачею, що потребує значних затрат

часу та ручної роботи, саме тому для його виконання потрібні аналітики з експертними знаннями в області програмного забезпечення та реінжинірингу. Алгоритми та методи науки про дані (Data Science) та машинного навчання (МН, Machine Learning) можуть автоматизувати деякі етапи аналізу зловмисного ПЗ. Зазвичай основна частина таких алгоритмів покладається на вилучення значущих особливостей (feature extraction) із даних, що є нетривіальним завданням, яке потребує спеціалістів із глибокими знаннями в доменній області.

Під час аналізу шкідливих програм також потрібно враховувати змову додатків. Змова додатків (App collusion) – це два або більше додатків, які працюють разом для досягнення шкідливої мети [2]. Однак, якщо ці програми працюють окремо, немає ніякої ймовірності зловмисної діяльності. Це є обов'язковим для виявлення зловмисної комунікації між програмами та дозволів додатків для виявлення змови додатків.

Вихідний код складається із значної кількості коду і перш ніж запускатися як програмна реалізація на комп'ютері проходить ряд кроків. Розуміння цих кроків дуже важливо для будь-якого аналітика шкідливого ПЗ. Існує приблизно така ж кількість різних типів шкідливих програм, скільки різних типів програмного забезпечення. Кожен тип потенційно написаний іншою мовою програмування, орієнтований на різні середовища виконання та має різні вимоги до виконання. Маючи доступ до високорівневого коду (наприклад, C/C++, Java або Python), відносно легко зрозуміти, що робить програма і як передбачити її поведінку. Однак, швидше за все, аналітики не можуть з легкістю отримати доступ до високорівневого коду, який використовується для створення шкідливих програм. Більшість шкідливих програм створюються в середовищах із обмеженим доступом, продаються на підпільних форумах, їх знаходять на пристроях мимовільних жертв, приманках для зловмисників (honeypots). У своєму запакованому та розгорнутому стані більшість шкідливих програм існує як двійкові файли, які часто не читаються людиною і призначені для безпосереднього машинного виконання. Передбачення характеристик і поведінки

зловмисного ПЗ є процесом реінжинірингу, щоб з'ясувати дії та оцінити подальший вплив.

Бінарні файли є обфусковані, що створює великі труднощі для аналітиків кібербезпеки та для тих, хто намагається витягти з них інформацію. Без знання контексту інтерпретації, стандартів кодування та алгоритму декодування двійкові дані самі по собі не несуть ніякої корисної інформації та позбавлені сенсу. При цьому якість системи машинного навчання (МН) прямокорельвана із якістю вхідних даних, на яких вона навчається. Необроблені дані для алгоритмів МН вимагають плану збору, очищення та перевірки даних перед застосуванням в моделі МН. Попередня обробка цих необроблених даних важлива для вибору оптимального формату та представлення для введення в алгоритм навчання.

Щоб визначити та отримати діагностичні характеристики комп'ютерних двійкових файлів для виконання аналізу безпеки, потрібне глибоке розуміння внутрішніх компонентів ПЗ. Ця область дослідження називається реверс-інжинірингом ПЗ – процес вилучення інформації та знань про внутрішню роботу ПЗ для повного розуміння його властивостей, його роботи та недоліків. За допомогою реверс інжинірингу двійкового файлу можна зрозуміти його функціональність, його призначення, а іноді навіть його походження.

Шкідливе ПЗ може бути вбудовано в різноманітні двійкові формати, алгоритм роботи якого суттєво відрізняється один від одного. Наприклад, файли Windows PE (Portable Executables, з розширеннями файлів .exe, .dll, .ei тощо), файли Unix ELF (Executable and Linkable Format) і Android APK (формат Android Package Kit з розширеннями файлів .apk тощо) мають дуже різні файлові структури та контексти виконання. Відповідно базові знання, що є необхідними для аналізу кожного класу виконуваних файлів, також відрізняються. Зловмисне ПЗ на основі документів із розширеннями файлів, наприклад .doc, .pdf і .rtf, зазвичай використовує макроси та динамічні виконувані елементи в структурі документа для здійснення шкідливих дій. Шкідливе ПЗ також може поставлятися у формі розширень і плагінів для популярних програмних платформ, таких як веб-браузери та веб-фреймворки.

Класифікація шкідливого ПЗ об’єднує декілька екземплярів зловмисного ПЗ на основі загальних властивостей та в залежності від мети. Наприклад, група з питань безпеки може групувати зловмисне ПЗ за серйозністю та функціями, щоб ефективно сортувати ризик, який він становить для організації. Групи реагування з безпеки можуть групувати зловмисне ПЗ за потенційним масштабом пошкодження та вектором проникнення, щоб розробити стратегії усунення та пом’якшення. Дослідники шкідливого ПЗ можуть класифікувати зловмисне програмне забезпечення за походженням та авторством, щоб зрозуміти його генеалогію та призначення.

Зазвичай практикують групують зразки шкідливого ПЗ за сімейством – термін, який використовують аналітики зловмисного ПЗ, що дозволяє відстежувати авторство, співвідносити інформацію та ідентифікувати нові варіанти нещодавно знайденого шкідливого ПЗ. Зразки шкідливих програм одного сімейства можуть мати схожий код, можливості, авторство, функції, призначення.

Відомим прикладом сімейства шкідливих програм є Conicker, хробак, націлений на операційну систему Microsoft Windows. Незважаючи на те, що існує багато варіацій хробака Conficker, кожна з яких має різний код, авторів і поведінку, певні характеристики хробаків призводять до того, що їх відносять до одного і того ж шкідливого програмного забезпечення. Наприклад, усі хробаки Conficker експлуатують вразливості ОС Windows і беруть участь у атаках на словники, щоб зламати пароль облікового запису адміністратора, після чого встановлюють приховане ПЗ на експлуатованому хості для участі в діяльності бот-мережі.

Відмінності між зразками зловмисного ПЗ в межах одного сімейства є породжені різними компіляторами, які використовуються для компіляції вихідного коду, або від розділів коду, доданих чи видалених, щоб змінити функціональність зловмисного ПЗ. Зразки зловмисного ПЗ, які розвиваються з часом у відповідь на зміну стратегій виявлення або пом’якшення, часто також

демонструють схожість між старішою та новішою версіями, що дозволяє аналітикам простежити еволюцію сімейства шкідливих програм.

В узагальненій класифікації шкідливих програм є клас нешкідливих двійкових файлів. Цей тип класифікації використовується, щоб визначити, чи є частина програмного забезпечення шкідливою. Враховуючи довільний двійковий файл, можна взнати ймовірність того, чи може користувач довіряти йому та виконувати його в надійному середовищі. По суті визначення цієї ймовірності є основною метою антивірусного ПЗ. Традиційно це завдання обумовлено відповідністю сигнатур: з огляду на безліч властивостей і поведінки раніше поміченого зловмисного ПЗ, нові вхідні двійкові файли можна порівняти з цим набором даних, щоб визначити, чи відповідає він тому, що було додано в базу.

Метод відповідності сигнатур працює добре, якщо автори шкідливого ПЗ не істотно змінюють властивості та поведінку нового зловмисного ПЗ, щоб уникнути виявлення, а вибрані властивості та поведінка свідчать про подібність до певного сімейства, що характеризується аналогічними властивостями та поведінкою.

Метаморфічні або поліморфні віруси та хробаки використовують статичні та динамічні методи обфускації, щоб змінити характеристики свого коду, поведінки та властивостей, які використовуються в алгоритмах генерації сигнатур механізмів ідентифікації шкідливих програм. Цей рівень створення зловмисного ПЗ раніше був рідкісним, але став більш поширеним завдяки його постійним успіхам у запобіганні синтаксичним сигнатурним механізмам виявлення шкідливого ПЗ.

1.2 Сучасні процеси виконання коду

Для аналізу шкідливого ПЗ представимо процеси створення і виконання загальних класів сучасних програм, і представимо, як можна перевірити двійкові файли та програми, що виконуються, щоб зрозуміти їх внутрішню роботу без будь-якого доступу до написаного коду.

Існує два типи виконання коду: компільоване та інтерпретоване. У компільованому виконанні написаний код перекладається в рідні машинні

інструкції за допомогою ряду кроків перетворення (часто згадується як збірка програмного забезпечення процес). Ці машинні інструкції упаковуються в двійкові файли, які потім можуть виконуватися безпосередньо апаратними засобами.

У реалізаціях інтерпретованого виконання написаний код (іноді його називають сценарієм) перекладається в проміжний формат, який потім подається в інтерпретатор для виконання програми. Інтерпретатор відповідає за виконання інструкцій програми на обладнанні. Проміжний формат варіюється в різних реалізаціях, але найчастіше є формою байт-коду (бінарних машинних інструкцій), який буде виконуватися на віртуальній машині.

Деякі реалізації можуть бути гібридом скомпільованого та інтерпретованого виконання. Таке гібридне виконання використовують процес, званий компіляцією “точно вчасно” (jyst in time, JIT), у якому байт-код інтерпретатора компілюється у власні машинні інструкції в режимі реального часу.

На рис. 1.1 зображено загальні процеси виконання коду для деяких сучасних програмних реалізацій.

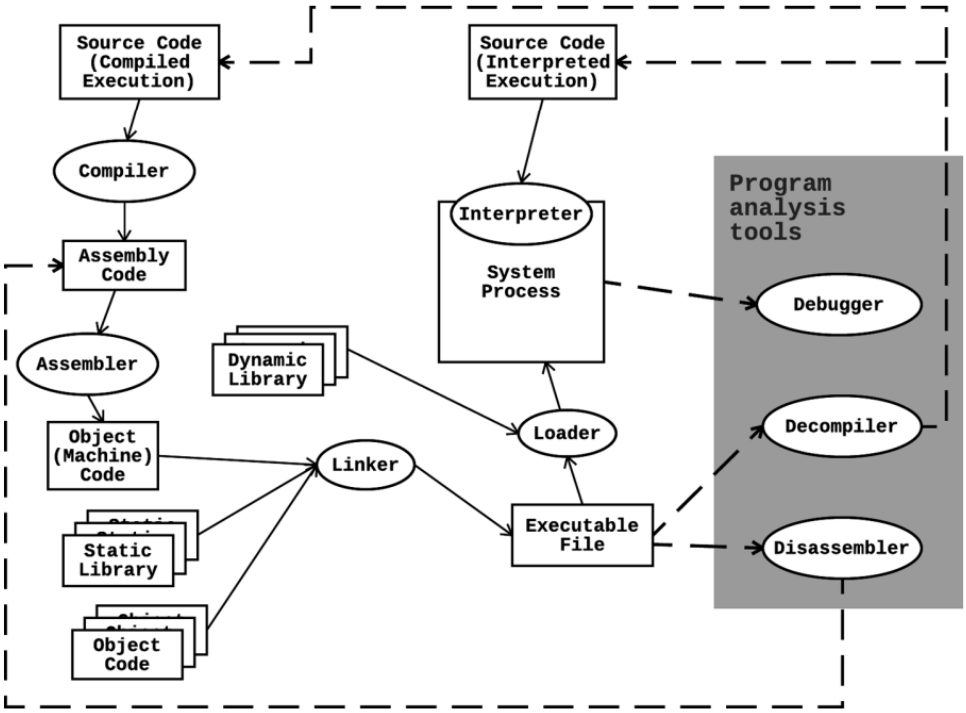


Рисунок 1.1 – Блок-схема виконання коду та аналізу програми [3]

Прямокутні поля представляють програму в різних станах її існування. Еліпси відображають кроки перетворення ПЗ, з якого інтерпретується програма з одного стан в інший. Суцільні стрілки між вузлами представляють прогрес коду від його написаного людиною стану до його кінцевого виконання на апаратному забезпеченні. Сірий прямокутник містить деякі інструменти, які можуть використовувати реверс-інженери для перевірки статичного або динамічного стану двійкового або запущеного ПЗ програми.

Процес виконання коду для інтерпретованих мов коротший, оскільки немає обов'язкового етапу збирання чи компіляції: код запускають одразу після його написання. Проте не можливо запустити байт-код Python безпосередньо на цільовому обладнанні, оскільки він потребує інтерпретації віртуальною машиною Python і подальший переклад її у машинний код. Цей процес призводить до певної неефективності та втрат продуктивності в порівнянні з мовами "нижнього рівня", такими як C.

Маючи доступ до написаного людиною вихідного коду, можна легко аналізувати конкретні властивості та наміри частини ПЗ, що дозволяє нам точно класифікувати його за сімейством та функціями. Код проходить чітко визначений шлях на своєму шляху від створення до виконання. Перехоплення в будь-якій точці шляху дозволить виявити велику кількість інформації про програму.

1.3 Типовий процес атаки зловмисного ПЗ

Щоб вивчити та класифікувати зловмисне ПЗ, надзвичайно важливо розуміти, що робить шкідливе ПЗ та як відбувається зловживання. Різні типи зловмисного програмного забезпечення мають різні методи поширення, служать різним цілям і становлять різний рівень ризику для окремих осіб і організацій. Типовий процес атаки зловмисного програмного забезпечення представлено на рис 1.2.

На початковому етапі (етап 1) початкові зусилля є пасивними, використовуючи непрямі методи для визначення цілі. Після цього проводяться

активні розвідувальні заходи, наприклад, сканування портів, щоб зібрати більш конкретну та актуальну інформацію про ціль, виявляючи вразливість для проникнення. Вразливістю може бути відкритий порт, на якому запущено невиправлене уразливе ПЗ, або співробітник, схильний до фішингових атак. Використання вразливості призводить до того, що шкідливе ПЗ успішно проникне до периметру.

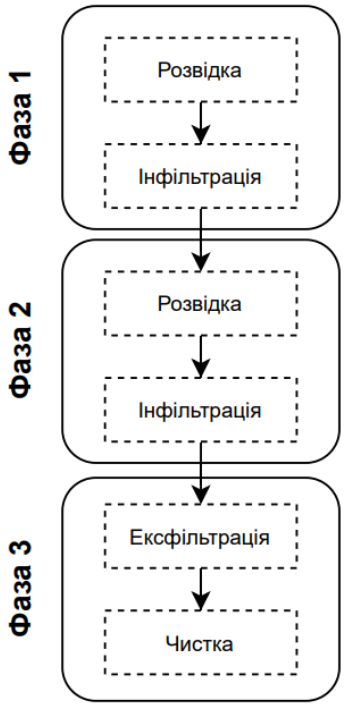


Рисунок 1.2 – Типовий процес атаки шкідливого ПЗ

На другому етапі зловмисне ПЗ вже знаходиться в середовищі жертви. Завдяки процесу внутрішньої розвідки та повороту хоста зловмисне ПЗ може маневрувати через мережу, щоб знайти високоцінні хости. Згодом воно закріплюється в середовищі, використовуючи такі засоби, як встановлення бекдорів для майбутнього доступу або встановлення себе як постійного фонового процесу-демона.

На третьому етапі зловмисне ПЗ може видалити себе з середовища і не залишити слідів. Для зловмисного ПЗ, яке здійснює будь-який вид крадіжки приватної інформації, етап ексфільтрації надсилає вкрадені дані (наприклад, облікові дані користувача, номери кредитних карток та важливу бізнес-логіку) на

віддалений сервер. Коли завдання буде виконано, зловмисне ПЗ самоочищується та видаляє всі сліди своїх дій з машини-жертви.

Шкідливе ПЗ демонструє певні типи поведінки, що специфічні саме для нього:

- *Приховання присутності.* Шкідливе ПЗ використовує пакувальники та методи шифрування для стиснення та маскування свого коду. Метою цього є уникнення виявлення та перешкоджання аналізу.
- *Виконання своєї функції.* Щоб ефективно виконувати свої функції, зловмисне ПЗ має забезпечити певний ступінь стійкості, щоб воно не було стерто системними змінами чи не виявлено адміністраторами. Зазвичай використовуються методи ухилення від захисту, такі як бічне завантаження DLL та знищення антивірусних процесів.
- *Збір даних і телефонний дзвінок "додому".* Після того, як зловмисне ПЗ отримає всі необхідні йому дані (облікові дані сервера/програми, журнали веб-доступу, записи бази даних), то надсилає дані до зовнішнього пункту збору. Також може "зателефонувати додому" на сервер віддаленого командування та керування і отримати подальші інструкції.

1.4 Архітектура Android

У цьому пункті представлений поверхневий огляд архітектури Android та його вбудованих засобів безпеки, що є надзвичайно важливо для розуміння предметної області.

Android побудовано на основі ядра Linux. Linux було обрано враховуючи те, що він має відкритий вихідний код, перевіряє докази шляху, надає драйвери та механізми для мереж, а також керує віртуальною пам'яттю, живленням пристрою та безпекою. Android має багатошарову архітектуру [4], що зображено на рис. 1.3

Шари розташовуються знизу вгору. Поверх рівня ядра Linux рівень апаратної абстракції (Hardware Abstraction Layer), власні бібліотеки C/C++ і середовище виконання Android (Android Runtime), фреймворк програмного

інтерфейсу програм Java (API) (Java Application Programming Interface (API) Framework) і системні програми розташовані однин над одним. Кожен шар відповідає за конкретне завдання. Наприклад, Java API Framework надає бібліотеки Java для виконання дій, пов'язаних із застосуванням визначення місцезнаходження, наприклад визначення широти та довготи. Розглянемо більш детальніше кожен з них [5]:

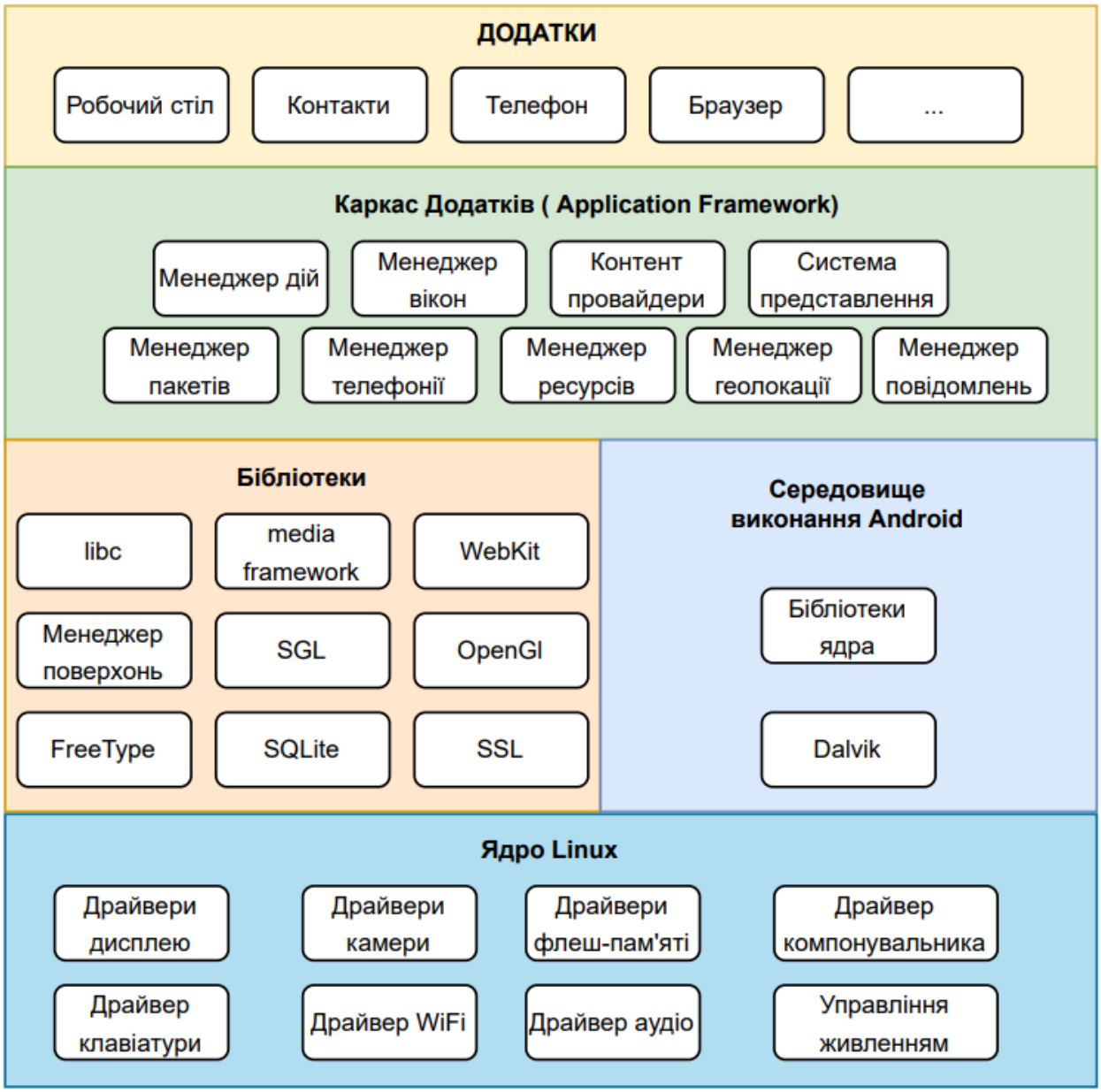


Рисунок 1.3 – Багатошарова архітектура ОС Android

Рівень додатків. До складу ОС Android входить набір базових додатків, а саме: клієнти електронної пошти та SMS, календар, різні карти, браузер, програма

для керування контактами та багато іншого. Усі програми повинні бути написані мовою Java (Kotlin).

- система представлень (View System) – це значний набір представлень з функціональністю, що розширюється. Набір потрібен для побудови UX додатків, що включає такі компоненти, як списки, таблиці, поля введення, кнопки;
- менеджер ресурсів (Resource Manager) призначений для доступу до рядкових, графічних та інших типів ресурсів;
- менеджер сповіщень (Notification Manager) дозволяє відображати повідомлення користувача в рядку статусу.

Рівень бібліотек включає набір C/C++ бібліотек, які використовуються різними компонентами ОС Android. Для Android-програмістів доступ до функцій цих бібліотек реалізований за допомогою Application Framework. Деякі представлені нижче:

- System C library – BSD-реалізація стандартної системної бібліотеки C (libc) для пристроїв, що вбудовуються, заснованих на Linux.
- Media Libraries – бібліотеки на основі Packet Video's Open CORE, призначені для роботи із аудіо- та відео-форматами (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG); LibWebCore - ядро вбудованого web-браузера.
- Surface Manager – менеджер поверхонь управляє доступом до підсистеми відображення 2D- та 3D-графічних шарів.
- SGL (Scalable Graphics Library) – бібліотека для роботи з 2D-графікою, заснована на бібліотеці SDL (Simple Direct Media Layer).
- 3D libraries – бібліотеки для роботи з 3D-графікою, засновані на OpenGLES 1.0 API.
- FreeType – бібліотека, призначена для роботи зі шрифтами.
- SQLite – реляційна СУБД.

Рівень середовища виконання. Теперішні програми Android написані мовою, подібною до Java, а саме: Kotlin. Проте є чіткі відмінності між Java API та Android API. У типових параметрах виконання Java вихідний код Java компілюється в байт-код Java, який виконується Java віртуальною машиною (JVM). У попередніх

версіях Android (до Android 4.4 KitKat) скомпільований байт-код зберігався у файлах .dex (Dalvik Executable) і виконувався віртуальною машиною Dalvik. Dalvik має архітектуру на основі реєстрів, тоді як JVM має архітектуру на основі стека. Оскільки Dalvik розроблено для роботи в середовищі з обмеженими ресурсами, як-от мобільні пристрої та вбудовані системи, він також призначений для використання меншого місця та включає багато спрощень для підвищення ефективності. У нових версіях Android Runtime (ART) замінив Dalvik як новий стандарт для виконання програми для Android. ART використовує той самий байт-код .dex, але має ще більше оптимізації продуктивності (наприклад, завчасну компіляцію під час встановлення), щоб покращити швидкість і споживання ресурсів програм.

Рівень ядра Linux. Ядро служить шаром абстракції між апаратним та програмним забезпеченням, тобто керуванням пам'яттю, мережею, безпекою, процесами та живленням. Оскільки програми на базі Android працюють на пристроях із батарейним живленням та обмеженою пам'яттю, тому операційна система Android розроблена таким чином, що будь-яким ресурсом можна безперешкодно керувати [6]. Наприклад, ОС Android автоматично призупинить програму в пам'яті, якщо програма на даний момент не використовується. Цей стан відомий як стан виконання життєвого циклу програми. Роблячи це, ОС може зберегти потужність, яка може бути використана під час повторного запуску програми. В іншому випадку програми залишаються бездіяльними, доки їх не закриють [7].

Як окрему компоненту ОС Android розглянемо ***вбудовану систему безпеки***. Android поставляється з уже вбудованим захистом. Це привілейована відокремлена операційна система [8]. Техніка пісочниці (Sandbox) та система дозволів в Android зменшують деякі ризики та помилки в програмі. Техніка пісочниці в Android ізолює запущені програми за допомогою унікальних ідентифікаторів, які базуються на середовищі Linux. Без дозволів, наданих користувачем під час встановлення або переналаштування програми, програми не можуть отримати доступ до системних ресурсів. Якщо деякі з дозволів не надано,

то сама програма не буде придатною до використання. Коли відбувається оновлення системи, відбувається ряд покращень з точки зору безпеки та конфіденційності. Наприклад, Android 11 та остання стабільна версія Android, містить деякі зміни, пов'язані з безпекою та конфіденційністю, такі як примусове використання обсягу пам'яті, одноразові дозволи, автоматичне скидання дозволів, доступ до фонового розташування, видимість пакету та сервіси переднього плану. Однак існують можливості атак з застосуванням зловмисного ПЗ на основі деяких вразливостей у програмах, розроблених різними користувачами, оскільки Google Play Store не виявить деякі вразливості під час публікації додатків у Play Store, як у Apple App Store.

1.5 Постановка задачі

Користувачі та розробники додатків, програмісти ОС Android роблять помилки, які створюють сприятливі передумови для зараженням шкідливим ПЗ. Звичайно кожен користувач несе відповідальність за те, що він встановлює на свій персональний смартфон, які доступи надає для програмного додатку (доступ до списку контактів, камери, геолокації). Інформаційна компанія серед користувачів надзвичайно потрібна щодо захисту свого пристрою, проте вразливості спричинені програмним кодом для звичайного користувача не є явними, і оцінити ризик отримання зловмисного ПЗ є практично неможливим. Відповідно автоматизація процесу виявлення шкідливого ПЗ є актуальною задачею. Звичайно можна мати список додатків, які визначені як заборонені та як шкідливе ПЗ і не встановлювати його. Можна порівнювати сигнатури мобільних додатків із сигнатурами сімейств шкідливих додатків, проте такий підхід є надто виснажливий для людини, оскільки потребує персональної участі та затрат часу. Також частини прикладів сигнатур наявного шкідливого ПЗ практично не існує, тому значна кількість академічних та промислових досліджень були виконані з використанням алгоритмів МН на базі Android пристроїв. Такий напрям виявлення шкідливого ПЗ дозволяє на основі попередніх статистичних даних про

шкідливі та не шкідливі програмні додатки чи їх частини побудувати класифікатор за обраними діагностичними ознаками.

Основним завданням для досягнення мети кваліфікаційної роботи, визначеної у вступі, є побудова класифікатора з використанням математичного апарату МН. По суті, це є обрання моделі МН та оптимізація її параметрів, що вимагає вирішення наступних задач:

- Огляд існуючих моделей МН для виявлення шкідливого ПЗ на базі Android.
- Дослідження вхідного набору даних та виявлення діагностичних ознак.
- Розробка моделі МН для виявлення зловмисного ПЗ.
- Знаходження оптимальних параметрів моделі МН.
- Оцінка точності класифікації.

2 ТЕОРЕТИЧНІ ОСНОВИ ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ НА ANDROID ПРИСТРОЯХ

2.1 Популярність Android

У цю технологічну епоху використання смартфонів та пов'язаних із ними програм швидко зростає завдяки зручності та ефективності різноманітних додатків та постійному вдосконаленню апаратного та програмного забезпечення на розумних пристроях. Очікується, що до 2023 року кількість користувачів смартфонів досягне 4,3 мільярда [1]. Android – найпоширеніша мобільна операційна система (ОС). Як вказано у вступі, що станом на травень 2021 року 72,2% усього ринку смартфонів належить Android [9]. Друга за величиною частка ринку - 26,99% - належить Apple iOS, а решта 0,81% - будуть розділені Samsung, KaiOS та іншими меншими постачальниками. Google Play — офіційний магазин додатків для пристроїв Android. Станом на травень 2021 року на ньому було опубліковано понад 2,9 мільйона додатків. З них понад 2,5 мільйона додатків класифікуються як звичайні програми, а 400 000 додатків класифікуються як програми AppBrain низької якості. Світова популярність Android робить його більш привабливою мішенню для кіберзлочинців і більше піддається ризику зловмисного програмного забезпечення та вірусів.

2.2 Загрози Android

Незважаючи на те, що Android має хороші вбудовані заходи безпеки, є кілька недоліків архітектури та безпеки, які стали загрозою для його користувачів. Обізнаність про ці загрози також важлива для правильного виявлення шкідливих програм і аналізу вразливостей. Було опубліковано багато досліджень і технічних звітів, пов'язаних із загрозами Android та класифікованими загрозами Android на основі методології атаки. Атаки соціальної інженерії, атаки отримання фізичного доступу та мережеві атаки описані у способах отримання доступу до пристрою. Для вразливостей і методів експлуатації враховуються атаки людини посередині

(MITM), атаки повернення до libc, атаки JIT-Spraying, вразливості сторонніх бібліотек, уразливості Dalvik, вразливості архітектури мережі, вразливості віртуалізації, а також мости налагодження Android і вразливості ядра.

В опитуванні [10] було визначено чотири типи атак на Android:

- апаратні атаки;
- атаки на основі ядра;
- атаки на рівні апаратної абстракції (HAL);
- атаки на основі програм.

Атаки на основі апаратного забезпечення, такі як Drammer, Rowhammer, Glitch є пов'язаними з датчиками, сенсорними екранами, засобами зв'язку та DRAM.

Атаки на основі ядра, такі як Gooligan, DroidKungfu, Returnoriented Programming, пов'язані з привілеями root, пам'яттю, завантажувачем і драйвером пристрою.

Атаки на основі HAL, такі як Return to User і TocTou, пов'язані з інтерфейсами для камер, Bluetooth, Wi-Fi, глобальною системою позиціонування (GPS) і радіо. Атаки на основі програм, такі як AdDetect, WuKong і LibSift, пов'язані зі сторонніми бібліотеками, внутрішньо бібліотечною змовою та підвищенням привілеїв.

У додатки Android легко проникнути з належним знанням програмування Android, якщо відповідні механізми безпеки відсутні. Крім того, ринки Android, такі як Google Play, не дотримуються розширених протоколів безпеки, коли публікуються нові програми. Наприклад, гру для Android, відому як Angry Birds, зламали, і хакеру вдалося проникнути в її файл APK і вставити шкідливий код, який надсилав текстові повідомлення без відома користувача.

2.2.1 Атаки шкідливого ПЗ на Android

Атаки зловмисного ПЗ є найпоширенішим випадком, який можна визначити як загрозу для Android. Багато дослідників дають різні визначення шкідливого програмного забезпечення залежно від шкоди, яку вони завдають. Кінцевим

значенням зловмисного програмного забезпечення є будь-яка шкідлива програма з фрагментом шкідливого коду, який має негативний намір отримати несанкціонований доступ і не виконувати ні юридичні, ні етичні дії, порушуючи три основні принципи безпеки: конфіденційність, цілісність і доступність.

Шкідливе програмне забезпечення, пов'язане з розумними пристроями, можна класифікувати за трьома ознаками: цілі та поведінка атаки, шляхи розповсюдження та зараження, а також режими отримання привілеїв. Шахрайство, спам-повідомлення, крадіжка даних і зловживання ресурсами – все це можна назвати як цілі атаки та поведінкові аспекти. Ринки програмного забезпечення, браузери, мережі та пристрої можна визначити як шляхи розповсюдження та зараження. Технічна експлуатація та маніпулювання користувачами, такі як соціальна інженерія, можуть бути перераховані в режимах привілеїв і отримання. Шкідливе ПЗ, пов'язане з операційною системою Android, ідентифікується як шкідливе ПЗ Android (Android malware) [11], яке завдає шкоди або краде дані з мобільного пристрою на базі Android. Вони класифікуються як трояни, шпигунські програми, рекламне програмне забезпечення, програмне забезпечення-вимагач, хробаки, ботнети та бекдори. Google описує шкідливі програми як потенційно небезпечні програми. Вони класифікували зловмисне програмне забезпечення як комерційне та некомерційне шпигунське програмне забезпечення, бекдори, ескалацію привілеїв, фішинг, типи шахрайства, такі як шахрайство з кліками, шахрайство з платіжками, шахрайство із службою коротких повідомлень (SMS) та троянські програми.

2.2.2 Помилки користувачів і розробників додатків

Помилки можуть статися свідомо чи несвідомо як з боку розробників, так і з боку користувачів. Ці помилки можуть призвести до загроз для ОС Android та її програм.

Встановлено, що користувачі несуть відповідальність за більшість проблем безпеки. Деякі поширені помилки користувачів призводять до серйозних загроз у додатку Android. Під час встановлення програм Android користувачам

запропоновано надати деякі дозволи. Однак не усі користувачі можуть розуміти мету кожного дозволу. Вони дають дозвіл на запуск програми, не враховуючи її серйозність. Шахрайські програми можуть викрасти дані та виконувати ненавмисні завдання після отримання необхідних дозволів. Можливе виникнення загроз для систем Android через помилки, допущені розробниками додатків під час самого процесу розробки. На етапі публікації Android програм, Google Play матиме лише обмежений контроль над уразливими місцями коду в програмах. Іноді розробники помилковоі вказують небажані дозволи у файлі маніфесту Android, що спонукає користувача надавати дозволи, якщо дозволи були класифіковані як непрості. Незважаючи на те, що компанії, що розробляють додатки, і деякі магазини додатків радять дотримуватися вказівок безпеки, запроваджених під час розробки, багато розробників все ще не можуть писати безпечні коди для створення захищених мобільних додатків.

2.3 Методи виявлення шкідливого ПЗ на базі Android з використанням МН

Досліджуючи внутрішню структуру та роботу програм Android, необхідно застосувати методи реверс-інженерії, щоб знайти функції, які можуть допомогти ідентифікувати та класифікувати шкідливе ПЗ. Подібні дії вручну можуть допомогти нам створити розширені функції для кількох програм Android, але цей метод погано масштабується, коли потрібно застосувати ті самі виявлення ознак до більших наборів даних.

Загальна методологія ідентифікації ознак має бути якомога ретельнішою при розгляді корисних представлень даних. Коли кожен зразок складається лише з кількох булевих ознак, складної ідентифікації ознак не потрібно – достатньо буде просто використовувати вихідні дані як вхідні дані для алгоритмів класифікації. Однак, коли кожен зразок володіє значною інформацією і є складним, як програмне забезпечення чи виконувані двійкові файли, робота аналітика значно ускладняється. Не надто великий двійковий файл розміром 1 МБ містить 223 біти інформації, яка перетворюється в геометричній прогресії на 8 388 608 різних

можливих значень. Спроба виконати завдання класифікації з використанням інформації бітового рівня може швидко стати нерозв'язною, і це не є ефективно, оскільки дані містять багато зайвої інформації, яка не є корисною для процесу МН.

Виявлення шкідливих програм в Android можна виконати двома способами:

- методами виявлення на основі сигнатур;
- методами виявлення на основі поведінки [12].

Метод виявлення на основі сигнатур простий, ефективний і дає низький рівень помилкових спрацьовувань. Двійковий код програми порівнюється з сигнатурами за допомогою відомої бази даних шкідливих програм. Однак виявити невідоме шкідливе програмне забезпечення за допомогою цього методу неможливо. Таким чином, метод виявлення на основі поведінки/аномалій є найбільш поширеним способом. Цей метод зазвичай запозичує методи МН та DS. Багато досліджень було проведено для виявлення шкідливого програмного забезпечення Android за допомогою традиційних методів на основі МН, таких як дерева рішень (Decision Trees – DT) і методу опорних векторів (Support Vector Machines – SVM), а також нові моделі на основі DL, такі як Deep Convolutional Neural Network (Deep CNN). Ці дослідження показали, що ML можна ефективно використовувати для виявлення шкідливих програм в Android.

Виявлення зловмисного програмного забезпечення за допомогою МН включає дві основні фази: аналіз пакетів програм Android (Android Application Packages – APK), щоб отримати відповідний набір функцій, а потім використання МН і методів глибокого навчання (deep learning – DL) для отриманих функцій для розпізнавання шкідливих APK. Подібно до виявлення зловмисного програмного забезпечення, виявлення вразливостей у програмному коді включає дві основні фази, а саме створення ознак за допомогою аналізу коду та МН на отриманих ознаках для виявлення вразливих сегментів коду.

В загальному фактори, що є вразливостями Android, та відповідні методи виявлення шкідливого ПЗ представлено на рис. 2.1. Користувачі та розробники Android роблять помилки, які наражають їх на непотрібні небезпеки та ризики

зараження своїх пристроїв шкідливим програмним забезпеченням, тому їх виділено як окремі групи на рисунку.

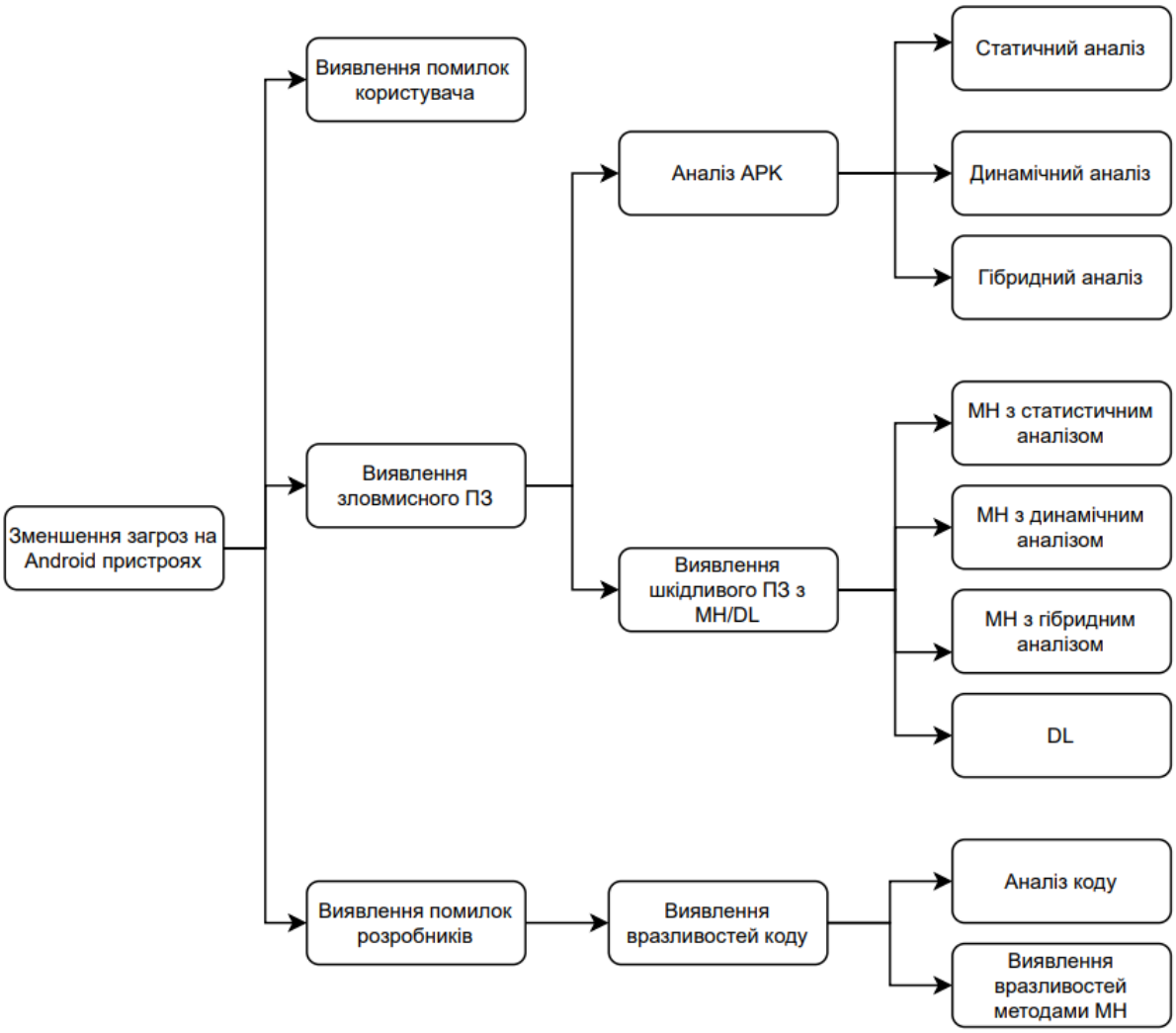


Рисунок 2.1 – Фактори зменшення загроз для Android та відповідні методи аналізу шкідливого ПЗ [13]

Для аналізу APK використовують три групи методів:

- Статистичний: входить структурний аналіз та статичний.
- Динамічний: входить аналіз поведінки, налагодження (debugging) та динамічні інструменти аналізу.
- Гібридний включає в себе комплекс методів аналізу з статистичного та динамічного методів одночасно.

Статичний аналіз можна виконати шляхом аналізу байт-коду та вихідного коду (або переробленого APK) замість того, щоб запускати його на мобільному

пристрої. Динамічний аналіз виявляє зловмисне програмне забезпечення, аналізуючи програму під час її запуску в симуляторі або реальному середовищі. Однак існує велика ймовірність певної міри піддати ризику середовище виконання в динамічному аналізі, оскільки будуть виконуватися шкідливі коди, які можуть зашкодити цьому середовищу. Гібридний аналіз включає методи як статичного, так і динамічного аналізу

Розглянемо більш детальніше деякі методи з кожної групи.

2.3.1 Статистичні методи аналізу

Статичний аналіз є широко використовуваним механізмом для виявлення шкідливих програм Android. Це тому, що їх не потрібно встановлювати на пристрої, оскільки цей підхід не використовує середовище виконання.

Запропоновано 4 аспекти в межах статистичного аналізу [14]: методи аналізу, аналіз чутливості, структуру даних та представлення коду. Методів аналізу дозволяє визначити символічне виконання, здійснити аналіз “плям”, розділення програми, абстрактну інтерпретацію, перевірку типу та інструментування коду. Для аналізу чутливості були визначені об’єкт (object), контекст (context), поле (field), шлях (path) і потік(flow). Для аспекту структури даних можна перерахувати графік викликів (call graph – CG), графік потоку керування (Control Flow Graph – CFG) та міжпроцедурний графік потоку керування (Inter Procedural Control Flow Graph – ICFG). Smali, Jimple, Wala-IR, Dex-Assembler, Java Byte код або клас були перераховані під аспектом представлення коду. Ядро, програму та емулятор можна взяти під аспект рівня перевірки.

Методи виділення ознак в статичному аналізі поділяються на дві групи: аналіз маніфесту та аналіз коду. Назва пакета, дозволи, наміри, дії, сервіси та постачальники визначаються в аналізі маніфесту. Під час аналізу коду такі дані, як виклики API, потік інформації, відстеження “плям”, коди операцій, первинний

код і аналіз відкритого тексту, можуть бути ідентифіковані як можливі ознаки для подальшої класифікації моделлю МН.

Статичний аналіз на основі маніфесту є надзвичайно вживаною технікою статичного аналізу. Екземпляр методи на основі маніфесту запропонований в SigPID [15] описує механізм виявлення шкідливого ПЗ Android на основі дозволів. В межах цієї моделі визначено лише 22 дозволи з усіх дозволів, перерахованих у зразках файлів .apk, які є значущими завдяки розробці трирівневого методу “пуррінгу” даних (data purring): рейтинг дозволів із негативною ставкою, рейтинг дозволів на основі підтримки та отримання дозволів із правилами асоціації. Після цього для виявлення зловмисного ПЗ були використані алгоритми МН. Для цього процесу був використаний набір даних дозволів у двійковому форматі, який був створений за допомогою бази даних шкідливих програм і безпечних програм із Google Play. SVM показала найкращі результати при класифікації (у порівнянні із NB та DT) з точністю понад 90%.

Дозволи та наміри Android використовувалися як основні статичні ознаки класифікації шкідливих програм, а URL-адреси, електронні адреси та IP-адреси використовувалися як основні динамічні ознаки [16]. Спочатку файли APK були декомпільовані за допомогою ApkTool. Цей модуль витягував різні типи інформації, пов’язаної зі зловмисним програмним забезпеченням. Після вилучення даних шляхом розбирання файлів .dex дані зберігалися в текстових файлах і використовувалися для створення вектора ознак. Потім алгоритми RF, NB, Gradient Boosting (GB) та Ada Boosting (AB) були використані для навчання та тестування моделі виявлення шкідливих програм із використанням набору даних Drebin та Google Play Store. Після виконання частини навчання та тестування МН для кожної з функцій дозволу, наміру та мережі окремо було виявлено, що вищезгадані алгоритми МН працювали з різною точністю. Для дозволів RF показав хороші результати з 0,98 точністю, для намірів NB показав хороші результати з точністю 0,92.

Для динамічного аналізу було визначено п’ять методів виділення ознак. До них відносять: (1) аналіз мережевого трафіку для таких функцій, як уніфіковані

локатори ресурсів (URL), Інтернет-протокол (IP), мережеві протоколи, сертифікати та незашифровані дані, (2) інструментарій коду для таких функцій, як класи Java, наміри та мережевий трафік, (3) Аналіз системних викликів, (4) Аналіз системних ресурсів для таких функцій, як використання процесора, пам'яті та акумулятора, звіти про процес, використання мережі та (5) Аналіз взаємодії користувачів для таких функцій, як кнопки, значки та дії/події . У дослідженні [17] вивчається безпека ML для методів виявлення шкідливого програмного забезпечення Android з використанням класифікатора на основі навчання з викликами API, витягнутими з конвертованих smali-файлів. Потім пропонується складний безпечний метод навчання, який показав, що можна підвищити безпеку системи від широкого спектру атак ухилення. Ця модель також застосовна до областей виявлення спаму та шахрайства. Це дослідження можна додатково покращити, досліджуючи можливості виявлення атак, які можуть змінити навчальний процес.

Техніка виявлення шкідливого програмного забезпечення Android з використанням зважування функцій з об'єднаною оптимізацією відображення ваги та моделі параметрів класифікатора запропонована в JOWMDroid Framework [18]. Ця модель вибирає певну кількість ознак із вилучених із програми ознак, пов'язаних із виявленням зловмисного ПЗ. Цей процес було здійснено шляхом декомпіляції файлу .apk у файли manifest і class.dex та підготовлено двійкову матрицю функцій. Початкова вага була розрахована за допомогою моделей Random Forest, SVM, LR та KNN. Функції вагової машини були розроблені для того, щоб співвіднести початкову вагу з кінцевою вагою. На останньому етапі класифікатори та параметри ваг були спільно оптимізовані за допомогою диференціального еволюційного алгоритму (Differential Evolutional algorithm). Для навчання моделі були використані набори даних Drebin, Google Play і APKPure. В результаті серед класифікаторів, які не знають вагу, RF працював

краще з точністю 95,25%, а для класифікаторів, що враховують вагу, KNN працював найкраще.

2.3.2 Динамічні методи аналізу

Використовуючи динамічний підхід, можна виявити шкідливе ПЗ за допомогою МН після запуску програми в середовищі виконання. В роботі [19] представлено виявлення зловмисного ПЗ Android за допомогою мережевого підходу. Спеціальна програма виявлення була розроблена для такого підходу. Вона містила три модулі: збір мережевих слідів, знаходження мережевих функцій та ідентифікацію. У модулі збору слідів відстежувалися мережеві дії запущених додатків і реєструвалися мережеві сліди періодично. Модуль знаходження ознак витягує характеристики мережі, що використовуються програмами. Цими ознаками були характеристики на основі системи доменних імен (DNS), на основі протоколу передачі гіпертексту (HTTP), ознаки, засновані на пункті призначення походження та характеристики на основі протоколу керування передачею (TCP). У модулі виявлення використовувалися алгоритми DT, LR, KNN, мережі Байєса та RF. Найвищу точність серед них забезпечив RF-алгоритм

У [20] була запропоновано певний алгоритм Service Monitor, яка є не надто складною системою виявлення на базі хоста, яка може виявляти шкідливе ПЗ на пристроях. Service Monitor відстежує способи запиту системних послуг для створення моделі ланцюга Маркова. Ланцюг Маркова використовується як вектор ознак для виконання завдань класифікації за допомогою алгоритмів МН.

Механізм під назвою DATDroid був запропонований в [91], який є методом виявлення шкідливих програм на основі динамічного аналізу із загальною точністю 91,7% з використанням алгоритму RF. На початковому етапі вилучення функцій виконувалося шляхом збору системних викликів, запису використання ЦП і пам'яті, а також запису передачі мережевих пакетів.

Для покращення процесу генерації подій у виявленні шкідливих програм Android було запропоновано фреймворк під назвою MEGDroid, який використовує динамічний аналіз, і представлено у роботі [21]. Суть методу

полягає у тому, що вінавтоматично витягує і представляє інформацію, пов’язану із шкідливим ПЗ, як модель, специфічну для домену. Декомпіляція, виявлення моделі, інтеграція та перетворення, аналіз і перетворення та створення подій були кроками, включеними в цю модель. Потім модель була використана для аналізу шкідливого ПЗ після навчання з набором даних AMD. Ця модель витягла всі можливі джерела подій із шкідливого програмного коду та була розроблена як плагін Eclipse. Виходячи з результатів, MEGDroid забезпечує краще покриття виявлення шкідливих програм за допомогою створення інтерфейсу користувача, при цьому логінізація системних події та моніторинг системних викликів є відсутніми в цьому методі.

3 ПРАКТИЧНА ЧАСТИНА. ВИЯВЛЕННЯ ШКІДЛИВОГО ПЗ В ANDROID ПРИСТРОЯХ

3.1 Етапи виявлення шкідливого ПЗ з набору даних

В практичній частині кваліфікаційної роботи було використано два набори даних, один з них це набір дозволів, який надається програмному додатку на Android, а інший набір – сигнатурний. Тобто у роботі використовується статистичний підхід до виявлення зловмисного ПЗ, що описаний більш детальніше у п. 2.3.1. Одним із завдань є порівняння якості класифікації, а відповідно і виявлення зловмисного ПЗ на кожному з цих наборів.

Для досягнення цієї мети практичну роботу розділено на кілька послідовних етапів, що представлені на рис. 3.1.

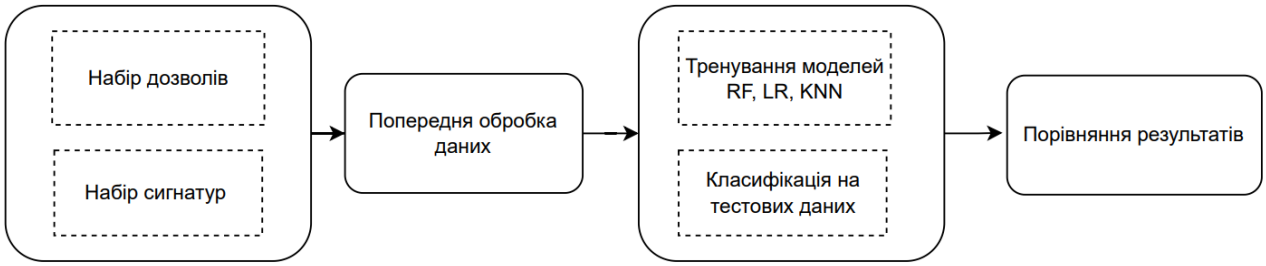


Рисунок 3.1 – Етапи виявлення шкідливого ПЗ в Android пристроях

Для попередньої обробки використовуються бібліотека Pandas (для завнтаження даних) та NumPy. На цьому етапі необроблені дані перевіряються на їх розподіл, тобто співвідношення промаркованих даних в наборі не зловмисного ПЗ до зловмисного, відбувається виділення важливих ознак та формується набрі даних для подання його на модель МН.

Другий етап характеризується застосуванням моделей Logistic Regression, KNN та RF до кожного із підготовлених наборів даних, відбувається оптимізація параметрів моделей, тобто їх тренування на тренувальному наборі даних. Для

виконання задачі другого етапу було використано бібліотеку sklearn для класифікації та пов’язаних операцій.

На третьому етапі виконано порівняння результатів та зроблено висновки щодо найоптимальнішої моделі до кожного з датасетів.

На кожному етапі застосовувалась бібліотеки візуалізації matplotlib і seaborn для створення графіків та ергономічного представлення даних.

3.2 Попередня обробка даних та формування набору ознак

Набір необроблених дозволів програмних додатків Android (dataset_1.csv) отримано з джерела: Махіндру, Арвінд (2018), “Зловмисне програмне забезпечення Android та набір даних про нормальні дозволи”, дані Mendeley, V5, doi: 10.17632/958wvr38gy.5

Дані про сигнатури (dataset_2.csv) отримані з відкритої бази даних Malgenome – Android Malware Genome Project, який є сховищем тисяч додатків Android, попередньо класифікованих як шкідливі та незловмисні.

В цьому пункті набори дозволів та сигнатур досліджуються окремо, після того до них застосовуються методи зменшення ознак та виділення найбільш інформативних, в результаті буде обтримано вхідні набори для виконання алгоритмів класифікації. Для ініціалізації роботи було підключено усі необхідні біліотеки, що представлені у Лістингу 3.1.

Лістинг 3.1 – Імпорт необхідних бібліотек

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import itertools
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

3.2.1 Попередня обробка набору дозволів для програмних додатків

Для початку було досліджено dataset_1.csv та визначено які саме ознаки представлені у наборі, їхнє значення та загальна кількість.

Лістинг 3.2 – Завантаження даних набору дозволів

```
data_1 = pd.read_csv('dataset_1.csv') # dataset_1 is the permissions data
print('The permissions data has {} row and {} columns'.format(data_1.shape[0], data_1.shape[1]))
```

```
The permissions data has 28849 row and 176 columns
```

У наборі є 176 стовпців, більшість яких із них відображають дозволи для програмного додатку. Для початку було проглянуто перші кілька рядків, щоб зрозуміти структуру цього набору.

Лістинг 3.3 – Відображення перших 5 рядків з набору даних дозволів

```
pd.options.display.max_columns = 5
data_1.head()
```

	Package	Category	...	your_personal_information_write_to_user_defined_dictionary	class
0	net.iconchanger	Personalization	...	0	malware
1	com.Jaaru.TruthorDareFree	Comics	...	0	malware
2	com.jrtstudio.iSyncr	Music & Audio	...	0	malware
3	com.ftbsports.fmrm	Sports Games	...	0	malware
4	com.sillycube.android.Rushing	Sports Games	...	0	malware

5 rows × 176 columns

“Package” є ідентифікатором кожного рядка набору. “Category” – відображає інформацію про категорію додатку. “Class” вказує чи є програма шкідливою чи незловмисною. Усі інші стовпці пов’язані з дозволами, і вони є булевими змінними (bool) – 1 представляє запити програми на цей дозвіл, а 0 показує, що програма не запитує цього дозволу.

Зроблено перевірку чи є в наборі даних пропущені чи не заповнені ознаки. У випадку, якщо такі є, їх необхідно видалити. Зазвичай такі не заповнені рядки заповнюють середніми значеннями, нулями, найбліш часто повторювальними. Обрання техніки заповнення даних, які є відсутніми, залежить від набору даних та природи ознак. Також для виконання операцій класифікації, змінна в колонці “Class” повинна бути булевою, тому також було виконано перетворення.

Лістинг 3.4 – Перевірка на відсутні дані в рядках та перетворення типу

```
data_1['class'] = data_1['class'].map({'malware': 1, 'benign': 0})
print('The maximum number of missing values in any column is: {}'.format(data_1.isnull().sum().max()))
```

```
The maximum number of missing values in any column is: 0
```

Як бачимо не заповнених даних немає, тому додаткових дій з набором даних не було виконано. Також було перетворено дані в колонці “Class” в булевий тип.

Представимо кілька дозволів для кращого розуміння:

Таблиця 3.1 – Приклади дозволів у наборі даних

default_access_all_system_downloads	default_modify_google_settings
default_directly_install_applications	your_messages_receive_sms
default_read_phone_state_and_identity	default_change_screen_orientation
your_personal_information_read_contact_data	default_write_contact_data
hardware_controls_take_pictures_and_videos	default_read_contact_data

Набір даних має 173 стовпці, що відображають дозвіл програмному додатку виконувати певні функції. Проте не всі потрібно використовувати в класифікації, оскільки це спричинить проблеми з продуктивністю багатьох алгоритмів аналізу даних. Закон економії свідчить про те, що найкращим рішенням проблеми є те, яке використовує найменшу кількість ознак. Тому в роботі було реалізовано зменшення кількості ознак.

Для початку не будемо використовувати ті ознаки, які у відповідному стовпці мають для всіх рядків одне і те ж значення. Тобто за допомогою них не можливо відрізнити зловмисне ПЗ від не шкідливого. Для їх пошуку застосуємо Лістинг 3.5. Отримано в результаті лише таких 10 колонок, що не є надто багато.

Тому для знаходження найбільш важливих ознак було використано χ^2-тест для категоріальних змінних. Обчислюючи показники χ^2 між кожною ознакою та цільовою змінною “класу”, можна визначити основні відмінні ознаки.

Лістинг 3.5 – Знаходження ознак із однаковими значенням в усіх рядках

```
freq_cols = data_1.loc[:,[x for x in data_1.columns.tolist() if x not in ['Package', 'Category', 'class']]].apply(
    lambda col: col.value_counts(normalize=True)).transpose()

select_cols = freq_cols.loc[np.logical_and(pd.isnull(freq_cols[0]) == False,
                                           pd.isnull(freq_cols[1]) == False),:].index.tolist()

print('There are {} columns that are all 0 or all 1'.format(len(freq_cols) - len(select_cols)))
```

```
There are 10 columns that are all 0 or all 1
```

Оцінка χ^2 розраховується на основі комбінованої таблиці частот для кожної ознаки як предсталено на рис 3.2:

		Клас		
		Безпечне	Шкідливе	Всього
Ознака	0	18	5	23
	1	2	30	32
	Всього	20	35	55

очікуване$_{11}$ =20*23/55=8,36

O_{11} =(18-8,36)*(18-8,36)/8,36=11,11

		Клас	
		Безпечне	Шкідливе
Ознака	0	O_{11}	O_{12}
	1	O_{21}	O_{22}

$\chi^2 = O_{11} + O_{12} + O_{21} + O_{22}$

Рисунок 3.2 – Приклад розрахунку оцінки χ^2 для однієї ознаки

Якщо оцінка χ^2 перевищує критичне значення χ^2 для вибраного рівня значущості, можна сказати, що ознака має певну асоціацію зі змінною класу і її потрібно використати. Даний тест був застосований до всіх ознак, 20 кращих ознак були обрані відповідно до їх оцінки χ^2 із змінною класу. Лістинг 3.6 демонструє це і в результаті було отримано 20 ознак, які є інформативними (діагностичними) і будуть використовуватись в подальшій класифікації.

Для класифікатора дуже важливо, щоб в наборі даних була достатня кількість промаркований екземплярів про зловмисне та не шкідливе ПЗ. Тому було перевірено розподіл між ними та представлено на рис. 3.3. В результаті 65% даних належить до класу “Не шкідливе ПЗ”, а 35% – до класу “Зловмисне ПЗ”. У кожному класі достатньо зразків, щоб продовжити аналіз.

Лістинг 3.6 – Знаходження інформативних ознак

```
chi2_features = SelectKBest(chi2, k=20)
X_kbest = chi2_features.fit_transform(data_1_select.loc[:,
                                                        [x for x in data_1_select.columns.tolist() if
                                                         x not in ['Package', 'Category', 'class']]], data_1_select['class'])
select_cols = [data_1_select.columns.tolist()[i] for i in chi2_features.get_support(indices=True)]
data_1_select = data_1_select.loc[:, select_cols + ['class']]
select_cols
```

```
['default_audio_file_access',
 'default_access_the_cache_filesystem',
 'default_access_to_passwords_for_google_accounts',
 'default_control_location_update_notifications',
 'default_discover_known_accounts',
 'default_force_device_reboot',
 'default_interact_with_a_device_admin',
 'default_permission_to_install_a_location_provider',
 'default_read_phone_state_and_identity',
 'default_write_contact_data',
 'hardware_controls_record_audio',
 'system_tools_format_external_storage',
 'your_accounts_access_all_google_services',
 'your_accounts_contacts_data_in_google_accounts',
 'your_messages_read_sms_or_mms',
 'your_messages_receive_sms',
 'your_messages_send_sms-received_broadcast',
 'your_personal_information_add_or_modify_calendar_events_and_send_email_to_guests',
 'your_personal_information_read_calendar_events',
 'your_personal_information_write_contact_data']
```

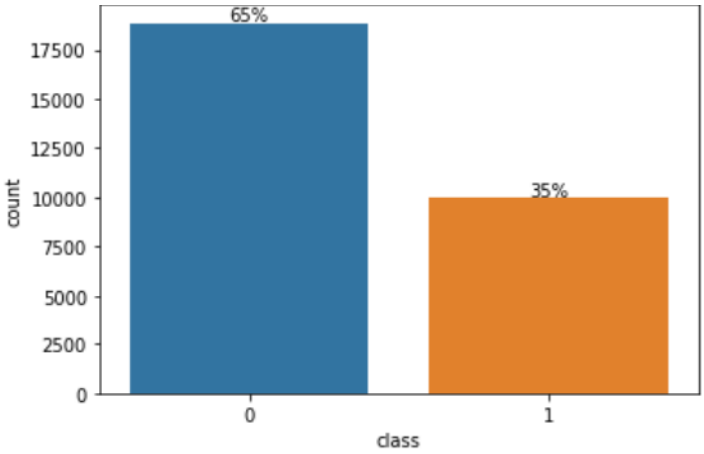


Рисунок 3.3 – Розподіл ознаки “Class” в наборі даних дозволів (синій – “Безпечне ПЗ”, оранжевий – “Зловмисне ПЗ”)

Останнім кроком є візуальне представлення розподілу “0” та “1” для кожної ознаки за показниками “Безпечного ПЗ” та “Зловмисного ПЗ”. Такий крок був необхідний для того, щоб визначити які функції найбільше відрізняють шкідливі та безпечні програми. Результати представлені у додатку А та свідчать, що ознаки “read_phone_state_and_identity” і “write_contact_data” відрізняють шкідливе програмне забезпечення від безпечного в більшості зразків. Це означає, що 23% шкідливих програм запитують дозвіл на “read_phone_state_and_identity”, а 0%

безпечних програми роблять це. Тому, якщо програма запитує цей дозвіл, ми будемо класифікувати як зловмисне ПЗ.

3.2.2 Попередня обробка набору сигнатур ПЗ

Першим кроком є первинне знайомство із dataset_2.csv та визначено які саме ознаки представлені у наборі, їхнє значення та загальна кількість.

Лістинг 3.7 – Завантаження даних набору сигнатур

```
data_2 = pd.read_csv('dataset_2.csv')
print('The signatures data has {} row and {} columns'.format(data_2.shape[0], data_2.shape[1]))
```

```
The signatures data has 3799 row and 73 columns
```

Лістинг 3.8 – Відображення перших 5 рядків з набору даних дозволів

```
pd.options.display.max_columns = 8
data_2.head()
```

	transact	onServiceConnected	bindService	attachInterface	...	Ljava.lang.Class.getResource	defineClass	findClass	class
0	0	0	0	0	...	1	0	0	S
1	0	0	0	0	...	0	0	0	S
2	0	0	0	0	...	0	0	0	S
3	0	0	0	0	...	0	0	0	S
4	1	1	1	1	...	1	0	0	S

5 rows × 73 columns

У наборі є 73 стовпців, більшість яких із них відображають сигнатури API тоді як одна є змінною “Class”, яка вказує, чи є запис шкідливим програмним забезпеченням чи безпечним. Сигнатури API в основному є частинами вихідного коду програми Android. Ці сигнатури для певної програми міститимуть усі можливі операції та взаємодії програми з телефоном Android. Кожен стовпець у цьому наборі даних представляє одну таку функцію у вихідному коді програми.

Як видно з перших кількох рядків вище, усі змінні, пов’язані з сигнатурою, є двійковими. “1” вказує, що вихідний код програми містить сигнатуру, а “0” вказує, що в ньому немає цієї сигнатури.

Зроблено перевірку чи є в наборі даних пропущені чи не заповнені ознаки аналогічно до попереднього прикладу з набором даних дозволів. Також для

виконання операцій класифікації, змінна в колонці “Class” повинна бути булевою, тому також було виконано перетворення.

Лістинг 3.9 – Перевірка на відсутні дані в рядках та перетворення типу

```
print('The maximum number of missing values in any column is: {}'.format(data_2.isnull().sum().max()))
The maximum number of missing values in any column is: 0
data_2['class'] = data_2['class'].map({'S': 1, 'B': 0})
```

Як бачимо не заповнених даних немає, тому додаткових дій з набором даних не було виконано. Також було перетворено дані в колонці “Class” в булевий тип.

Як і раніше, деякі назви стовпців сигнатур показані нижче, щоб зрозуміти тип доступних функцій:

Таблиця 3.2 – Приклади сигнатур у наборі даних

transact	System.loadLibrary
android.content.pm.PackageInfo	Ljava.net.URLDecoder
android.telephony.SmsManager	sendMultipartTextMessage
HttpPost.init	Ljava.lang.Class.getCanonicalName

Як видно з назв цих стовпців, ці сигнатури є частиною коду програми для Android і використовуються для виконання конкретних операцій. Можуть бути деякі сигнатури, які не є поширеними в безпечних програмах, але поширені в програмах зловмисного ПЗ.

Аналогічно до попереднього набору будемо обирати ознаки (всього 72 в наборі сигнатур), які можуть бути діагностичними і нашоб метою було отримання 20 таких ознак. Для початку було обчислено кількість значень кожного стовпця в наборі даних, щоб побачити, які з 10 найчастіших сигнатур характеризують зловмисне ПЗ (лістинг 3.10). шкідливих програм. не будемо використовувати ті ознаки, які у відповідному стовпці мають для всіх рядків одне і те ж значення. Тобто за допомогою них не можливо відрізнити зловмисне ПЗ від не шкідливого. Для їх пошуку застосуємо Лістинг 3.5.

На основі результату виконання лістингу 3.5 можна помітити, що деякі функції мають аналогічні назви, наприклад, “Binder” і “IBinder”. За допомогою кореляційного аналізу було виділено сигнатури, які сильно корелюють з іншими. На рис. 3.4 наведена кореляційна теплова карта всіх функцій у наборі даних. Більш світлі кольори в сітці вказують на вищу кореляцію.

Лістинг 3.7 – Знаходження ознак популярних для зловмисного ПЗ

```
freq_cols = data_2.loc[:,[x for x in data_2.columns.tolist() if x not in ['class']]].apply(
    lambda col: col.value_counts(normalize=True)).transpose()

freq_cols.sort_values(1, ascending=False).head(10)
```

	0	1
Binder	0.108450	0.891550
IBinder	0.114767	0.885233
android.os.IBinder	0.114767	0.885233
Ljava.lang.Object.getClass	0.182153	0.817847
android.content.pm.PackageInfo	0.194525	0.805475
onBind	0.214793	0.785207
HttpUriRequest	0.382996	0.617004
TelephonyManager.getDeviceId	0.385101	0.614899
Ljava.lang.Class.forName	0.396420	0.603580
Ljava.lang.Class.getMethod	0.414846	0.585154

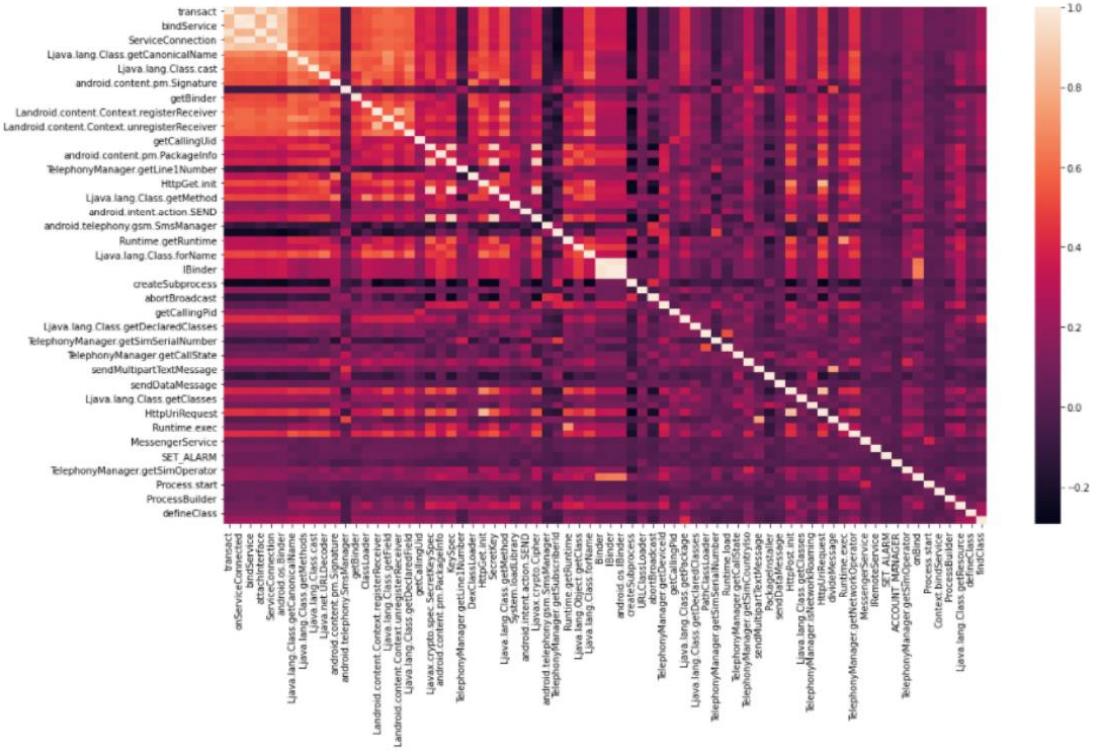


Рисунок 3.4 – Теплова карта кореляції сигнатур

Як бачимо існує багато змінних, які сильно корелюють з один з одним, по суті вони є взаємозамінними. Було встановлено поріг 0,8, щоб представити високу кореляцію та видалити всі змінні, які мають коефіцієнт кореляції з будь-якими іншими змінними, більшими за 0,8 (Лістинг 3.8).

Лістинг 3.8 – Видалення ознак з коефіцієнтом кореляції більшим за 0,8

```
rmv_cols = []
for c in select_cols:
    if c not in rmv_cols:
        corr_cols = cor.loc[cor[c] >= 0.8,c].index.tolist()
        corr_cols = [x for x in corr_cols if x != c]
        if len(corr_cols) > 0:
            rmv_cols += corr_cols

print('Using correlation, {} columns are removed'.format(len(rmv_cols)))
```

```
Using correlation, 13 columns are removed
```

Після виконання вказаних операцій кількість ознак набору даних було зменшено до 59. Аналогічно до попереднього прикладу обираємо 20 найбільш значущих ознак за χ^2 тестом.

Лістинг 3.9 – Знаходження інформативних ознак

```
chi2_features = SelectKBest(chi2, k=20)
X_kbest = chi2_features.fit_transform(data_2_select.loc[:,select_cols], data_2_select['class'])
```

```
select_cols = [data_2_select.columns.tolist()[i] for i in chi2_features.get_support(indices=True)]
select_cols
```

```
['transact',
 'Ljava.lang.Class.getCanonicalName',
 'Ljava.lang.Class.getMethods',
 'Ljava.lang.Class.cast',
 'Ljava.net.URLDecoder',
 'getBinder',
 'Landroid.content.Context.registerReceiver',
 'Ljava.lang.Class.getField',
 'Ljava.lang.Class.getDeclaredField',
 'getCallingUid',
 'TelephonyManager.getLine1Number',
 'HttpGet.init',
 'android.telephony.gsm.SmsManager',
 'TelephonyManager.getSubscriberId',
 'createSubprocess',
 'abortBroadcast',
 'TelephonyManager.getDeviceId',
 'Ljava.lang.Class.getPackage',
 'TelephonyManager.getSimSerialNumber',
 'PackageInstaller']
```

Було перевірено розподіл між ознаками щодо представників шкідивого та безпечного ПЗ та представлено на рис. 3.5. В результаті 6% даних належить до

класу “Безпечне ПЗ”, а 33% – до класу “Зловмисне ПЗ”. У кожному класі достатньо зразків, щоб продовжити аналіз.

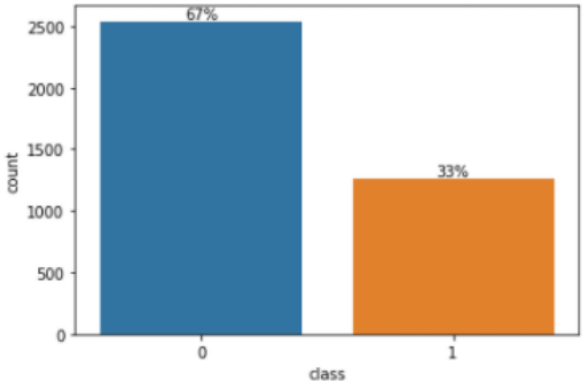


Рисунок 3.5 – Розподіл ознаки “Class” в наборі даних дозволів (синій – “Безпечне ПЗ”, оранжевий – “Зловмисне ПЗ”)

Останнім кроком є візуальне представлення розподілу “0” та “1” для кожної ознаки за показниками “Безпечного ПЗ” та “Зловмисного ПЗ”. Такий крок був необхідний для того, щоб визначити які функції найбільше відрізняють шкідливі та безпечні програми. Результати представлені у додатку Б та свідчать, що ознаки “transact”, “getCanonicalName”, “getSubscriberId” найбільше відрізняють зразки зловмисного ПЗ від безпечного. тощо.

На основі наведених вище аналізів завершено дослідницький аналіз і етап вибору ознак. Наступним кроком є створення моделей класифікації для порівняння показників продуктивності в межах двох типів методів виявлення шкідливих програм і між ними.

3.3 Класифікація з використанням методів LR, KNN, RF

На основі аналітичного огляду в розділі 1 більшість авторів при статистичному аналізу використовують наступні алгоритми машинного навчання LR, KNN, RF, які показували достатньо точні показники при класифікації. Саме тому в роботі було використано їх за аналогією.

LR є одним із найпопулярніших алгоритмів на практиці завдяки ряду властивостей: модель можна навчати дуже ефективно та розподілено, вона добре

масштабується до мільйонів ознак, допускає стислий опис та швидкий алгоритм оцінки (простий точковий добуток), і це можна пояснити – оскільки можливо обчислити внесок кожної функції в остаточну оцінку.

Однак, потрібно зауважити кілька властивостей LR:

- LR передбачає лінійність ознак (незалежних змінних) і логарифмічних частками, вимагаючи, щоб характеристики були лінійно пов'язані з логарифмічними частками. Якщо це припущення порушується, модель буде працювати погано.
- Характеристики не повинні мати або не мати багатоколінеарності; тобто незалежні змінні повинні бути дійсно незалежними одна від одної.
- LR зазвичай вимагає більшого розміру вибірки порівняно з іншими алгоритмами машинного навчання, як-от лінійна регресія.

RF утворюються шляхом простого ансамблювання множинних дерев рішень, як правило, від десятків до тисяч дерев. Після навчання кожного окремого дерева рішень, загальні випадкові прогнози RF робляться шляхом використання статистичного режиму прогнозів окремих дерев для дерев класифікації (тобто кожне дерево "голосує") і середнього статистичного прогнозу окремих дерев для дерев регресії.

Алгоритм *KNN* є найбільш відомим прикладом алгоритму відкладеного навчання. Цей тип техніки машинного навчання відкладає більшість обчислень на час класифікації замість того, щоб виконувати роботу під час навчання. Ліниві моделі навчання не вивчають узагальнення даних на етапі навчання. Натомість вони записують всі точки навчальних даних, які вони пройшли, і використовують цю інформацію, щоб зробити локальні узагальнення навколо тестової вибірки під час класифікації. KNN є одним із найпростіших алгоритмів машинного навчання:

- Фаза навчання просто складається із збереження всіх векторів ознак і відповідних міток зразка в моделі.
- Прогноз класифікації є просто найближчою міткою з k найближчих сусідів тестової вибірки.

Отож в попередніх пунктах 3.2.1 та 3.2.2 було підготовлено дані, що подаються на вхід класифікатора. Для того, щоб навчити класифікатор будемо виконувати дві важливі операції:

- Розділення даних на тестові та тренувальні набори з використанням функції train_test_split.
- Пошук оптимальних параметрів моделі з використанням функції GridSearchCV.

В результаті ми отримаємо в тренувальному наборі 70% вхідних даних з попередньо підготовлених наборів дозволів та сигнатур та 30% в тестувальному. І було визначено, що оптимальним параметром для KNN алгоритму є *k=3,* для LR потрібно параметр *solver="liblinear", n_estimator=100, criterion="gini".*

Тому для кожного набору даних було ініціалізовано моделі наступним чином. З метою оцінки моделей та їх порівняння було використано оцінки точності, повноти та F1. Також виконано 5-кратну перехресну перевірку, яка розбиває тренувальний набір на п'ять рівних частин і запускає модель 5 разів, кожен раз розглядаючи одну з п'яти частин як набір для перевірки. Таким чином обчислюється середнє значення оцінок, що є більш надійним значенням, ніж якщо б модель запускалася лише один раз (лістинг 3.10).

Лістинг 3.10 – Ініціалізація моделей

```
scr = 'recall' # Recall score used for model evaluation
model_lr = LogisticRegression(solver='liblinear')
model_kn = KNeighborsClassifier(n_neighbors=3)
model_rf = RandomForestClassifier()
skf = StratifiedKFold(5) # 5 fold stratified cross validation
```

3.3.1 Виявлення шкідливого ПЗ з використанням методів LR, KNN, RF на наборі даних дозволів програмних додатків

Для початку було натренована KNN модель на 70% первинного попередньо опрацьованого набору даних дозволів для програмних додатків. Після того модель класифікація було перевірено на тестових даних та представлено показники її продуктивності разом із матрицею помилок (лістинг 3.11).

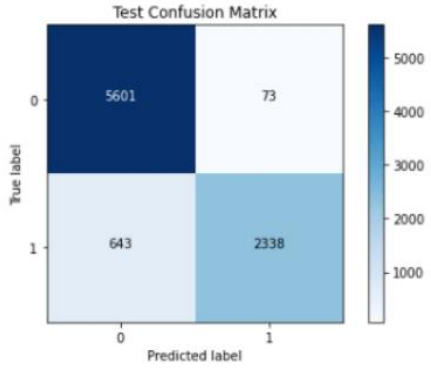
Лістинг 3.11 – Ініціалізація моделі KNN на тренувальних даних та результати роботи моделі на тестувальних даних

```
# kNN with cross validation to evaluate the average performance score with train dataset
sc_kn = cross_val_score(model_kn, X_train, y_train, cv=skf, scoring=scr)
print("kNN's average recall across validation sets is: " + str(round(sc_kn.mean() * 100, 2)) + '%')
```

```
kNN's average recall across validation sets is: 78.9%
```

```
y_pred_test_kn = kn.predict(X_test)
cnf_matrix_test_kn = confusion_matrix(y_test, y_pred_test_kn)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_kn, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 96.97%
Recall: 78.43%
F1 Score: 86.72
```



Аналогічно виконуємо для LR (лістинг 3.12) та RF (лістинг 3.13).

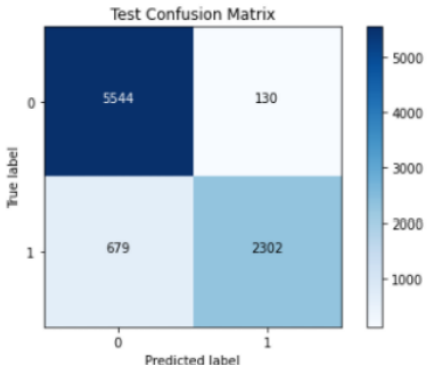
Лістинг 3.12 – Ініціалізація моделі LR на тренувальних даних та результати роботи моделі на тестувальних даних

```
sc_lr = cross_val_score(model_lr, X_train, y_train, cv=skf, scoring=scr)
print("Logistic Regression's average recall across validation sets is: " + str(round(sc_lr.mean() * 100, 2)) + '%')
```

```
Logistic Regression's average recall across validation sets is: 77.71%
```

```
# Make predictions on the test dataset using logistic regression and visualize the confusion matrix
y_pred_test_lr = lr.predict(X_test)
cnf_matrix_test_lr = confusion_matrix(y_test, y_pred_test_lr)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_lr, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 94.65%
Recall: 77.22%
F1 Score: 85.05
```



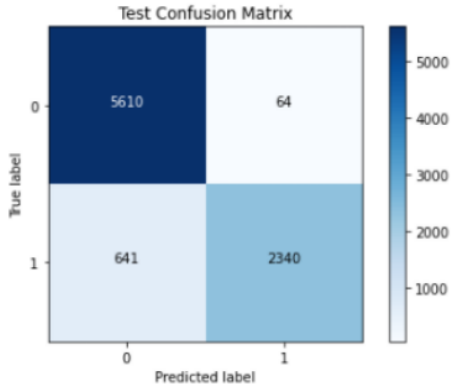
Лістинг 3.13 – Ініціалізація моделі RF на тренувальних даних та результати роботи моделі на тестувальних даних

```
sc_rf = cross_val_score(model_rf, X_train, y_train, cv=skf, scoring=scr)
print("Random Forest's average recall across validation sets is: " + str(round(sc_rf.mean() * 100, 2)) + '%')
```

```
Random Forest's average recall across validation sets is: 79.22%
```

```
y_pred_test_rf = rf.predict(X_test)
cnf_matrix_test_rf = confusion_matrix(y_test, y_pred_test_rf)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_rf, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 97.34%
Recall: 78.5%
F1 Score: 86.91
```



В результаті в межах набору дозволів програмних додатків найкращі показники за оцінками повноти, точності та F1 показав алгоритм RF.

3.3.2 Виявлення шкідливого ПЗ з використанням методів LR, KNN, RF на наборі даних сигнатур

Аналогічно до набору дозволів та класифікації на його основі з використанням моделей LR, KNN, RF було виконано навчання та клаисфікацію на наборі даних сигнатур. Спершу кожна з моделей навчалась на тренувальному наборі, що також складав 70% первинного попередньо опрацьованого набору даних сигнатур програмних додатків. Після того модель класифікація було перевірено на тестових даних. Якість навчання вимірюється з використанням трьох оцінок точності, повноти та F1 для яких побудовано матрицю похибок, що відображаю відповідно TP, FP, FN, TN.

Лістинги роботи моделей та результати класифікацій з викорситанням моделей KNN, LR, RF представлені відповідно у лістингах 3.14, 3.15, 3.16.

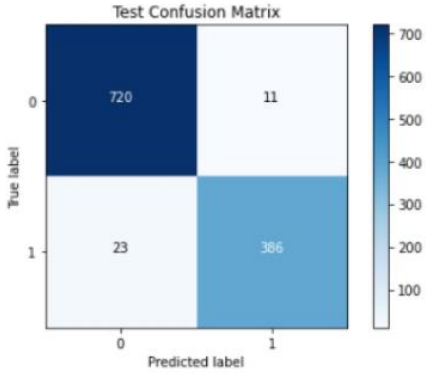
Лістинг 3.14 – Ініціалізація моделі KNN на тренувальних даних сигнатур та результати роботи моделі на тестувальних даних

```
sc_kn = cross_val_score(model_kn, X_train, y_train, cv=skf, scoring=scr)
print("kNN's average recall across validation sets is: " + str(round(sc_kn.mean() * 100, 2)) + '%')
```

```
kNN's average recall across validation sets is: 94.01%
```

```
y_pred_test_kn = kn.predict(X_test)
cnf_matrix_test_kn = confusion_matrix(y_test, y_pred_test_kn)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_kn, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 97.23%
Recall: 94.38%
F1 Score: 95.78
```



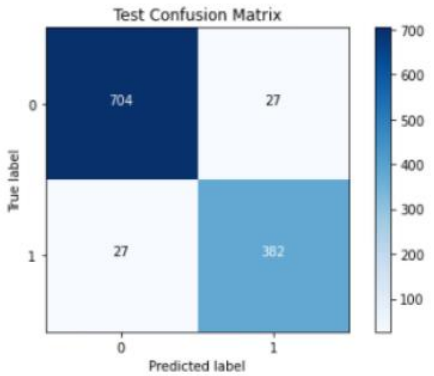
Лістинг 3.15 – Ініціалізація моделі LR на тренувальних даних сигнатур та результати роботи моделі на тестувальних даних

```
# Logistic Regression with cross validation to evaluate the average performance score with train dataset
sc_lr = cross_val_score(model_lr, X_train, y_train, cv=skf, scoring=scr)
print("Logistic Regression's average recall across validation sets is: " + str(round(sc_lr.mean() * 100, 2)) + '%')
```

```
Logistic Regression's average recall across validation sets is: 92.6%
```

```
# Make predictions on the test dataset using logistic regression and visualize the confusion matrix
y_pred_test_lr = lr.predict(X_test)
cnf_matrix_test_lr = confusion_matrix(y_test, y_pred_test_lr)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_lr, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 93.4%
Recall: 93.4%
F1 Score: 93.4
```



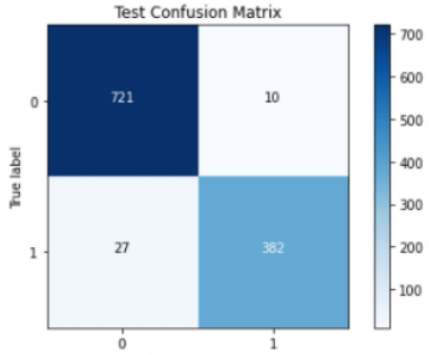
Лістинг 3.16 – Ініціалізація моделі RF на тренувальних даних сигнатур та результати роботи моделі на тестувальних даних

```
# Random Forest with cross validation to evaluate the average performance score with train dataset
sc_rf = cross_val_score(model_rf, X_train, y_train, cv=skf, scoring=scr)
print("Random Forest's average recall across validation sets is: " + str(round(sc_rf.mean() * 100, 2)) + '%')
```

```
Random Forest's average recall across validation sets is: 93.9%
```

```
# Make predictions on the test dataset using Random Forest and visualize the confusion matrix
y_pred_test_rf = rf.predict(X_test)
cnf_matrix_test_rf = confusion_matrix(y_test, y_pred_test_rf)
class_names = ['0','1']
plot_confusion_matrix(cnf_matrix_test_rf, classes = class_names, title = 'Test Confusion Matrix')
```

```
Precision: 97.45%
Recall: 93.4%
F1 Score: 95.38
```



В результаті в межах набору дозволів програмних додатків найкращі показники за оцінками повноти, точності та F1 показав алгоритм KNN.

3.4 Порівняння результатів

Представимо результати класифікації двох наборів дозволі та сигнатур.

Таблиця 3.3 – Порівняльний аналіз якості класифікаційних моделей для наборів дозволів та сигнатур.

Вхідні дані	Оцінки моделі	KNN	LR	RF
Набір дозволів	Точність, %	96,97	94,65	97,34
	Повнота, %	78,43	77,22	78,5
	F1-score, %	86,72	85,05	86,91
Набір сигнатур	Точність, %	97,23	93,4	97,45
	Повнота, %	94,38	93,4	93,4
	F1-score, %	95,78	93,4	95,38

Найкращі моделі для кожного набору даних виділені вище кольором. Як ми бачимо, порівнявши цифри, найкращою моделлю для підходу на основі дозволів є Random Forest, а найкращою моделлю для підходу на основі сигнатур є KNN Classifier. Однак для всіх практичних цілей ми можемо припустити, що і Random Forest, і KNN Classifier працюють однаково.

Крім того, якщо порівняти обидва підходи, можна побачити, що підхід на основі сигнатур дає набагато кращу оцінку повноти, ніж підхід на основі дозволів, що вказує на те, що використання функцій на основі сигнатур краще для виявлення зловмисного ПЗ.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

4.1 Проведення інструктажів з охорони праці

У статті 18 Закону про охорону праці визначено, що працівники під час прийняття на роботу та протягом роботи мають проходити інструктаж з питань охорони праці. Тих, хто не пройшов інструктажу, не допускають до роботи. Порядок проведення інструктажів з питань охорони праці на підприємстві визначає глава 6 Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держнагляд охорони праці від 26.01.2005 № 15 (далі – Типове положення). Відповідальність за організацію і проведення інструктажів з охорони праці несе роботодавець. Будь-який інструктаж з охорони праці на робочому місці, навчання та перевірку знань з ОП роботодавець зобов’язаний організувати за свої кошти. Основним документом щодо підтвердження проведення інструктажу з охорони праці є запис у журналі реєстрації інструктажів. Види інструктажів з питань охорони праці За характером і часом проведення інструктажі з охорони праці поділяють на: вступний; первинний; повторний; позаплановий; цільовий. Далі розглянемо детальніше, як проводити кожен із цих видів інструктажів.

Вступний інструктаж з охорони праці на підприємстві проводять: усім працівникам, яких беруть на постійну або тимчасову роботу, незалежно від їх освіти, стажу роботи та посади; працівникам інших організацій, які прибули на підприємство і беруть безпосередню участь у виробничому процесі або виконують інші роботи для підприємства; учням та студентам, які проходитимуть на підприємстві трудове або професійне навчання; учасникам екскурсії на підприємство. Первинний інструктаж з охорони праці Первинний інструктаж з питань охорони праці проводять із працівниками: новоприйнятими на постійну чи тимчасову роботу; відрядженими з іншого підприємства; яких перевели з іншого структурного підрозділу підприємства; які виконуватимуть нову роботу. Також цей вид інструктажу потрібно слухачами та студентами навчальних закладів: до

початку трудового або професійного навчання; перед виконанням кожного навчального завдання, пов'язаного з використанням різних механізмів, інструментів, матеріалів тощо.

Повторний інструктаж з охорони праці проводять, щоб працівник повторив і закріпив знання, які здобув під час первинного інструктажу на робочому місці. Терміни проведення повторного інструктажу встановлюються НПАОП, які діють у галузі, або роботодавцем з урахуванням конкретних умов праці, проте не рідше: одного разу на три місяці – для робіт з підвищеною небезпекою; одного разу на шість місяців – для інших робіт.

Позаплановий інструктаж з охорони праці проводять у тому разі, якщо на підприємстві: введено в дію нові або переглянуті НПАОП, внесено зміни та доповнення до них; змінено технологічний процес, замінено або модернізовано устаткування, прилади, інструменти, вихідну сировину, матеріали тощо; працівниками порушено вимоги НПАОП, й це призвело до травм, аварій, пожеж тощо; у разі перерви понад 30 календарних днів у роботі виконавця робіт з підвищеною небезпекою та понад 60 днів – у роботі виконавця інших робіт.

Цільовий інструктаж з охорони праці проводять у разі ліквідації аварії або стихійного лиха та проведення робіт, на які потрібен наряд-допуск, наказ або розпорядження. У разі виконання робіт, на які потрібно оформлювати наряд-допуск, цільовий інструктаж реєструють у цьому наряді-допуску, а в журналі реєстрації інструктажів – не обов’язково.

Хто проводить інструктажі з питань охорони праці на підприємстві Вступний інструктаж проводить спеціаліст служби охорони праці або інший фахівець відповідно до наказу роботодавця, який пройшов навчання і перевірку знань із питань охорони праці. Розподіл обов’язків з організації навчання з питань охорони праці на підприємстві зі строками його проведення визначають у локальних нормативних документах підприємства, таких, як Положення про навчання з питань охорони праці підприємства, Положення про кадрову службу підприємства, а також у наказах та посадових інструкціях конкретних працівників. Первинний, повторний, позаплановий і цільовий інструктажі з ОП

проводять безпосередній керівник робіт або фізична особа, яка використовує найману працю. Контроль за проведенням інструктажів з ОП здійснює служба охорони праці відповідно до пункту 3.14 Типового положення про службу охорони праці, затвердженого наказом Держпраці від 15.11.2004 № 255, а також комісія адміністративно-громадського контролю, що діє на підприємстві.

Після первинного, повторного, позапланового і цільового інструктажів проводять перевірку знань у формі усного опитування або за допомогою технічних засобів, а також перевірку набутих навичок безпечних методів праці. Якщо результати перевірки після первинного, повторного чи позапланового інструктажів незадовільні, протягом 10 днів додатково проводять інструктаж і повторну перевірку знань. У разі незадовільних результатів перевірки після цільового інструктажу особу не допускають до виконання робіт (у цьому разі заборонено й повторну перевірку знань).

4.2 Значення адаптації в трудовому процесі

Закон України “Про охорону праці” передбачає, що роботодавець зобов’язаний створити на робочому місці умови праці та забезпечити додержання вимог законодавства щодо прав працівників у галузі охорони праці. З цією метою роботодавець забезпечує функціонування системи управління охороною праці та несе безпосередню відповідальність за порушення вимог з охорони праці на підприємстві.

Праця людини безпосередньо пов’язана із виробничим середовищем. Працівник може нормально здійснювати трудову діяльність лише тоді, коли умови зовнішнього середовища відповідають оптимальним. Якщо вони стають несприятливими та на протидію їм організм людини включає спеціальний механізм, який зберігає постійність внутрішнього середовища, або змінює його в межах допустимого. Такий механізм називається адаптацією. Адаптація є важливим засобом попередження травмування, виникнення нещасних випадків у трудовому процесі і відіграє значну роль в охороні праці.

Адаптація – це динамічний процес пристосування організму та його органів до мінливих умов зовнішнього середовища.

Адаптація в трудовій діяльності поділяється на фізіологічну, психічну, соціальну та професійну:

- Фізіологічна адаптація – це сукупність фізіологічних реакцій, які є в основі пристосування організму до змін оточуючого середовища і направлені на збереження відносної постійності його внутрішнього середовища. Суть механізму адаптації полягає у змінах меж чутливості аналізаторів, розширенні діапазону фізіологічних резервів організму та зміні в певних межах параметрів фізіологічних функцій (підвищується стійкість організму до холоду, тепла, недостачі кисню, змін барометричного тиску та ін.) Фізіологічна адаптація до праці має активний характер і за сприятливих умов виробничого середовища та оптимальних навантажень веде до підвищення стійкості та продуктивності організму, збільшення його резервних можливостей, зменшення захворювань і травматизму.
- Психічна адаптація – це процес встановлення оптимальної відповідності особистості до оточуючого середовища в процесі діяльності. Психічна адаптація в процесі праці залежить від психічних властивостей працівника, його психічного стану, психологічних реакцій на стреси, що виникають на роботі, кваліфікації та культури людини, особливостей професійної діяльності, конкурентних умов праці тощо.
- Соціальна адаптація – це пристосування працюючої людини до системи відносин у робочому колективі з його нормами, правилами, традиціями, ціннісними орієнтаціями. При несприятливому протіканні соціальної адаптації підвищується рівень стресу на роботі, наслідки якого позначаються на поведінці працівника та можуть призвести до між особових конфліктів, нещасних випадків.
- Професійна адаптація – це адаптація до трудової діяльності з усіма її складовими і адаптація до робочого місця, знарядь та засобів праці, об'єктів та предметів праці, особливостей технологічного процесу, головних параметрів роботи тощо. Професійна адаптація виражається у розвитку стійкого позитивного

ставлення працівника до своєї професії, певного рівня оволодіння ним специфічними навичками та уміннями у формуванні необхідних для якісного виконання роботи властивостей.

У процесі адаптації працівник проходить наступні стадії:

- стадія ознайомлення, на якій працівник одержує інформацію про нову ситуацію в цілому, про критерії оцінки різних дій, про еталони, норми поведінки;
- стадія пристосування, на цьому етапі працівник переорієнтується, визнаючи головні елементи нової системи цінностей, але поки продовжує зберігати багато своїх установок;
- стадія асиміляції, коли здійснюється повне пристосування до середовища, ідентифікація з новою групою. Ідентифікація, коли особисті цілі працівника ототожнюються з цілями трудової організації, підприємства, фірми і т.д. По характері ідентифікації розрізняють три категорії працівників: байдужні, частково ідентифіковані і цілком ідентифіковані. Ядро будь-якої трудової організації складають цілком ідентифіковані працівники. І кінцеві результати такої трудової організації завжди високі.

Швидкість адаптації залежить від багатьох факторів. Нормальний термін адаптації для різних категорій працівників складає від 1 року до 3 років. Невміння ввійти в трудову організацію (колектив), адаптуватися в ній викликає явище виробничої і соціальної дезорганізації. Більшість соціально-трудових відносин виступає для людини у формі соціально-трудових процесів, у яких вона бере участь протягом трудового життя, починаючи зі свого першого робочого місця та участі в процесі виробничої адаптації.

Виробнича адаптація. У соціології праці останніми роками зростає увага до проблем адаптації людини у сфері праці, що є свідченням визнання тієї суттєвої ролі, яку відіграє процес адаптації в трудовій діяльності людини. Водночас розуміння сутності адаптації утруднене існуючими ще функціональними підходами до адаптації працівника з виокремленням лише деяких сторін його взаємодії з виробництвом. Такий підхід залишається типовим для багатьох соціологів, у тому числі й західних.

Необхідно підходити до дослідження адаптації особистості працівника, розглядаючи його як цілісну людину в різноманітності видів і форм виконуваної ним діяльності. З огляду на цілісне розуміння сутності працівника виробнича адаптація не обмежується професійною сферою, а охоплює сукупність соціально-трудових відносин, що зумовлює її структуру. До основних структурних елементів виробничої адаптації відносять професійну, організаційну, матеріально-побутову, соціально-психологічну та адаптацію у сфері дозвілля. В основі механізму виробничої адаптації лежить адаптивна потреба індивіда, опосередкована взаємодією з потребою його в трудовій самореалізації. При цьому формується, з одного боку, орієнтаційний мотив поведінки, що спонукає індивіда до здобуття інформації про трудову ситуацію, розширення контактів із соціально-виробничим середовищем, оцінки характеру адаптивної ситуації. З другого боку, мотив опанування конкретної трудової діяльності і досягнення оптимальної взаємодії з виробництвом опосередковує зміст інформації і спрямованість особистісних контактів. В результаті складної полімотивації здійснюється виробнича адаптація, яка враховує можливості реалізації на даному підприємстві життєвих цілей працівників. Успішне (або утруднене) проходження виробничої адаптації молодого працівника залежить від того, наскільки сприятливі (або несприятливі) умови склалися для задоволення його адаптивної потреби. З огляду на структуру останньої до таких умов належать певний рівень взаємної інформації між індивідом і виробництвом, взаємних контактів; порівняння життєвих цілей індивіда з завданнями даного підприємства, а також наявність на виробництві умов для успішної трудової діяльності індивіда.

Отже, кожен із розглянутих видів адаптації впливає на працездатність та здоров'я працівника, формує у нього певний рівень чутливості та стійкості до психоемоційних перевантажень, внаслідок розвитку яких може істотно змінитися надійність професійної діяльності. Наявність і повнота прояву сукупності умов, необхідних для проходження адаптації, будуть визначати як її ефективність, динамізм, так і межі самого процесу. Таким чином, адаптація працівника являє

собою процес його взаємодії з соціально-виробничим середовищем з метою оволодіти новою трудовою ситуацією.

ВИСНОВКИ

Літературний огляд алгоритму атаки шкідливого програмного забезпечення, архітектури Android та її вразливостей, методів виявлення зловмисного програмного забезпечення розглянуто в теоретичних розділах. Дослідження предметної області є передумовою для аналітика в галузі кібербезпеки, а тим паче при виявленні шкідливого ПЗ. Звичайно дуже важливо отримати відомості про шкідливе ПЗ до моменту його розгортання на пристрої, проте такий сценарій важко реалізувати. В роботі було представлено відомі методи виявлення шкідливого ПЗ з використанням статичних та динамічних підходів. Перші дозволяють виявляти шкідливе ПЗ вже по факту зараження ним пристрою, а другі потребують симулювання, що потребує значних затрат.

Практична частина висвітлює покроковий процес виявлення шкідливого ПЗ на основі даних сигнатур та дозволів для програмних додатків. Надзвичайно важливо обрати правильні ознаки для подальшого класифікатора для кожного набору окремо. Для вирішення цієї задачі було використано χ^2-тест та кореляційний аналіз. В результаті із наборів, що складались із 173 та 72 ознак було обрано по 20 ознак відповідно. Це дозволило підвищити якість класифікації, зменшити час на навчання моделей.

Під час дослідження підхід на основі сигнатур забезпечує високоякісні результати. Підхід на основі сигнатур використовує функції, пов’язані з підписами API, доступними у вихідному коді програми. Коди різних додатків пишуться по-різному. Таким чином, в коді є маркери, які допоможуть відрізнити шкідливі програми від безпечних. Крім того, Random Forest і KNN Classifier забезпечують найкращі результати в обох підходах. У проекті було досягнуто показника точності - 95%-97%. Це означає, що ми можемо ідентифікувати 95%-97% шкідливих програм за допомогою цієї методики.

В результаті розроблено алгоритм для виявлення шкідливого ПЗ в Android пристроях, яке є серйозною проблемою для відділів у всьому світі. Імплементація цього алгоритму дозволить швидко ідентифікувати шкідливе програмне

забезпечення, яке може бути потенційно шкідливим для користувачів мобільних пристроїв.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Number of Mobile Phone Users Worldwide from 2016 to 2023 (In Billions). Доступ до ресурсу: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.
2. Asavoae, I.M.; Blasco, J.; Chen, T.M.; Kalutarage, H.K.; Muttik, I.; Nguyen, H.N.; Roggenbach, M.; Shaikh, S.A. Towards automated android app collusion detection. arXiv 2016, arXiv:1603.02308.
3. Chio, C.; Freeman, D. Machine Learning and Security, O'Relly Media, 2018, 125-180.
4. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. J. Netw. Comput. Appl. 2020, 153, 102526.
5. Khan, J.; Shahzad, S. Android Architecture and Related Security Risks. Asian J. Technol. Manag. Res. [ISSN: 2249–0892] 2015, 5, 14–18. Доступ до ресурсу: http://www.ajtmr.com/papers/Vol5Issue2/Vol5Iss2_P4.pdf.
6. Android Runtime (ART) and Dalvik. Доступ до ресурсу: https://source.android.com/devices/tech/dalvik.
7. Cai, H.; Ryder, B.G. Understanding Android application programming and security: A dynamic study. In Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17–22 September 2017; pp. 364–375.
8. Liu, K.; Xu, S.; Xu, G.; Zhang, M.; Sun, D.; Liu, H. A Review of Android Malware Detection Approaches Based on Machine Learning. IEEE Access 2020, 8, 124579–124607.
9. Mobile Operating System Market Share Worldwide. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide/.
10. Bhat, P.; Dutta, K. A survey on various threats and current state of security in android platform. ACM Comput. Surv. (CSUR) 2019, 52, 1–35.

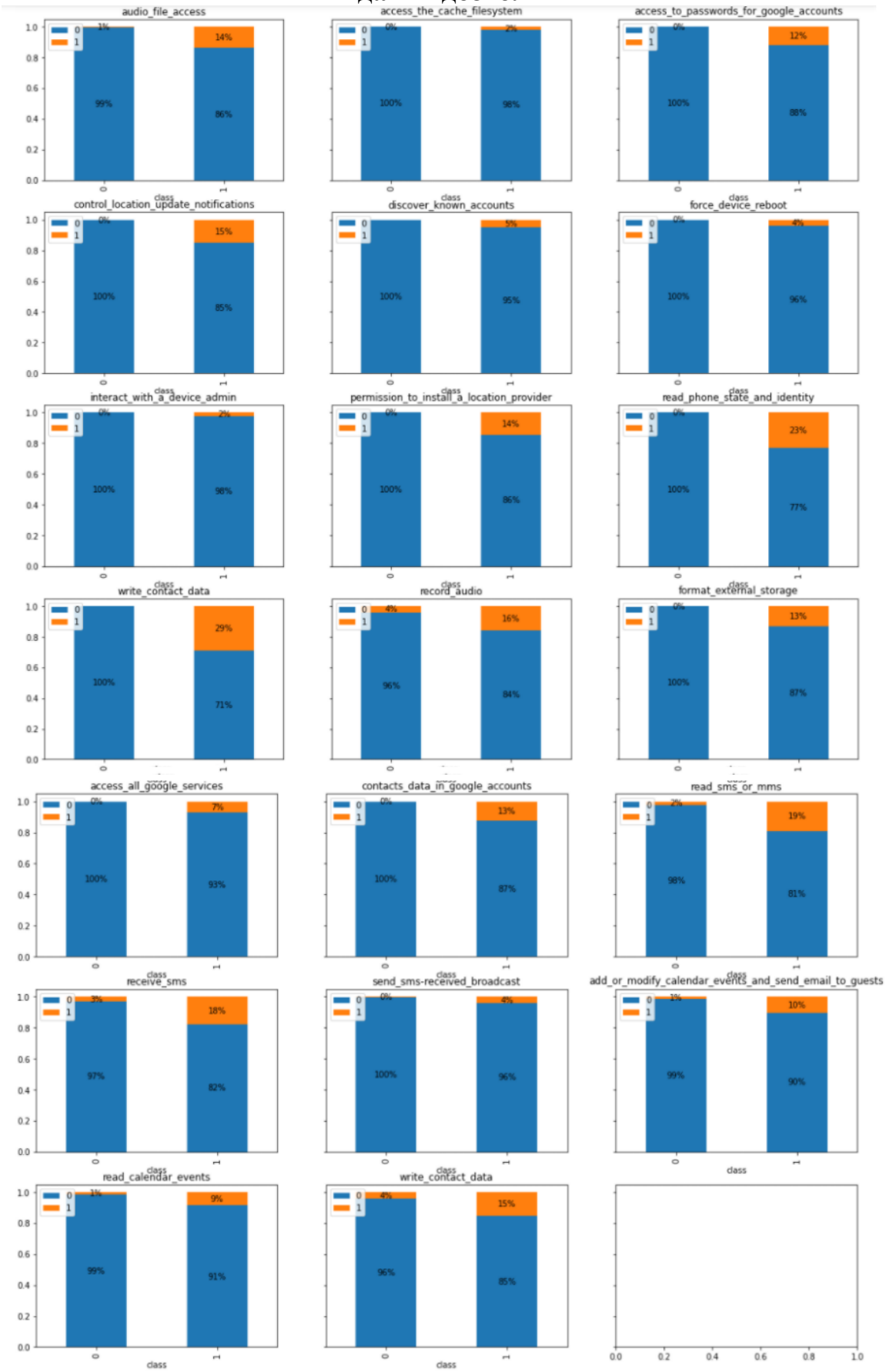
11. Mos, A.; Chowdhury, M.M. Mobile Security: A Look into Android. In Proceedings of the 2020 IEEE International Conference on Electro Information Technology (EIT), Chicago, IL, USA, 31 July–1 August 2020; pp. 638–642.
12. Chen, T.; Mao, Q.; Yang, Y.; Lv, M.; Zhu, J. TinyDroid: A lightweight and efficient model for Android malware detection and classification. Mob. Inf. Syst. 2018, 2018
13. Senanayake, J.; Kalutarage, H.; Al-Kadri, M.O. Android Mobile Malware Detection Using Machine Learning: A Systematic Review. Electronics 2021, 10, 1606. https:// doi.org/10.3390/electronics10131606/.
14. Garg, S.; Baliyan, N. Android Security Assessment: A Review, Taxonomy and Research Gap Study. Comput. Secur. 2020, 100, 102087.
15. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant permission identification for machine-learning-based android malware detection. IEEE Trans. Ind. Inform. 2018, 14, 3216–3225.
16. Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. Information 2021, 12, 185.
17. Choudhary, M.; Kishore, B. HAAMD: Hybrid analysis for Android malware detection. In Proceedings of the 2018 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 4–6 January 2018; pp. 1–4.
18. Cai, L.; Li, Y.; Xiong, Z. JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. Comput. Secur. 2021, 100, 102086.
19. Garg, S.; Peddoju, S.K.; Sarje, A.K. Network-based detection of Android malicious apps. Int. J. Inf. Secur. 2017, 16, 385–400.
20. Thangavelooa, R.; Jinga, W.W.; Lenga, C.K.; Abdullaha, J. DATDroid: Dynamic Analysis Technique in Android Malware Detection. Int. J. Adv. Sci. Eng. Inf. Technol. 2020, 10, 536–541.

21. Hasan, H.; Ladani, B.T.; Zamani, B. MEGDroid: A model-driven event generation framework for dynamic android malware analysis. Inf. Softw. Technol. 2021, 135, 106569.

ДОДАТКИ

ДОДАТОК А

Візуальне представлення розподілу “0” та “1” для кожної ознаки набору даних дозволів



ДОДАТОК Б

Візуальне представлення розподілу “0” та “1” для кожної ознаки набору даних сигнатур