Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _Комп'ютерної інженерії та управління_____
(повна назва)
Кафедра ___Автоматизації проектування обчислювальної техніки_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
_____Система аналізу та виявлення потенційно_____
_____зловмисних дозволів у Android-застосунках_____
(тема)

Виконав:
здобувач ___2_____ року навчання,
групи__СКСм-23-1_____
_____Білаш Д.А._____
(прізвище, ініціали)

Спеціальність 123 – Комп'ютерна інженерія
_____
(код і повна назва спеціальності)
Тип програми ___освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма __Спеціалізовані_____
_комп'ютерні системи_____
(повна назва освітньої програми)

Керівник _____доцент_Адамов О.С._
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ _Чумаченко С.В.__
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет_____Комп'ютерної інженерії та управління_____

Кафедра_____Автоматизації проектування обчислювальної техніки_____

Рівень вищої освіти_____другий (магістерський)_____

Спеціальність_____123 – Комп'ютерна інженерія_____
(код і повна назва)

Тип програми _____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____Спеціалізовані комп'ютерні системи_____
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«__02_»___вересня_ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві_Білашу Дмитру Анатолійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи Система аналізу та виявлення потенційно зловмисних дозволів у Android-застосунках
_____
затверджена наказом університету від __08__ _листопада__2024 р. № 1189

2. Термін подання здобувачем роботи до екзаменаційної комісії _31_ __січня__ 2025 р.

3. Вихідні дані до роботи _____
____Платформа Android Studio
____Android OS
____Дозволи ОС Android
____Платформа OpenAI
____Мова програмування Kotlin
_____
_____
_____

4. Перелік питань, що потрібно опрацювати в роботі _____
__1 Аналіз дозволів ОС Android
__2 Дослідження та аналіз методів пошуку зловмисних дозволів за допомогою машинного навчання
__3 Дослідження та аналіз інших методів пошуку зловмисних дозволів
__4 Розробка системи для аналізу та виявлення потенційно зловмисних

__дозволів у Android-застосунках_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)__18_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.09.2024 - 02.09.2024	
2	Аналіз проблемної галузі, постановка завдання	03.09.2024 - 10.09.2024	
3	Вибір інструментальних засобів та розробка	11.09.2024 - 18.09.2024	
4	Розробка програми	19.09.2024 - 10.10.2024	
5	Програмна реалізація	11.11.2024 – 25.11.2024	
6	Тестування розробленої системи	26.11.2024 - 03.12.2024	
7	Оформлення пояснювальної записки	04.12.2024 - 18.12.2024	
8	Оформлення графічного матеріалу	19.12.2024 - 05.01.2025	
9	Перевірка виконаного проекту керівником		
10	Захист роботи		

Дата видачі завдання _01_ _вересня_ 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____доцент_Адамов О.С._
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить 87 сторінок, 20 рисунків, 1 таблиця та 26 джерел за переліком посилань.

МОВА ПРОГРАМУВАННЯ. МАШИННЕ НАВЧАННЯ, АНАЛІЗ, БЕЗПЕКА, ДОЗВІЛ, ОС ANDROID, ЗАСТОСУНОК, ПРОГРАМА.

Метою кваліфікаційної роботи є дослідження методів ідентифікації небезпечних дозволів в мобільних застосунках на базі Android та розробка системи ідентифікації цих дозволів. Значна увага приділена застосуванню різних методів пошуку потенційно небезпечних дозволів, що дозволяє підвищити точність та ефективність ідентифікації шкідливого програмного забезпечення. Аналізується ряд технік та вже реалізованих способів отримання відповідної інформації.

В результаті роботи отримано загальні принципи побудови застосунку для пошуку небезпечного програмного забезпечення за допомогою наведених в дослідженні аналізу дозволів, що використовуються застосунком як в стані роботи так і під час завантаження застосунку. Також проведено безпосередню розробку системи для пошуку цих дозволів та їх груп. Дані цієї роботи можуть бути використані в подальшому аналізі шкідливого програмного забезпечення на ОС Android та написання комерційних систем для пошуку шкідливої поведінки безпосередньо при роботі пристрою.

Об’єктом дослідження є системи безпеки операційної системи Android.

Предметом дослідження є методи ідентифікації небезпечних дозволів в мобільних застосунках на базі Android.

ABSTRACT

The explanatory note to the qualification work contains 87 pages, 20 figures, 1 table and 26 sources in the list of references.

PROGRAMMING LANGUAGE. MACHINE LEARNING, ANALYSIS, SECURITY, PERMISSION, ANDROID OS, APP, PROGRAM.

The purpose of the qualification work is to study methods for identifying dangerous permissions in Android-based mobile applications and develop a system for identifying these permissions. Considerable attention is paid to the use of various methods for searching for potentially dangerous permissions, which allows to increase the accuracy and efficiency of identifying malicious software. A number of techniques and already implemented methods for obtaining relevant information are analyzed.

As a result of the work, general principles for building an application for searching for dangerous software were obtained using the analysis of permissions used by the application both in the running state and during application loading, as presented in the study. A system for searching for these permissions and their groups was also directly developed. The data from this work can be used in further analysis of malicious software on the Android OS and writing commercial systems for searching for malicious behavior directly during device operation.

The object of the research is the security systems of the Android operating system.

The subject of the research is methods for identifying dangerous permissions in Android-based mobile applications.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних;

ОС – операційна система;

ПЗ – програмне забезпечення;

ШІ – штучний інтелект;

API – інтерфейс програмування додатків (англ. Application Programming Interface);

GPT – трансформер генеративного попереднього навчання (англ. Generative Pre-trained Transformer);

HTTP – протокол передачі гіпертексту (англ. Hypertext Transfer Protocol);

IoT – інтернет речей (англ. Internet of Things);

ML – машинне навчання (англ. Machine Learning);

MLP – багатошаровий перцептрон (англ. Multilayer Perceptron);

ReLU – випрямлена лінійна одиниця (англ. Rectified Linear Unit);

RNN – повторна нейронна мережа (англ. Recurrent Neural Network);

SDK – комплект засобів розробки програмного забезпечення (англ. Software Development Kit);

SVM – опорна векторна машина (англ. Support Vector Machine);

URL – уніфікований покажчик ресурсу (англ. Uniform Resource Locator).

ВСТУП

У сучасному цифровому середовищі мобільні програми стали невід'ємною частиною повсякденного життя, а Android є домінуючою операційною системою, яка працює на мільярдах пристроїв у всьому світі. Мобільні програми стали важливими у нашому житті та роботі, оскільки багато послуг, якими ми користуємося, надаються через мобільні програми. Крім того, ОС Android стала домінуючою платформою не лише для мобільних телефонів і планшетів, а й для пристроїв Інтернету речей (IoT). У цьому контексті Google запустив Android Things, ОС Android для пристроїв IoT, де розробники отримують переваги від зрілого стеку Android для розробки програм IoT для тонких пристроїв. На відміну від персональних комп'ютерів (немобільних систем), мобільні пристрої оснащені складними датчиками, від камер і мікрофонів до гіроскопів і GPS. Ці датчики створюють цілий новий світ застосунків для кінцевих користувачів і генерують великі обсяги даних із надзвичайно конфіденційною інформацією.

Складність мобільних пристроїв і всюдисущість об'єктів IoT допомагають побудувати розумний світ, а також розкривають безпрецедентний потенціал для кіберзагроз. Такі загрози можуть бути вчинені зловмисниками, які можуть отримати доступ до конфіденційної інформації та ресурсів за допомогою шкідливих програм Android. У цьому контексті захист пристроїв Android від шкідливих програм є надзвичайно важливим. Це підвищує потребу в рішеннях безпеки для захисту користувачів від шкідливих програм, які використовують складність і конфіденційний вміст інтелектуальних пристроїв.

Програми Android працюють у рамках моделі на основі дозволів, де кожна програма має явно запитувати дозвіл на доступ до конфіденційних ресурсів або виконувати певні дії на пристрої за допомогою його фізичних компонентів. Ці дозволи призначені для захисту даних користувача та

цілісності системи, запобігаючи виконанню програмами неавторизованих операцій. Однак система дозволів може бути як благом, так і прокляттям. Хоча він забезпечує надійний механізм згоди користувача, він також є вектором атаки для зловмисників. Програми, які запитують непотрібні або занадто широкі дозволи, можуть використовувати їх для доступу до особистої інформації, відстеження активності користувачів або навіть віддаленого керування пристроєм.

Розвиток застосунків для Android забезпечив користувачам високу зручність і функціональність у різних сферах – від соціальних мереж і комунікацій до банківських операцій та контролю за здоров'ям. Однак, масове використання таких додатків породжує й серйозні проблеми безпеки, зокрема стосовно дозволів, які вони запитують, і можливого зловживання цими дозволами. Додатки Android функціонують за моделлю дозволів, де кожен додаток має запитати дозвіл на доступ до конфіденційних даних або певних функцій пристрою.

Ця система створена для захисту інформації користувачів і збереження цілісності системи, блокуючи несанкціоновані дії. Проте система дозволів є подвійним мечем: вона забезпечує механізм згоди, але одночасно відкриває шлях для атак. Додатки, які вимагають надмірні або непотрібні дозволи, можуть використовувати їх для збору особистих даних, моніторингу активності або навіть для віддаленого керування пристроєм.

Проблема зловживання дозволами в Android-застосунках – це не просто теорія: існують численні підтверджені випадки, коли на перший погляд безпечні програми використовували отримані дозволи для шкідливих цілей. Це включає збір конфіденційної інформації, відстеження місцезнаходження, відправлення та читання несанкціонованих SMS і навіть запуск сторонніх програм. Ці інциденти підкреслюють необхідність уважного аналізу запитів додатків на дозволи для зниження ризиків. Після зараження шкідливим ПЗ користувачі можуть втратити контроль над особистими даними, що ставить під загрозу їхню приватність, безпеку та цілісність.

Хоча компанія Google впроваджує покращені інструменти безпеки для захисту користувачів Android, зловмисні розробники іноді встигають розмістити компрометовані програми раніше, ніж їх видаляють. У ранніх версіях Android користувачі були змушені надавати всі дозволи одразу під час встановлення цієї програми на пристрій, що викликало занепокоєння.

У пізніших версіях операційної системи Android з'явилося динамічне управління дозволами, що дозволяє користувачам вибірково погоджуватися на доступ до даних під час роботи програми. Проект спрямований на розробку системи для аналізу дозволів у Android-програмах, щоб визначати й оцінювати потенційно шкідливу поведінку програм. Це передбачає не тільки визначення ризикованих дозволів, але й аналіз контексту їхнього використання. Наприклад, додаток для обміну фото може потребувати доступу до мережі для завантаження зображень і геолокації, але не повинен передавати дані рекламодавцям без дозволу користувача.

Детальний аналіз відокремить програми, які запитують дозволи на виконання реальних функцій, від тих, які використовують ці дозволи зі зловмисною метою. Щоб перевірити ефективність створеного інструменту, планується зібрати та дослідити дані про популярні Android-додатки, щоб виявити типові шаблони запитів дозволів. Загалом цей проект має на меті вирішити важливу проблему мобільної безпеки, пропонуючи практичний інструмент для аналізу та виявлення зловмисних дозволів у програмах Android [1-2].

Такий підхід буде корисним як для окремих користувачів і розробників, так і для наукового та професійного середовища в галузі мобільної безпеки, формуючи основу для подальших досліджень і розробок у цій важливій галузі. Результати дослідження будуть представлені у звіті, який міститиме цінну інформацію про поточні ризики безпеці додатків Android і запропонує рекомендації для користувачів, розробників і спеціалістів із мобільної безпеки.

1 ДОЗВОЛИ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

1.1 Дозволи програм

На мобільних пристроях зазвичай працюють різноманітні програми, які потенційно можуть містити вразливі місця, які можна використати, або навмисну зловмисну поведінку. Враховуючи ці ризики, Android ізолює застосунки в ізольованому програмному середовищі, ізолюючи їх одне від одного та від основного пристрою. Програми повинні отримати дозвіл, перш ніж отримати доступ до конфіденційних ресурсів або виконувати конфіденційні операції.

Модель безпеки Android базується головним чином на дозволах. Дозвіл – це обмеження доступу до частини коду або даних на пристрої. Обмеження накладено для захисту критично важливих даних і коду, які можуть бути використані з метою спотворення або пошкодження взаємодії з користувачем. Дозволи використовуються для надання або обмеження доступу програми до обмежених API та ресурсів. Наприклад, дозвіл Android INTERNET потрібен програмам для здійснення мережевого зв’язку, тому відкриття мережевого підключення обмежено дозволом INTERNET. Щоб отримати дозвіл, розробник вказує використання файлу маніфесту в оголошенні атрибута <uses-permission>. Поле android:name визначає назву дозволу. Дозволи оголошуються у файлі маніфесту програми (AndroidManifest.xml) і повинні бути надані користувачем під час встановлення (для старіших версій Android) або під час виконання (для новіших версій Android) (Рисунок 1.1).

Android вимагає, щоб програми запитували дозволи перед доступом до конфіденційних ресурсів або виконанням конфіденційних операцій. Програми мають декларувати кожен дозвіл у своєму файлі AndroidManifest.xml за допомогою запису <uses-permission>. Залежно від типу дозволу їм також може знадобитися попросити користувача надати дозвіл під час виконання

програми.

Сучасні версії Android також використовують механізми автоматичного обмеження дозволів для програм, які довгий час не використовувалися, зменшуючи ризик їх потенційного використання зловмисниками.

Ця система гарантує, що програми не можуть виконувати певні операції без явної згоди користувача. Ця модель розроблена для захисту даних користувача та цілісності системи шляхом обмеження доступу до конфіденційних ресурсів і функцій, таких як камера пристрою, контакти, дані про місцезнаходження та доступ до мережі.

Зловмисники можуть поширювати шкідливі програми, які запитують і використовують дозволи, або вони можуть використовувати вразливі місця в законних програмах, які мають дозволи.

```
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Рисунок 1.1 – Дозволи програми в файлі маніфесту

Якщо програма пропонує функції, які можуть вимагати доступу до обмежених даних або обмежених дій, визначте, чи можливо отримати інформацію або виконати дії без необхідності оголошувати дозволи. Існує можливість виконувати багато варіантів використання у своїй програмі, як-от фотографувати, призупиняти відтворення медіафайлів і відображати релевантну рекламу, без необхідності оголошувати будь-які дозволи.

Якщо буде вирішено, що програма повинна отримувати доступ до обмежених даних або виконувати обмежені дії, щоб виконати певний варіант використання, оголосите відповідні дозволи. Деякі дозволи, відомі як дозволи

під час встановлення, автоматично надаються під час встановлення програми. Інші дозволи, відомі як дозволи під час виконання, вимагають, щоб ваша програма пішла далі й запитала дозвіл під час виконання.

1.2 Типи дозволів

Дозволи Android класифікуються на основі їх впливу на конфіденційність користувача та безпеку системи. Основні категорії включають (рисунок 1.1).

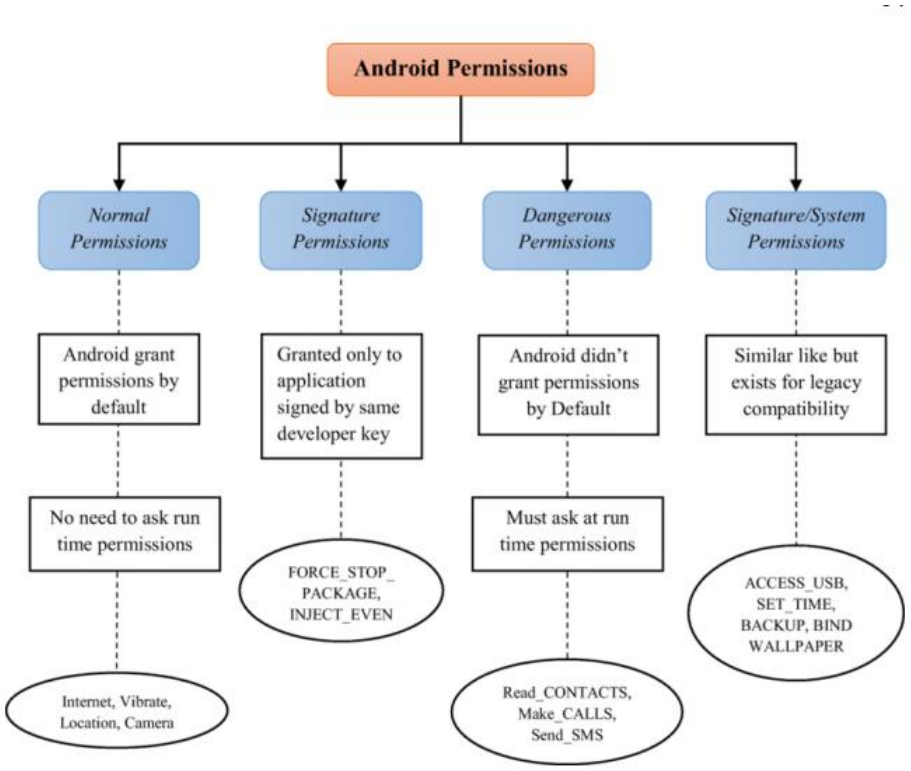

Рисунок 1.2 – Типи дозволів в програмах Android

Звичайні дозволи «Normal Permissions» – значення за замовчуванням. Дозвіл із низьким рівнем ризику, який надає програмі-запитувачу доступ до ізольованих функцій на рівні програми з мінімальним ризиком для інших програм, системи або користувача. Система автоматично надає цей тип дозволу програмі-запитувачу під час встановлення, не запитуючи явного

схвалення користувача, хоча користувач завжди має можливість переглянути ці дозволи перед встановленням [3-4]. Приклади включають встановлення програми як шпалери або доступ до Інтернету.

Дозволи на підпис «Signature Permissions» – дозвіл, який система надає тільки в тому випадку, якщо заявка, що запитує дозвіл, підписана тим самим сертифікатом, що і заявка, яка декларує дозвіл. Якщо сертифікати збігаються, система автоматично надає дозвіл, не повідомляючи користувача і не запитуючи його явного схвалення.

Небезпечні дозволи «Dangerous permission» – ці дозволи вважаються небезпечними, оскільки вони надають доступ до конфіденційних даних користувача або можуть впливати на збережені дані користувача. Користувачі повинні явно надати ці дозволи під час виконання, тому їх ще називають дозволами виконання.

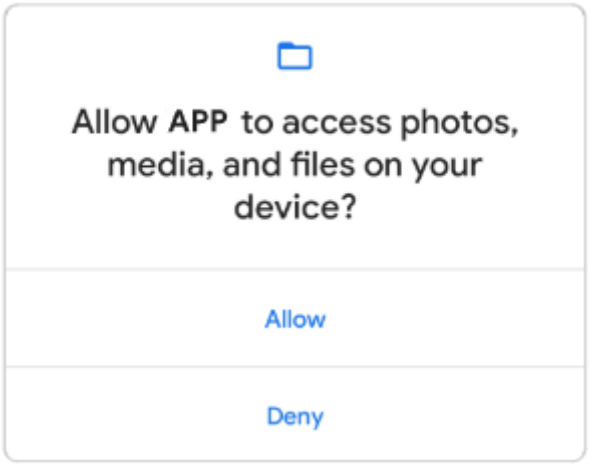

Рисунок 1.3 – Повідомлення про необхідність доступу до небезпечних дозволів

Потрібно запитати дозволи на час виконання у своїй програмі, перш ніж ви зможете отримати доступ до обмежених даних або виконувати обмежені дії. Навіть якщо ці дозволи були надані раніше- необхідно їх перевіряти і, якщо потрібно, явно запитувати їх перед кожним доступом.

Коли програма запитує дозвіл на виконання, система показує запит на дозвіл на виконання (рисунок 1.3).

Багато дозволів виконання мають доступ до приватних даних користувача, особливого типу обмежених даних, які містять потенційно конфіденційну інформацію. Приклади приватних даних користувача включають місцезнаходження та контактну інформацію. Мікрофон і камера забезпечують доступ до особливо конфіденційної інформації. Таким чином, система допомагає пояснити, чому ваша програма отримує доступ до цієї інформації.

Також, до версії Android 6.0 існував тип дозволу – «Install-time permissions». Дозволи під час встановлення надавали застосунку обмежений доступ до обмежених даних або дозволяють застосунку виконувати обмежені дії, які мінімально впливають на систему чи інші програми. Магазин додатків надає користувачеві повідомлення про дозвіл під час встановлення, коли він переглядає сторінку з інформацією про програму (рисунок)

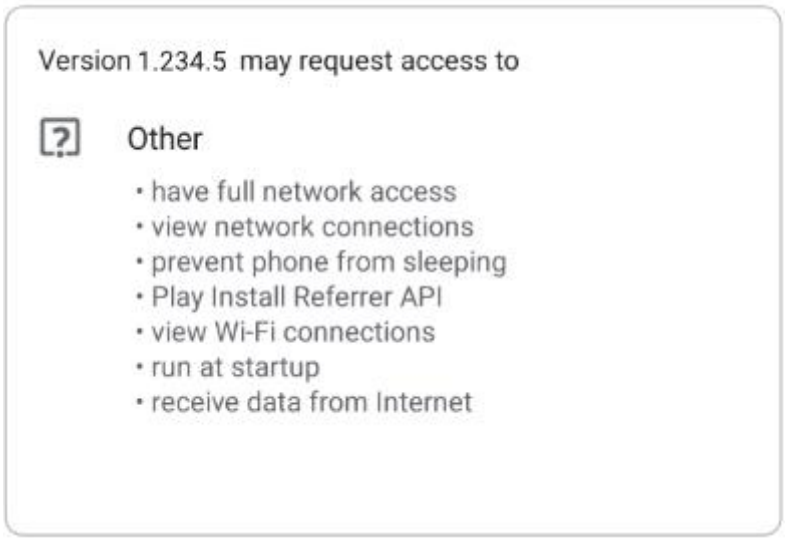

Рисунок 1.4 – Повідомлення продозволи під час встановлення

Система автоматично надає вашій програмі дозволи, коли користувач встановлює цей застосунок.

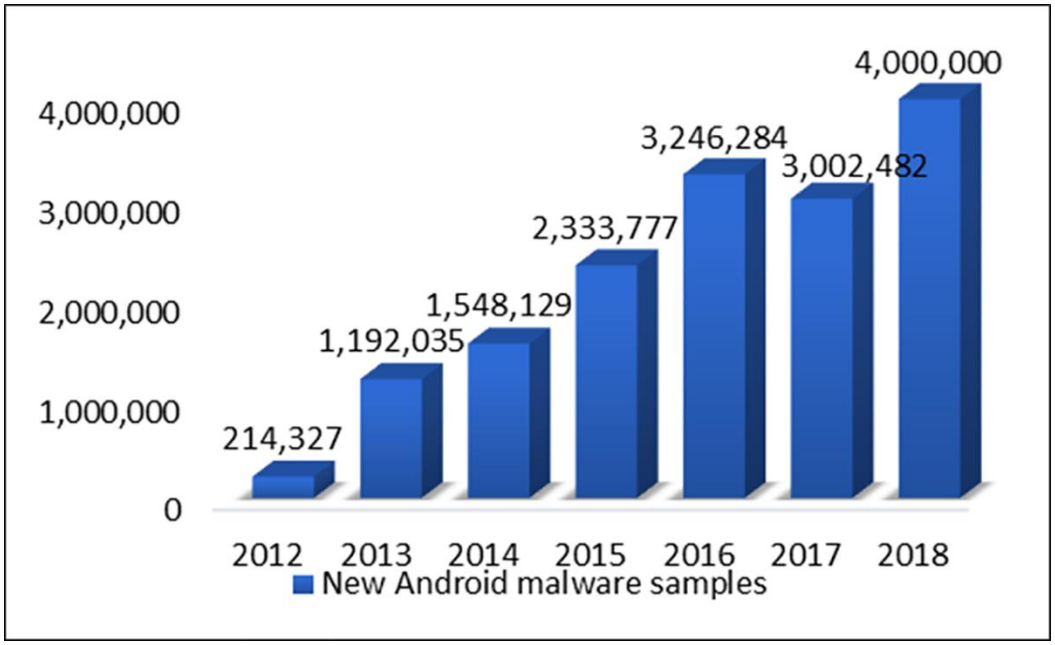

Рисунок 1.5 – Стрімке зростання зловмисних програм Anroid

Використання зловмисного програмного забезпечення Android піднялося до надзвичайного рівня. Показано різке зростання нових зразків шкідливого програмного забезпечення для Android з 2012 року по 2018 рік.

Приклади включають доступ до контактів, місцезнаходження або камери користувача. Дозволи на підпис: ці дозволи надаються, лише якщо програма, яка запитує, підписана тим самим сертифікатом, що й програма, яка визначила дозвіл. Це гарантує, що лише довірені програми від того самого розробника матимуть доступ до цих функцій.

Аналіз G DATA Cyber Defense показав, що загалом було створено 3,2 мільйона нових шкідливих програм виявлено у III кварталі 2018 року на відміну від III кварталу 2017 року, в якому виявлено 2 258 357 шкідливих програм (рисунок 1.5).

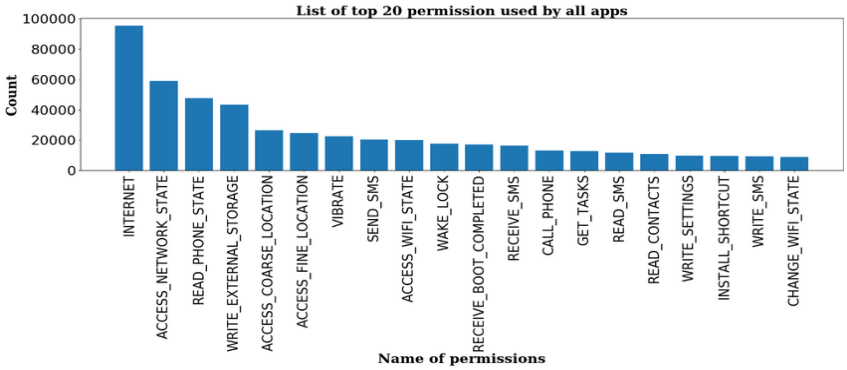

Рисунок 1.6 – Двадцять найбільш популярних дозволів за усіма застосунками

Найбільш популярними дозволами є дозволи доступу до Інтернету, запис в па’ять, вібрація, доступ до СМС та ін. (рисунок 3.3-3.4)

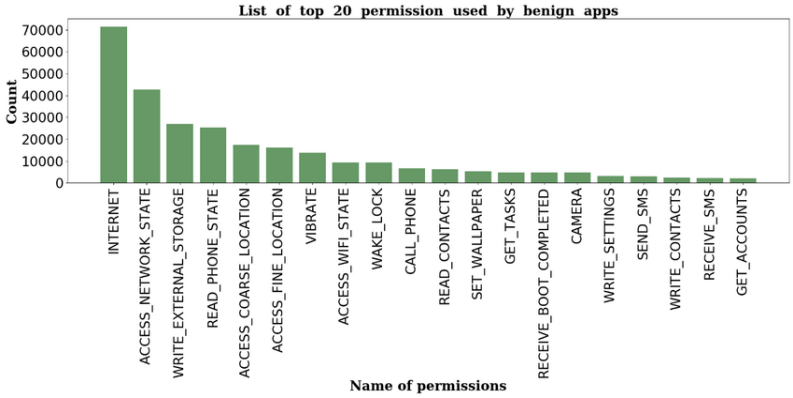

Рисунок 1.7 – Двадцять найбільш популярних дозволів за доброякісними застосунками застосунками

Декілька дозволів часто використовуються зловмисними програмами для порушення конфіденційності користувача та безпеки пристрою.

Деякі з дозволів, якими найчастіше користуються неправильно, включають:

INTERNET: дозвіл дозволяє доступ в Інтернет.

READ_CONTACTS і WRITE_CONTACTS: ці дозволи дозволяють програмі читати та змінювати контакти користувача. Шкідливі програми

можуть використовувати ці дозволи, щоб викрасти контактну інформацію для спаму або фішингу.

ACCESS_FINE_LOCATION і ACCESS_COARSE_LOCATION: ці дозволи надають доступ до даних про місцезнаходження пристрою. Шкідливі програми можуть відстежувати переміщення користувачів і створювати детальні профілі їхньої діяльності.

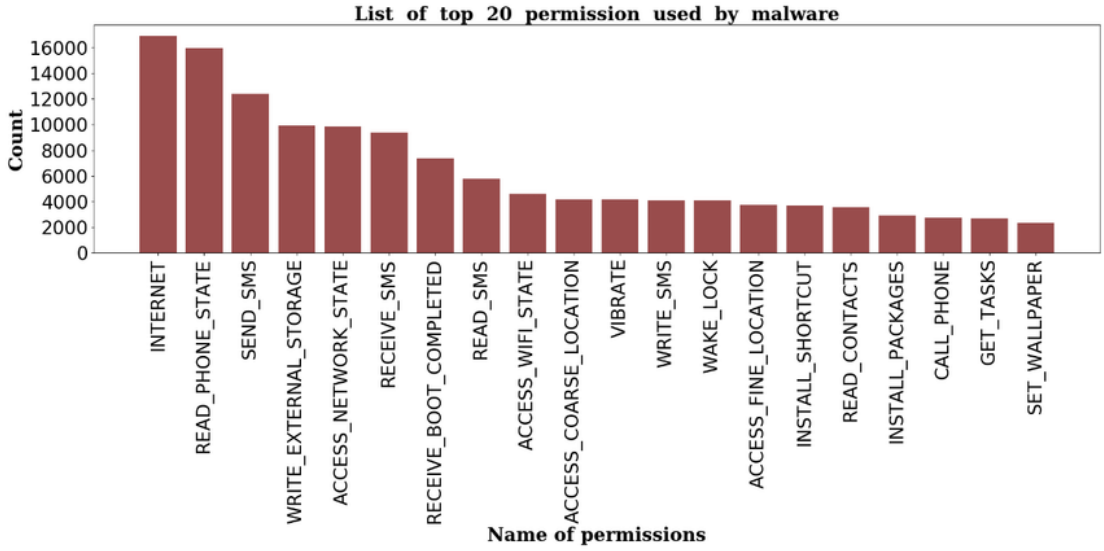

Рисунок 1.8 – Двадцять найбільш популярних дозволів використованих зловмисними застосунками

READ_SMS і SEND_SMS: ці дозволи дозволяють програмі читати та надсилати SMS-повідомлення. Шкідливі програми можуть використовувати ці дозволи для перехоплення повідомлень або надсилання неавторизовані повідомлення, що потенційно може призвести до фінансових втрат для користувача.

READ_PHONE_STATE: цей дозвіл надає доступ до інформації про стан телефону, включаючи номер телефону, інформацію про поточну мережу та статус поточних викликів. Шкідливі програми можуть використовувати цю інформацію для цілеспрямованих атак або маніпулювання мережевими

службами.

RECORD_AUDIO: цей дозвіл надає доступ програмі до запису звуку через мікрофон пристрою. Шкідливі програми можуть використовувати цей дозвіл для підслуховування розмов без відома користувача.

1.3 Тематичні дослідження зловмисних дозволів

Численні тематичні дослідження ілюструють реальні наслідки зловмисних дозволів. Наприклад, зловмисне програмне забезпечення «Judy», яке вразило мільйони пристроїв, використовувало надмірні дозволи для здійснення шахрайських кліків оголошень, приносячи дохід зловмисникам за рахунок приватності користувачів і використання даних.

Виявлене дослідниками Check Point у травні 2017 року зловмисне програмне забезпечення Judy є одним із найпоширеніших випадків зловживання дозволами в Google Play Store.

Ця кампанія зловмисного програмного забезпечення була активною протягом кількох років і, за оцінками, вразила понад 36,5 мільйонів користувачів.

Вектор зараження: зловмисне програмне забезпечення Judy було вбудовано в понад сорок, здавалося б, безпечних програм, розроблених корейською компанією. Ці програми виявилися законними і тому легко обійшли перевірки безпеки Google.

Після встановлення ці програми запитували низку дозволів, які здавалися необхідними для їх функціональності, але насправді були призначені для зловмисних цілей. Дозволи включали доступ до Інтернету, читання та запис у зовнішню пам’ять, а також можливість завантажувати та виконувати додатковий код.

Основною метою зловмисного програмного забезпечення Judy було генерування шахрайських кліків реклами, забезпечуючи прибуток для зловмисників. Після того, як користувач відкрив заражену програму,

зловмисне програмне забезпечення мовчки завантажувало додаткові корисні дані, які містили код JavaScript.

Потім цей код ініціював би заражений пристрій генерувати фальшиві кліки реклами на веб-сайтах, без відома користувача. Цей процес виснажував дані користувача та акумулятор, приносячи значний дохід від реклами для зловмисників. Зловмисне програмне забезпечення Judy виявило значні недоліки в процесі перевірки Google Play Store. Незважаючи на зусилля Google щодо підвищення безпеки, величезна кількість постраждалих користувачів підкреслила потенційний масштаб і вплив неправильного використання дозволів у програмах Android [5-6].

Сімейство шкідливих програм BankBot Сімейство зловмисних програм BankBot представляє серію банківських троянів, які розвивалися з часом і націлені на користувачів Android для викрадення конфіденційної фінансової інформації. Вперше виявлений на початку 2017 року. З тих пір були ідентифіковані різні версії BankBot, кожна з яких має все більш складні методи обходу заходів безпеки.

BankBot часто маскується під законні програми, такі як програми-ліхтарики або прості утиліти. Ці програми поширюються як через Google Play Store, так і через сторонні магазини програм.

Застосунки BankBot зазвичай запитують комбінацію дозволів, які свідчать про їхні зловмисні наміри, зокрема:

READ_SMS і RECEIVE_SMS для перехоплення SMS-повідомлень і запису кодів двофакторної автентифікації;

SYSTEM_ALERT_WINDOW: для накладання фішингових екранів на законні банківські програми;

READ_CONTACTS: для поширення зловмисного програмного забезпечення шляхом надсилання SMS-повідомлень контактам користувача. Після встановлення, BankBot отримує необхідні дозволи для моніторингу дій користувача та накладання фішингових екранів, коли користувач відкриває законну банківську програму.

Таблиця 1.1 – Напоширеніші дозволи, запитувані шкідливим ПЗ

Ім’я дозволу	Відсоток використання	Група дозволів	Класифікація на основі дозволів
INTERNET	98.8	Мережа	Небезпечно
ACCESS_NETWORK_STATE	94.4	Мережа	Нормально
READ_PHONE_STATE	82.7	Телефонний дзвінок	Небезпечно
WRITE_EXTERNAL_STORAGE	70.4	Сховище	Небезпечно
ACCESS_COARSE_LOCATION	56.2	Розташування	Небезпечно
ACCESS_WIFI_STATE	54.9	Мережа	Нормально
ACCESS_FINE_LOCATION	53.7	Розташування	Небезпечно
GET_ACCOUNTS	29.0	Облікові записи	Нормально
C2D_MESSAGE	24.7	-	Підпис
RECEIVE_BOOT_COMPLETED	24.7	Інформація про програму	Нормально
SYSTEM_ALERT_WINDOW	17.3	Відображення	Небезпечно
CALL_PHONE	14.2	Дзвінки на телефон	Небезпечно
CAMERA	13.0	Камера	Небезпечно
RECORD_AUDIO	12.3	Мікрофон	Небезпечно
READ_HISTORY_BOOKMARKS	11.7	Закладки	Небезпечно
SEND_SMS	11.7	Повідомлення	Небезпечно
READ_EXTERNAL_STORAGE	14.8	Зберігання	Нормально

Ці фішингові екрани імітують сторінки входу в різні банки, обманом змушуючи користувачів ввести свої облікові дані. Потім викрадена інформація надсилається зловмисникам, які можуть використовувати її для доступу до банківських рахунків жертв. BankBot мав значний вплив як на користувачів, так і на фінансові установи. Викравши облікові дані для входу, BankBot сприяв неавторизованим транзакціям і фінансовим крадіжкам, що

призвело до значних грошових втрат для користувачів.

Складність зловмисного програмного забезпечення, зокрема його здатність обходити двофакторну автентифікацію, створила значні проблеми для фахівців із безпеки [7-8].

Ці тематичні дослідження розкривають кілька важливих уявлень про природу та вплив зловмисних дозволів. Шкідливі програми часто представляють себе як законні, корисні програми, що ускладнює користувачам розпізнати їхню справжню природу. Це підкреслює потребу в більш ефективних механізмах виявлення та навчанні користувачів щодо дозволів програм. Багато програм вимагають більше дозволів, ніж необхідно, і використовують їх для різних зловмисних дій. Користувачам слід остерігатися програм, які запитують дозволи, не пов’язані з їхніми основними функціями.

Незважаючи на вжиті заходи безпеки, офіційні магазини програм, такі як Google Play, не захищені від шкідливих програм. Це підкреслює необхідність постійного вдосконалення процесів перевірки застосунків і впровадження більш просунутих протоколів безпеки. Важливість оновлень системи безпеки. Регулярні оновлення програм і операційної системи мають вирішальне значення для зменшення ризиків, створених шкідливими програмами. Оновлення безпеки можуть закрити вразливості, які використовує таке шкідливе програмне забезпечення.

Розробники повинні чітко обґрунтовувати використання дозволів і звести їх до мінімуму. Це допомагає не лише знизити ризик компрометації, а й сприяє довірі користувачів до програми.

На інших платформах, таких як iOS, дозволи для програм мають жорсткіші обмеження, а процес перевірки в App Store більш ретельний. Android може запозичити частину цих підходів для вдосконалення власної системи безпеки. Необхідність постійного вдосконалення процесів перевірки Незважаючи на вжиті заходи безпеки, офіційні магазини програм, такі як Google Play, не захищені від шкідливих програм

2 ПІДХОДИ ВИЯВЛЕННЯ ЗЛОВМИСНИХ ДОЗВОЛІВ ЗА ДОПОМОГОЮ МАШИННОГО НАВЧАННЯ

2.1 Загальні відомості

Під час аналізу зловмисного програмного забезпечення найчастіше використовуються методи машинного навчання, щоб класифікувати програми як нешкідливі, шкідливі або, в деяких випадках, можливо шкідливі. На глибшому рівні складніші методи можуть надавати дедалі точніші мітки, які ідентифікують програми як певний тип шкідливого програмного забезпечення, як-от шпигунське програмне забезпечення, банківські трояни тощо [9].

Враховуючи підтримку, яку надають автоматизовані методи, аналітики безпеки можуть зосередитися на дослідженні застосунків «сірої зони» або тих, які точно не класифікуються як хороше або шкідливе програмне забезпечення. Машинне навчання значно зменшує кількість застосунків, які аналітики мають переглядати вручну. Алгоритми машинного навчання можуть бути керованими або неконтрольованими.

Контрольовані алгоритми вимагають позначених наборів даних, тоді як неконтрольовані алгоритми вивчають закономірності, властиві даним. Алгоритми класифікації є найпоширенішим типом контрольованих алгоритмів, тоді як кластеризація та виявлення аномалій є типовими прикладами неконтрольованих алгоритмів. Кожен із них має власну мету в машинному навчанні, пов’язаному з безпекою.

Алгоритми класифікації, також звані класифікаторами, розглядають інформацію про сутність, таку як програма, зображення або обліковий запис користувача, і поміщають її в один або кілька класів. Наприклад, у випадку програми для Android може бути два класи інтересів: зловмисне та хороше програмне забезпечення. Але якщо необхідно класифікувати щось інше,

наприклад, облікові записи в Instagram, може бути набагато більше класів: діти молодше 18 років, молоді люди віком 18–40 років, люди середнього віку 41–65 років, і люди похилого віку для тих, кому 66 років і старше.

Інженери, які працюють над проблемою класифікації, визначають точну кількість категорій і значення кожної з них. Однією з проблем є те, що для створення точних моделей алгоритми класифікації часто потребують великої кількості мічених зразків (уже класифікованих зразків, на яких алгоритм може навчитися), які можуть бути не завжди доступними.

Алгоритми кластеризації беруть інформацію від кількох об’єктів і групують подібні зразки в кластери. Наприклад, зловмисники часто створюють кілька версій свого шкідливого програмного забезпечення з часом, шукаючи способи уникнути виявлення або додаючи нові функції. У таких випадках кластери можуть відповідати різним версіям одного сімейства шкідливих програм. Алгоритмам кластеризації потрібен спосіб вимірювання подібності або відстані між об’єктами, що знаходяться під спостереженням, і експерти домену відповідають за визначення того, як ця подібність обчислюватиметься на основі цілі кластеризації.

Хоча алгоритми кластеризації не вимагають мічених даних, створені ними кластери може бути важко інтерпретувати, якщо алгоритм не знає, що шукає аналітик. Зловмисне та корисне програмне забезпечення часто мають спільні SDK та бібліотеки, що може заплутати систему кластеризації, змушуючи її групувати зловмисне та хороше програмне забезпечення разом просто тому, що вони мають спільний SDK. Алгоритми виявлення аномалій або викидів намагаються ідентифікувати сутності, які суттєво відрізняються від майже всіх інших у даному наборі даних [10].

Наприклад, було докладено зусиль, щоб знайти шкідливі програми, перевіривши, чи їх поведінка суттєво відрізняється від норми. Однак проблемою для виявлення зловмисного програмного забезпечення Android є те, що більшість зловмисного програмного забезпечення працює в рамках моделі безпеки Android, запитуючи у мимовільних жертв дозволу на

виконання шкідливих дій. Чи програма, яка надсилає всі ваші повідомлення на віддалений сервер, є шпигунським програмним забезпеченням, чи це програма для резервного копіювання SMS. Алгоритмам виявлення аномалій може бути важко розрізнити ці два випадки.

Як виявилося, переважна більшість успішних спроб використання машинного навчання для виявлення зловмисного програмного забезпечення покладалися на алгоритми класифікації. Однак деякі методи використовують поєднання кластеризації та класифікації, щоб ідентифікувати сімейство, до якого належить зразок зловмисного програмного забезпечення.

Аналіз на основі машинного навчання пропонує надійний підхід до виявлення небезпечних дозволів у програмах Android. Використовуючи історичні дані та складні алгоритми, цей метод може ефективно ідентифікувати шаблони, що вказують на зловмисну поведінку. Описані вище кроки забезпечують комплексну структуру для розробки інструменту на основі машинного навчання для аналізу дозволів Android, від збору даних і вилучення функцій до навчання моделі та розгортання. Цей підхід не тільки підвищує точність виявлення, але й сприяє безпечнішій екосистемі мобільного зв’язку завдяки проактивному виявленню потенційних загроз [11-12].

Алгоритми машинного навчання зазвичай створюють хороші моделі, коли дані збалансовані, тобто вони мають достатньо порівняні відсотки вибірок у різних класах. І навпаки, деякі алгоритми можуть погано працювати, коли дані значно незбалансовані. Наприклад, припустімо, що ми намагаємося відрізнити шпигунське програмне забезпечення Android від хорошого програмного забезпечення. У більшості доступних сьогодні наборів даних про зловмисне програмне забезпечення кількість зразків шпигунського програмного забезпечення буде набагато меншою, ніж кількість зразків хорошого програмного забезпечення. Це може бути пов’язано з тим, що набір даних було зібрано, щоб відрізнити всі форми зловмисного програмного забезпечення (не лише шпигунського) від хорошого програмного забезпечення. Якщо класифікатор навчений відокремлювати шпигунське

програмне забезпечення від хорошого програмного забезпечення, кількість зразків шпигунського програмного забезпечення буде відносно невеликою порівняно з кількістю зразків хорошого програмного забезпечення. Такі дисбаланси в розмірах класів можуть серйозно вплинути на продуктивність алгоритмів класифікації.

2.2 Ідентифікація функцій програми

Більшість алгоритмів машинного навчання припускають, що з кожним об’єктом інтересу пов’язаний вектор ознак, який є впорядкованим списком значень, що належать до важливих властивостей об’єкта, що вивчається. У випадку аналізу зловмисного програмного забезпечення інтерес представляють самі програми. Вектор функцій включає в себе атрибути, отримані в результаті аналізу APK або програми, що виконується, і може бути створеним вручну або автоматично. З метою виявлення та класифікації зловмисного програмного забезпечення функції можуть стосуватися того, чи запитує програма певний дозвіл (наприклад, дозвіл на читання вхідних повідомлень), чи містить код зашифровані частини, чи намагається код підключитися до зовнішній сервер і так далі. Для кожного з цих запитань функція встановлюється на 1, якщо відповідь так, і на 0 в іншому випадку. Для кожного з цих запитань функція встановлюється на 1, якщо відповідь так, і на 0 в іншому випадку. Інші функції можуть мати не двійкові значення. Наприклад, у може бути функція, яка відповідає кількості викликів вихідного коду програми до даного пакета в Android API.

Методи машинного навчання використовують ці функції для ідентифікації та класифікації об’єктів. Лише вміст програми для Android надає, здавалося б, необмежену кількість потенційних функцій, але ми не повинні обмежувати набір функцій даними, що містяться в APK. Насправді є дивовижна цінність підключення функцій файлів APK до зовнішньої інформації. Наприклад, у випадку програми, яка підключається до певного

домену, можливо перетворити інформацію для цього домену на функції. Подібним чином, якщо програма підключається до певної IP-адреси, можна отримати інформацію про те, кому належить IP-адреса, центр обробки даних, який її обслуговує, країну, де розташований пов’язаний сервер, або навіть інформацію з самого сервера, наприклад операційну систему, яку він запускає, або інше програмне забезпечення, яке в ньому розміщено.

Щоб навести ще один приклад, якщо програма надсилає повідомлення на платний номер SMS, ми можемо визначити, якому оператору мобільного зв’язку належить цей номер і яку комерційну організацію він зареєстрував у партнерстві з оператором мобільного зв’язку. Розробники системи машинного навчання можуть включити цю інформацію як ознаки для характеристики певних сімейств шкідливого програмного забезпечення та розробників. Першим кроком у аналізі на основі машинного навчання є збір великого набору даних програм Android [13-14]. Цей набір даних має включати як безпечні, так і шкідливі програми, щоб переконатися, що модель може розрізняти їх. Попередня обробка даних передбачає підготовку вилучених функцій для навчання. Це включає обробку відсутніх значень, нормалізацію даних і поділ набору даних на навчальні та тестові набори. Масштабуйте функції до стандартного діапазону (наприклад, від 0 до 1), щоб гарантувати, що модель розглядає всі функції однаково.

Вибір правильної моделі машинного навчання має вирішальне значення для досягнення високої продуктивності. Зазвичай використовувані моделі для задач бінарної класифікації, як ця, включають:

-дерева рішень;

-випадкові ліси;

-машини опорних векторів;

-логістична регресія;

-нейронні мережі.

Кожна модель має свої сильні та слабкі сторони, тому може бути корисно поекспериментувати з кількома моделями, щоб знайти найкращу для

конкретного набору даних.

2.3 Дерево рішень

Дерево рішень – це універсальний контрольований алгоритм машинного навчання, який використовується як для завдань класифікації, так і для регресії. Його основна мета – змоделювати процес прийняття рішень за допомогою деревоподібної структури, яка нагадує блок-схему.

В основі дерева рішень лежить кореневий вузол, який є відправною точкою. Цей вузол представляє весь набір даних і відповідає за перше розбиття, поділ даних на основі ознаки, яка вважається найбільш важливою. Значимість визначається шляхом максимізації отримання інформації або мінімізації домішок, гарантуючи, що кореневий вузол створює найбільш ефективне початкове розділення. Коли дані проходять через дерево, вони стикаються з внутрішніми вузлами. Ці вузли служать точками прийняття рішень, які оцінюють конкретні умови, пов’язані з функціями. Наприклад, внутрішній вузол може поставити таке запитання, як «Чи запитується дозвіл READ_SMS?» Залежно від відповіді дані проходять одним із можливих шляхів, що ведуть від вузла.

Самі шляхи представлені гілками, які означають результати рішень, прийнятих на кожному вузлі. Гілка ілюструє двійковий або категоричний результат, наприклад, чи є умова у внутрішньому вузлі істинною чи хибною. Ці гілки спрямовують дані далі по дереву до кінцевої класифікації або результату регресії. У кінцевих точках цих гілок розташовані листові вузли, які забезпечують кінцевий результат процесу прийняття рішень. У завданнях класифікації ці кінцеві вузли містять окремі мітки класу, наприклад «Зловмисний» або «Безнебезпечний». У завданнях регресії вони містять безперервні значення, які представляють прогнозований результат. Вся структура дерева рішень, від кореня до листя, розроблена для систематичного розподілу даних і отримання точних прогнозів, які можна інтерпретувати.

Процес побудови дерева рішень передбачає систематичне розділення набору даних на основі умов ознак для створення ієрархічної структури, яка полегшує прогнозування. Дерево будується за допомогою серії кроків, спрямованих на максимізацію поділу між цільовими класами або мінімізацію помилок передбачення в регресії.

Початковим кроком у побудові дерева рішень є розбиття, коли набір даних ділиться на підмножини на основі умов, застосованих до однієї функції за раз. При кожному розділенні алгоритм вибирає функцію, яка оптимально розподіляє дані. Критерій відбору залежить від таких показників, як приріст інформації або зменшення дисперсії, залежно від завдання.

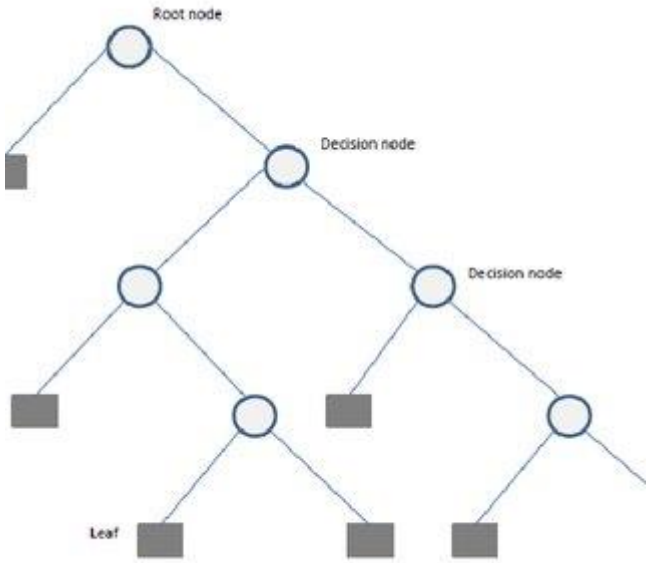

Рисунок 2.1 – Типове дерево рішень

Приріст інформації на основі енторії кількісно визначає зменшення ентропії, досягнуте шляхом поділу даних на певну функцію. Ентропія – це міра невизначеності або випадковості в наборі даних. Формула отримання інформації має наступний вигляд:

$$IG(D,A) = Entropy(D) - \sum_{u \in Values(A)} \left(\frac{|D_v|}{|D|}\right) \cdot Entropy(D_v), \quad (2.1)$$

де D – весь набір даних, A – функція, що оцінюється, Entropy (D) – ентропія набору даних, що розраховується за формулою 2.2, підмножини D_v, створені шляхом поділу на A, $|D_v|$ – Кількість зразків у підмножині D_v, а $|D|$ Загальна кількість зразків у D.

$$Entropy(D) = -\sum_{i=1}^{C} p_i \cdot \log_2(p_i), \quad (2.2)$$

де, C це кількість класів, а p_i це частка екземплярів у класі i.

Іншим способом розбиття є домішка Джіні, який вимірює ймовірність неправильної класифікації випадково вибраного елемента, якщо він був позначений відповідно до розподілу в підмножині. Він розраховується як:

$$G = 1 - \sum_{i=1}^{C} p_i^2, \quad (2.2)$$

де C – це кількість класів, а p_i це частка екземплярів у класі i.

Домішка Джіні – це вимірювання, яке використовується для побудови дерев рішень, щоб визначити, як функції набору даних мають розділяти вузли для формування дерева. Якщо точніше, домішка Джіні набору даних – це число від 0 до 0,5, яке вказує на ймовірність неправильної класифікації нових випадкових даних, якщо їм присвоєно випадкову мітку класу відповідно до розподілу класів у наборі даних.

Інтуїтивне пояснення випадковий вибір: уявіть, що існує певний мішок з кольоровими кульками, що представляють різні класи (наприклад, червоний і синій). Якщо ви вибираєте м’яч випадковим чином, шанс вибрати м’яч певного кольору відображає розподіл класів. Якщо в мішку здебільшого червоні кульки та кілька синіх, ваш шанс вибрати червону кульку високий, що вказує на низький рівень домішок. Якщо в мішку рівна кількість червоних і

синіх кульок, ваші шанси вибрати будь-який колір однакові, що вказує на високий вміст домішок. Коли дані розділяються на основі ознаки (наприклад, віку чи доходу), є потреба створити групи (дочірні вузли), де один клас є домінуючим. Хороший розкол мінімізує домішку Джіні в отриманих вузлах: Чистий вузол (де всі екземпляри належать до одного класу) матиме домішку Джіні 0. Якщо розділення призводить до вузлів, які все ще змішані з різними класами, домішка Джіні залишатиметься високою. Менші значення індексу Джіні вказують на більш чисті підмножини, які є кращими для розщеплення.

Алгоритм дерева рішень шукає поділ, який призводить до найбільш значного зменшення домішок Джіні, таким чином створюючи чіткіші відмінності між класами. Це дає точніші прогнози, коли дерево росте.

Таким чином, домішка Джіні є мірою того, наскільки змішані класи в наборі даних. Більш низькі значення домішок Джині вказують на більш однорідні групи, тоді як вищі значення свідчать про суміш класів. Зводячи до мінімуму домішки Джіні за допомогою стратегічних поділів, дерева рішень можуть ефективно класифікувати дані, що забезпечує кращу ефективність прогнозування.

Процес побудови дерева рішень використовує кілька критеріїв, щоб зупинити зростання дерева, пом’якшуючи ризик переобладнання та підтримуючи належний рівень складності моделі. Зростання припиняється на вузлі, якщо він містить виключно екземпляри, що належать до одного класу, що робить подальше розбиття непотрібним. Попередньо визначене обмеження глибини накладається на дерево, ефективно контролюючи його розмір і складність.

Алгоритм зупиняє розбиття вузла, якщо кількість екземплярів у ньому падає нижче вказаного порогу. Це заважає дереву вивчати надто специфічні шаблони з шумних або недостатніх даних. Процес припиняється, якщо приріст інформації в результаті потенційного поділу падає нижче попередньо визначеного порогу. Це гарантує, що розглядатимуться лише розбиття із суттєвими покращеннями здатності моделі класифікувати екземпляри.

2.4 Випадкові ліси

Випадкові ліси – це широко використовуваний алгоритм машинного навчання, який використовує потужність ансамблевого навчання для досягнення високої точності та надійності. Вони працюють шляхом побудови безлічі дерев рішень під час фази навчання, кожне з яких навчається на різній підмножині навчальних даних і враховує випадкову підмножину функцій при кожному розділенні.

Цей процес, відомий як пакетування (початкове агрегування), вносить різноманітність серед дерев, зменшуючи ризик переобладнання, яке може статися з окремими деревами рішень. Під час передбачення випадковий ліс агрегує передбачення з кожного з його складових дерев. Для завдань класифікації це зазвичай передбачає більшість голосів, де вибирається клас, передбачений більшістю дерев. У задачах регресії середнє значення прогнозів усіх дерев використовується для визначення кінцевого результату.

В основі випадкових лісів лежить дерево рішень, проста, але потужна модель, яка робить прогнози на основі серії запитань «якщо-інакше». Випадкові ліси створюють ансамбль із кількох дерев рішень, кожне з яких навчається на різній підмножині навчальних даних і враховує випадкову підмножину функцій при кожному розділенні. Процес навчання кожного дерева на випадковій підмножині даних відомий як пакетування (початкове агрегування). Крім того, у кожному вузлі дерева для розбиття розглядається лише випадкова підмножина ознак, що ще більше підвищує різноманітність.

Щоб зробити прогноз, випадковий ліс об’єднує прогнози всіх дерев, що входять до нього. Для класифікації це зазвичай робиться більшістю голосів, тоді як для регресії береться середнє значення прогнозів.

Єдине дерево рішень – це ієрархічна модель, яка робить прогнози на основі серії запитань «якщо-інакше». Він починається в кореневому вузлі та розгалужується на основі значень ознак. Дерева рішень схильні до переобладнання, особливо зі складними наборами даних, оскільки вони

можуть вловлювати шум і викиди. Однак їх відносно легко інтерпретувати та візуалізувати, що полегшує розуміння процесу прийняття рішень. Через надмірне оснащення їх продуктивність може не досягти високої точності на складних наборах даних.

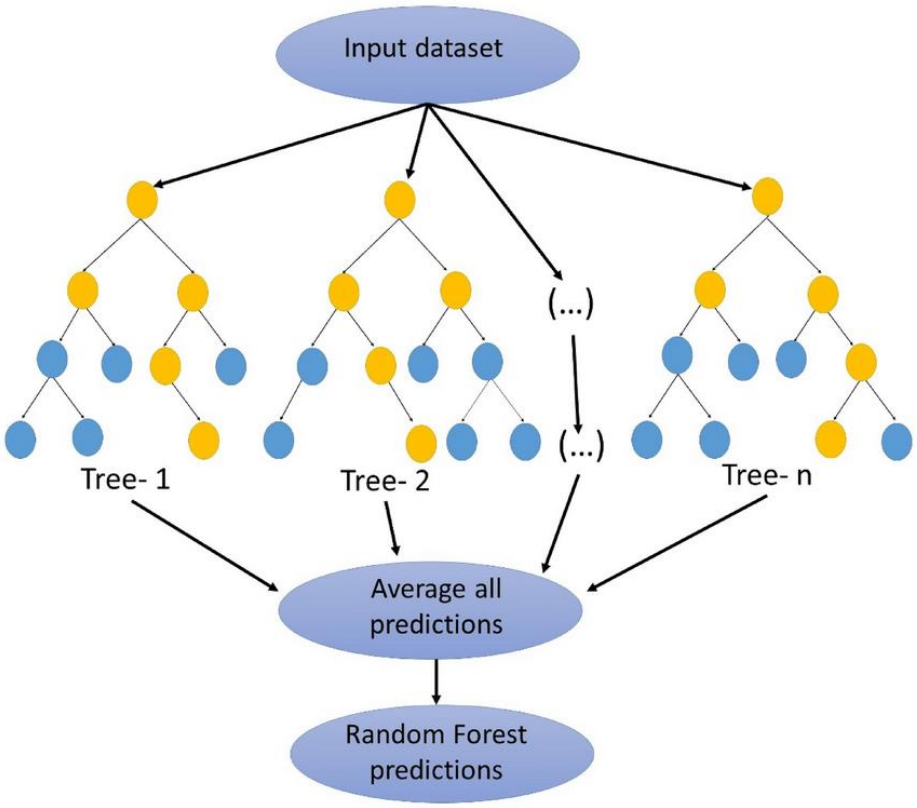

Рисунок 2.2 – Типовий вигляд випадкового лісу

Випадковий ліс – це набір кількох дерев рішень, кожне з яких навчено на різній підмножині даних і враховує випадкову підмножину ознак при кожному розділенні. Цей підхід, відомий як пакетування (початкове агрегування) і випадковість функцій, допомагає зменшити переобладнання. Отже, випадкові ліси часто досягають вищої точності, ніж окремі дерева рішень, особливо на складних наборах даних. Вони також забезпечують міру важливості ознак, вказуючи, які функції є найбільш впливовими в процесі прогнозування. Однак ансамблева природа випадкових лісів робить їх більш складними для інтерпретації, ніж окремі дерева рішень.

Ключові відмінності між деревами рішень і випадковими лісами включають їх тип моделі (окреме дерево чи ансамбль дерев), сприйнятливість до переобладнання, точність, можливість інтерпретації та здатність надавати важливість ознак. Вибираючи між деревом рішень і випадковим лісом, слід враховувати кілька факторів. Якщо розуміння процесу прийняття рішень має вирішальне значення, можна віддати перевагу дереву рішень.

Однак, якщо пріоритетом є досягнення найвищої можливої точності, випадковий ліс часто є кращим вибором. Для складних наборів даних з високою розмірністю випадкові ліси зазвичай працюють краще. Також важливо враховувати обчислювальні ресурси, оскільки навчання випадкових лісів може бути обчислювально дорожчим через наявність кількох дерев. Розуміючи ці ключові відмінності, можна прийняти обґрунтоване рішення щодо використання дерева рішень чи випадкового лісу для конкретного завдання машинного навчання.

2.5 Логістична регресія

Логістична регресія – це статистичний метод, який використовується для прогнозування ймовірності події. Це особливо корисно для завдань бінарної класифікації, де результатом може бути одна з двох можливих категорій (наприклад, так/ні, успіх/невдача, спам/не спам). На відміну від лінійної регресії, яка передбачає безперервне значення, логістична регресія моделює ймовірність події за допомогою сигмоїдної функції. Ця функція відображає будь-яке вхідне значення на значення від 0 до 1, що представляє ймовірність події. Основна ідея логістичної регресії полягає в моделюванні зв’язку між залежною змінною (результатом) і однією або кількома незалежними змінними (прогнозами). Це робиться шляхом оцінки ймовірності події, що відбудеться на основі значень незалежних змінних. Ця ймовірність потім використовується для прогнозування результату. Логістична регресія має кілька ключових переваг. Його відносно просто зрозуміти та

інтерпретувати, що робить його популярним вибором для багатьох програм. Він також ефективний з точки зору обчислень і може ефективно обробляти дані великої розмірності. Крім того, логістична регресія дає зрозуміти важливість різних змінних прогностики у визначенні результату.

Однак логістична регресія також має деякі обмеження. Він припускає лінійний зв'язок між предикторами та логарифмом шансів на результат, який не завжди відповідає дійсності в реальних сценаріях. Він також може бути чутливим до викидів і може не працювати належним чином, якщо дані дуже мультиколінеарні (тобто коли змінні предикторів сильно корельовані одна з одною).

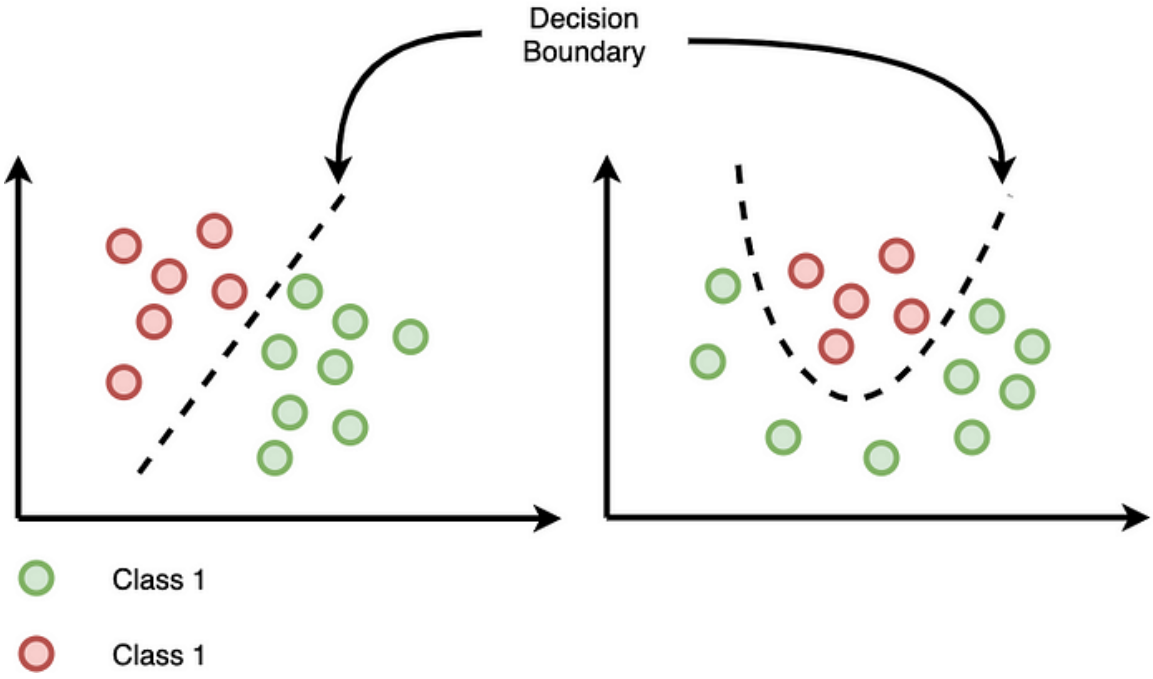

Рисунок 2.2 – Розділення двох класів за допомогою логістичної регресії

Незважаючи на ці обмеження, логістична регресія залишається широко використовуваним і цінним інструментом у різних галузях, включаючи медицину, фінанси, маркетинг і соціальні науки. Він часто використовується для таких завдань, як прогнозування захворювань, оцінка кредитного ризику, прогнозування відтоку клієнтів і аналіз настроїв.

Фундаментальне застосування логістичної регресії полягає у визначенні межі рішення для проблеми бінарної класифікації. Хоча базова лінія полягає у визначенні бінарної межі прийняття рішення, цей підхід можна дуже добре застосувати для сценаріїв із кількома класами класифікації або класифікацією з кількома класами.

На наведеній вище діаграмі пунктирну лінію можна визначити як межу прийняття рішення, оскільки ми спостерігатимемо екземпляри іншого класу з кожного боку межі. Наш намір у логістичній регресії полягав би в тому, щоб прийняти рішення про належну відповідність межі прийняття рішення, щоб ми могли передбачити, якому класу може відповідати новий набір функцій. Цікавим фактом про логістичну регресію є використання сигмоїдної функції як оцінки цільового класу. Давайте подивимося на інтуїцію цього рішення.

Для реалізації методу необхідно підготувати дані. Це передбачає вибір правильних функцій для створення моделі, обробки відсутніх значень і нормалізації даних. Якість підготовки даних значно вплине на продуктивність моделі.

Далі необхідно вибрати модель і оцінити її продуктивність. Щоб запобігти переобладнанню, часто використовується регулярізація. Також важливо експериментувати з різними значеннями гіперпараметрів, щоб досягти найкращих результатів. Ефективність моделі можна оцінити за допомогою таких показників, як точність, запам’ятовування, оцінка F1 і крива ROC.

Інтерпретація отриманої моделі дозволяє зрозуміти, як різні функції впливають на прогноз. Коефіцієнти моделі показують, як змінюється ймовірність належності до певного класу при зміні відповідної ознаки. Логістична регресія широко використовується в різних сферах, таких як прогнозування відтоку клієнтів, оцінка кредитного ризику, виявлення шахрайства та медична діагностика.

Для роботи з логістичною регресією можна використовувати різні програмні інструменти, наприклад Python, R і MATLAB. Кожна з них має

власні бібліотеки, які спрощують процес побудови та оцінки моделей. Однак, використовуючи логістичну регресію, важливо знати про можливі підводні камені. Наприклад, переобладнання, недооблаштування, неправильний вибір функцій та ігнорування мультиколінеарності можуть негативно вплинути на результати.

2.6 Support Vector Machine

Support Vector Machine (SVM) – це потужний керований алгоритм машинного навчання, який здатний виконувати завдання класифікації, регресії та навіть виявлення викидів. Однак його основне використання та популярність пов’язані з проблемами класифікації. SVM особливо ефективний у випадках, коли дані не є лінійно роздільними у своєму вихідному просторі функцій.

Цей метод прагне знайти оптимальну гіперплощину, яка найкраще розділяє класи в наборі даних. Термін «гіперплощина» відноситься до межі рішення, яка розділяє простір ознак на дві області, по одній для кожного класу. SVM досягають цього, знаходячи гіперплощину, яка максимізує запас між класами. Запас – це відстань між гіперплощиною та найближчими точками даних з обох класів, відомими як опорні вектори.

SVM вимагають позначених навчальних даних, де кожна точка даних представлена набором ознак і належить до одного з двох класів (для двійкової класифікації). SVM також може обробляти багатокласову класифікацію за допомогою таких стратегій, як «один проти решти» або «один проти одного». Якщо дані не можна лінійно відокремити у вихідному просторі функцій, SVM можуть відобразити дані у просторі більшої розмірності за допомогою функції ядра. Це перетворення дозволяє SVM методу знаходити лінійну гіперплощину в новому просторі, яка відповідає нелінійній межі рішення у вихідному просторі.

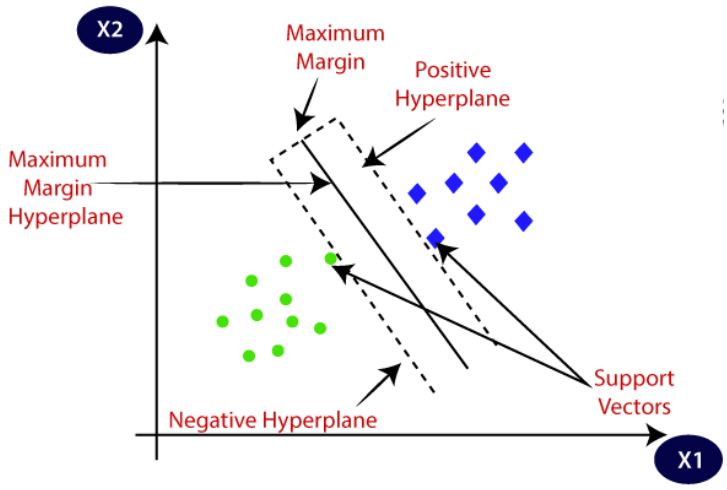

Рисунок 2.3 – Розділення двох класів за допомогою SVM

Метод спрямований на максимізацію запасу між опорними векторами двох класів. Запас визначається як відстань між гіперплощиною та найближчими точками даних з кожного класу. Він може ефективно обробляти нелінійні межі рішень, використовуючи функції ядра (наприклад, поліноміальна, радіальна базисна функція (RBF), сигмоїда), щоб відобразити вихідний простір ознак у простір вищої розмірності. Вибір функції ядра залежить від набору даних і складності необхідної межі рішення. Під час навчання SVM знаходить гіперплощину, яка не тільки розділяє дані, але й максимізує маржу. Ця максимізація запасу забезпечує краще узагальнення невидимих даних і покращує надійність класифікатора. Опорні вектори – це точки даних, які лежать найближче до межі прийняття рішень (гіперплощини). Вони відіграють вирішальну роль у визначенні оптимальної гіперплощини та використовуються для прогнозування нових точок даних.

SVM включає параметр регуляризації C, який контролює компроміс між максимізацією запасу та мінімізацією помилки класифікації даних навчання. Менший C дозволяє ширшу маржу (потенційно більше порушень маржі), тоді як більший C забезпечує суворішу маржу (потенційно менше порушень маржі). Після навчання SVM можуть передбачати мітку класу нових точок даних, оцінюючи, на яку сторону гіперплощини вони потрапляють. Параметр

"C" діє як штраф за неправильно класифіковані навчальні приклади. Менше значення C дозволяє отримати більший запас, навіть якщо це означає неправильну класифікацію кількох навчальних балів. Це спонукає модель віддавати пріоритет пошуку узагальненого рішення, потенційно ціною певної точності навчання. Більше значення C сильно карає неправильно класифіковані приклади навчання. Це змушує модель знаходити гіперплощину, яка мінімізує помилки навчання, навіть якщо це призводить до меншого запасу. Це може призвести до переобладнання, коли модель добре працює на навчальних даних, але погано на невидимих даних.

Малий C наголошується на пошуку більшого запасу, потенційно ціною певної точності навчання. Це може допомогти запобігти переобладнанню, особливо при роботі з шумними даними. Великий C в свою чергу, надає пріоритет мінімізації помилок навчання, що потенційно може призвести до меншого запасу та більшого ризику переобладнання, особливо коли дані зашумлені або межа рішення складна.

Оптимальне значення C залежить від конкретного набору даних і бажаного компромісу між максимізацією маржі та мінімізацією помилок навчання. Щоб знайти найкраще значення C для заданого набору даних, можна використовувати такі методи, як перехресна перевірка.

Параметр регуляризації C у SVM є критичним фактором, який суттєво впливає на поведінку моделі. Ретельно налаштовуючи цей параметр, практики можуть досягти найкращого балансу між ефективністю узагальнення та точністю навчання, зрештою покращуючи передбачувані можливості моделі.

2.7 Нейронна мережа та функції активації

Нейронна мережа – це програма чи модель машинного навчання, яка приймає рішення, подібні до людського мозку, використовуючи процеси, які імітують спільну роботу біологічних нейронів, щоб ідентифікувати явища, зважувати варіанти та робити висновки. Кожна нейронна мережа складається

з шарів вузлів або штучних нейронів – вхідного рівня, одного або кількох прихованих шарів і вихідного рівня. Кожен вузол підключений до інших і має власну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказаний поріг, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі. Нейронні мережі покладаються на навчальні дані, щоб навчатися та підвищувати свою точність з часом. Після їх точного налаштування на точність вони стають потужними інструментами в інформатиці та штучному інтелекті, що дозволяє класифікувати та кластеризувати дані з високою швидкістю. Завдання розпізнавання мовлення або розпізнавання зображень можуть тривати хвилини або години порівняно з ідентифікацією вручну експертами-людьми. Одним із найвідоміших прикладів нейронної мережі є пошуковий алгоритм Google. Нейронні мережі іноді називають штучними нейронними мережами або імітованими нейронними мережами. Вони є підмножиною машинного навчання та є основою моделей глибокого навчання.

Нейрон (персептрон) в штучній нейронній мережі необхідно розглядати як спрощену модель біологічного нейрона. Він приймає вхідні сигнали, обробляє їх і видає вихід. Він має певний набір вхідних значень (x1, x2, ..., x_n) та значень ваги, що характеризується важливістю вхідного значення (w1, w2, ..., w_n), зміщення, яке додає постійне значення, що дозволяє нейрону зміщувати вихідні дані та підвищувати гнучкість.

$$z = \sum_{i=1}^{n} \omega_i x_i + b, \qquad (2.3)$$

де z – це зважена сума, x_i – набір вхідних значень, ω_i – значення ваги набору, b – постійне значення гнучкості.

Зважена сума передається через функцію активації для того щоб отримати вихідне значення. Це вводить нелінійність. Без функцій активації нейронні мережі складалися б лише з лінійних операцій, таких як множення. Усі шари виконуватимуть лінійні перетворення вхідних даних, і жодних

нелінійностей не буде введено.

Більшість реальних даних є нелінійними. Наприклад, зв’язок між цінами на житло та розміром, доходом, покупками тощо є нелінійним. Якби нейронні мережі не мали функцій активації, вони б не змогли вивчити складні нелінійні шаблони, які існують у даних реального світу.

Функції активації дозволяють нейронним мережам вивчати ці нелінійні зв’язки, вводячи нелінійну поведінку за допомогою функцій активації. Це значно збільшує гнучкість і потужність нейронних мереж для моделювання складних даних із нюансами.

$$y = activation\_function(z), \qquad (2.4)$$

де y – вихідне значення, $activation\_function(z)$ – функція активації, z – зважена сума.

2.7.1 Сигмоїдна функція активації

Сигмоїдна функція є однією з класичних, яка перетворює вхідні дані на значення від 0 до 1 за допомогою формули:

Це схоже на обертання регулятора гучності на стереосистемі, де вхідні значення стискаються в м’який діапазон, що робить його ідеальним для завдань двійкової класифікації. Ця функція ідеально підходить, коли потрібні ймовірнісні виходи, наприклад, передбачити, чи є електронний лист спамом чи ні. Він перетворює будь-які вхідні дані в значення від 0 до 1, яке потім можна інтерпретувати як ймовірність. Функція є гладкою та диференційованою, що дозволяє ефективно використовувати її в методах градієнтної оптимізації.

$$\sigma(z) = \frac{1}{1+e^{-z}}, \qquad (2.5)$$

де, z – це зважена сума, $\sigma(z)$ – вихідний діапазон, що є обмеженим між 1 та 0, e – основа натурального логарифма ≈ 2.718.

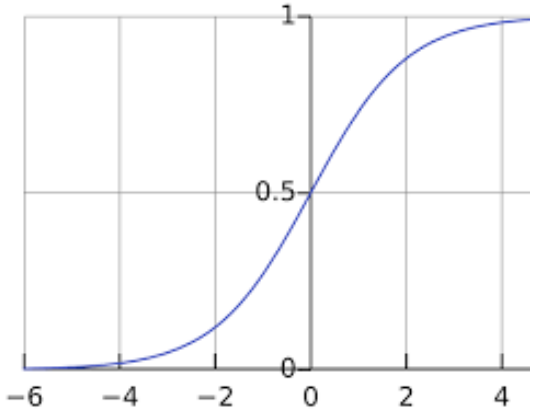

Рисунок 2.4 – Графік сигмоїдної функції активації

Проте, сигмовидна функція має свою погану сторону. Вона страждає від проблеми зникнення градієнта – коли вхідні дані занадто великі або занадто малі, функція вирівнюється, змушуючи градієнти наближатися до нуля під час зворотного поширення. Це сповільнює навчання, особливо в більш глибоких мережах. Незважаючи на цю проблему, функція все ще широко використовується у вихідному рівні для завдань двійкової класифікації. Прикладом можна вважати логістичну регресію – це вихід для двокласових задач, де потрібно вивести ймовірності.

2.7.2 Функція активації випрямлений лінійний блок (ReLU)

ReLU (випрямлений лінійний блок) є однією з найпопулярніших і широко використовуваних функцій активації в глибокому навчанні, особливо для прихованих рівнів у нейронних мережах. Його простота та ефективність роблять його чудовим вибором для навчання глибоких мереж.

$$Re\,LU(z) = max(0, z), \qquad (2.5)$$

де z – сума вхідних даних до нейрона, що розраховується за формулою 2.3.

Випрямлений лінійний блок виводить 0 для всіх від’ємних значень z, який ефективно «відключає» нейрони. Така розрідженість покращує ефективність обчислень і може допомогти запобігти переобладнанню. ReLU обчислювально недорогий порівняно з іншими функціями активації. Для цього потрібна лише проста операція встановлення порогу. Незважаючи на свою просту форму, випрямлена лінійна одиниця вводить нелінійність, дозволяючи мережі вивчати складні шаблони. випрямлена лінійна одиниця допомагає пом’якшити проблему зникнення градієнта, яка виникає з такими функціями активації, як сигмоїдна функція або функція гіперболічного тангенса. Градієнти залишаються значними для позитивних вхідних даних.

Іншими словами, якщо вхід позитивний, він проходить без змін; якщо він від’ємний, він дорівнює нулю. Жодного здавлювання, жодних вигадливих кривих – лише сира обчислювальна ефективність. Переваги Основна претензія ReLU на популярність полягає в тому, що він вирішує проблему зникаючого градієнта. Оскільки ReLU не збиває великі значення, градієнти не застряють, що дозволяє швидше навчатися. Крім того, випрямлена лінійна одиниця вводить розрідженість, тобто багато нейронів у мережі неактивні, що може підвищити ефективність обчислень.

Головною проблемою цієї функції є те, що нейрони можуть застрягти в неактивному стані, якщо їхні вхідні дані завжди негативні. Як тільки нейрон «вмирає», він більше ніколи не активується, що може погіршити продуктивність моделі.

Незважаючи на свой недолік, ReLU є функцією активації за замовчуванням для більшості сучасних архітектур глибокого навчання, від CNN (згорткових нейронних мереж) до MLP (багаторівневих перцептронів).

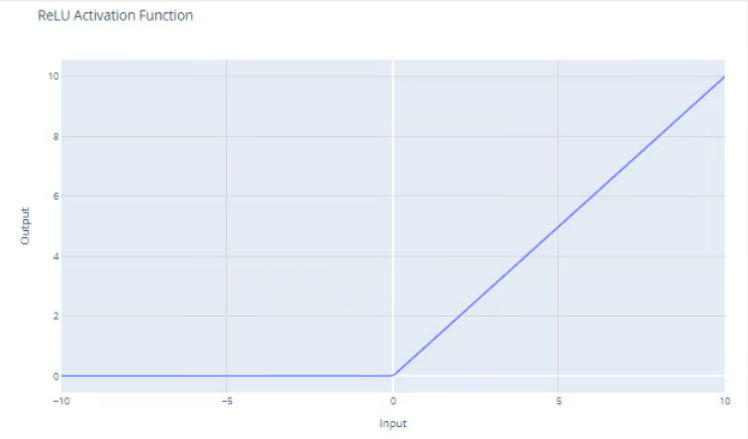

Рисунок 2.5 – Графік функції активації ReLU

Встановлюючи частину вихідних даних на 0, ReLU зменшує взаємозалежність параметрів, потенційно сприяючи кращому узагальненню. Якщо надто багато вхідних даних для нейрона негативні, нейрон виводить 0 і припиняє навчання (градієнт стає 0). Це може призвести до «мертвих нейронів», які постійно неактивні під час навчання. Необмежений вихід: Для великих позитивних значень ReLU може давати дуже високі результати, що може призвести до нестабільності навчання.

2.7.3 Функція активації гіперболічний тангенс

Іншою широкопоширьованою функцією активації є гіперболічний тангенс (tanh), схожий за своєю поведінкою з сигмоїдною функцією, але вона виводить значення від -1 до 1, що робить її відмінною від сигмоїдної функції, яка виводить значення від 0 до 1. Однією з основних переваг цієї функції є її нуль-центром, що означає, що його вихід симетричний навколо нуля. Ця симетрія може підвищити ефективність навчання алгоритмів, увімкнувши збалансовані градієнтні оновлення. Крім того, здатність обробляти як позитивні, так і негативні значення робить tanh більш ефективним вибором, коли вхідні дані містять обидва типи значень. Якщо сигмоїдна функція схожа на циферблат регулювання гучності, то tanh більше схожа на ваги балансу, які

підштовхують позитивні та негативні входи до сильніших крайнощів: -1 або +1. Функція активації має наступний вигляд:

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \quad (2.5)$$

де, $f(z)$ та $\tanh(z)$ – функція активації, z – це зважена сума, e – основа натурального логарифма ≈ 2.718.

Його вихідні дані зосереджені на нулі, тобто від'ємні значення більш виражені, що сприяє більш плавному переходу градієнтів під час навчання, особливо в глибших мережах.

Однак, як і його схожа функція, tanh не застрахована від проблеми зникнення градієнта. Коли вхідні дані стають занадто великими, функція насичується, уповільнюючи навчання. Оскільки тангенс має нульовий центр, його часто використовують у прихованих шарах, особливо в мережах, де потрібен більш збалансований результат. Це надійний вибір, якщо необхідно щось більш надійне, ніж сигмоїдна функція, але все ще просте.

Функція tanh також забезпечує сильніші градієнти порівняно з сигмоподібною функцією. Ця міцність пояснюється градієнтом tanh, що більший у своїх активних областях, що сприяє швидшому навчанню та конвергенції під час навчання. Однак, незважаючи на ці переваги, tanh не позбавлена проблем.

Властивість tanh з нульовим центром робить його особливо придатним для використання в прихованих шарах нейронних мереж. У поєднанні з вхідними даними, нормалізованими на нульове середнє значення, tanh може призвести до більш ефективного навчання. Ось чому його часто вибирають замість сигмоїдна у випадках, коли немає конкретної причини використовувати сигмоїдна. Однак для завдань, що вимагають імовірнісного виводу, сигмоїда є кращою для вихідних рівнів, оскільки вони видають значення в діапазоні, який узгоджується з інтерпретаціями ймовірності.

Як і сигмоїдна функція, вона страждає від проблеми зникнення

градієнта. Для дуже великих або дуже малих вхідних значень градієнти tanh наближаються до нуля, що може ускладнити ефективне навчання нейронних мереж у більш глибоких шарах. Ця проблема може значно сповільнити навчання та призвести до поганої конвергенції в глибоких архітектурах.

Незважаючи на свої обмеження, функція залишається сильним кандидатом для певних випадків використання, особливо в сценаріях, де швидша конвергенція або симетричні результати є перевагою. Однак сучасні нейронні мережі часто віддають перевагу функціям активації, таким як ReLU або його варіантам, для вирішення проблеми зникнення градієнта. Тим не менш, він продовжує залишатися цінним інструментом у наборі інструментів машинного навчання, особливо для завдань, які мають переваги завдяки його унікальним властивостям.

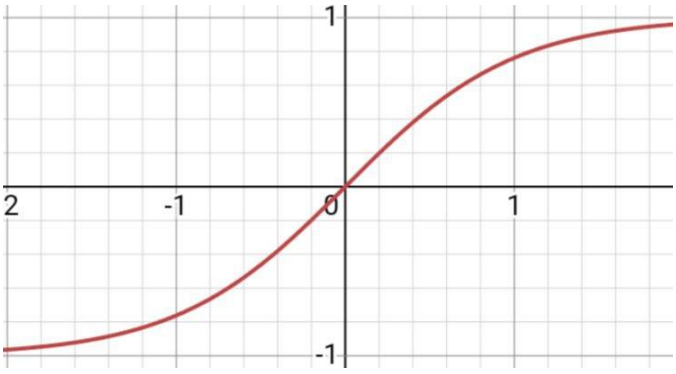

Рисунок 2.5 – Графік функції активації гіперболічний тангенс

2.7.4 Функція активації softmax

Для задач багатокласової класифікації на вихідному рівні нейронних мереж зазвичай використовується функція активації softmax. На відміну від інших функцій активації, які відображають вхідні дані в певних діапазонах, функція softmax перетворює вектор необроблених оцінок у розподіл ймовірностей. Це означає, що сума всіх вихідних ймовірностей дорівнює 1, що робить його ідеальним для завдань, де модель повинна передбачити

ймовірність кількох класів.

Математично функція softmax визначається як:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}}, \quad (2.6)$$

де, z_i – і-й елемент вхідних даних z, K– загальна кількість класів, кожен елемент у вихідному векторі представляє ймовірність певного класу, e – основа натурального логарифма ≈ 2.718.

Чисельник гарантує, що всі ймовірності додатні, тоді як знаменник нормалізує значення так, щоб їх сума дорівнювала 1. Цей крок нормалізації гарантує, що вихідні дані можна інтерпретувати як імовірності.

Функція softmax виділяє найважливіші значення у вхідному векторі, пригнічуючи менш важливі. Наприклад, якщо один із вхідних балів набагато більше, ніж інші, відповідний вихід softmax буде близьким до 1, тоді як інші будуть близькими до 0. Така поведінка є бажаною в задачах класифікації, оскільки вона гарантує, що модель віддає перевагу одному класу під час прогнозування.

Ключовою перевагою функції softmax є її здатність обробляти кілька класів одночасно. Наприклад, у завданні класифікації зображень із 10 можливими класами функція softmax виведе розподіл ймовірностей між усіма 10 класами, допомагаючи моделі призначити оцінки достовірності кожній можливій мітці.

Однак функція softmax не позбавлена обмежень. Функція може бути чутливою до великих вхідних значень, а його вихідні дані іноді можуть призвести до надмірної впевненості в прогнозах. Для вирішення цих проблем часто використовуються такі методи, як шкалювання температури або згладжування міток.

Таким чином, функція активації softmax є потужним інструментом для завдань багатокласової класифікації, забезпечуючи розподіл ймовірностей за

можливими результатами та дозволяючи моделям робити впевнені прогнози, які можна інтерпретувати.

2.8 Архітектури глибокого навчання

Структура багатошарового персептрона складається з кількох шарів взаємопов’язаних нейронів: Вхідний рівень отримує вихідні дані. Приховані шари – один або кілька шарів між вхідним і вихідним шарами, де відбуваються обчислення та перетворення. Рівень вихідних даних – це створює кінцевий результат (наприклад, передбачення класу, числові значення). Вхідні дані подаються на вхідний рівень. Зважена сума: кожен нейрон у шарі обчислює зважену суму своїх вхідних даних. Активація: кожен нейрон застосовує свою функцію активації до зваженої суми. Вихід кожного нейрона одного шару стає входом для нейронів наступного шару.

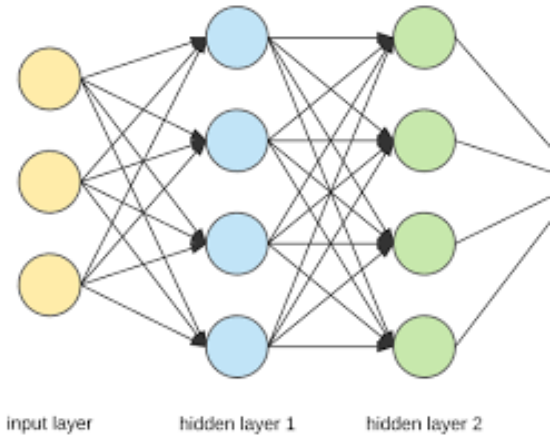

Рисунок 2.6 – Типова структура MLP архітектури

Цю архітектуру можна застосувати до проблеми аналізу дозволів Android. Він складається з вхідного шару, одного або кількох прихованих шарів і вихідного шару. Кожен шар повністю пов’язаний з наступним, при цьому нейрони кожного шару застосовують зважену суму, за якою слідує нелінійна функція активації. MLP можна використовувати для класифікації

дозволів, вивчаючи мічені набори даних, де дозволи позначені як безпечні або небезпечні. Навчання моделі передбачає подачу попередньо оброблених навчальних даних у вибраний алгоритм машинного навчання для вивчення шаблонів, пов’язаних із доброякісними та шкідливими програмами. Оцінка продуктивності моделі має важливе значення для того, щоб вона могла точно класифікувати нові застосунки. Загальні показники оцінювання включають моменти, коли модель навчена та оцінена, її можна розгорнути для класифікації нових програм. Розгорнута в межах іншого застосунку модель проаналізує дозволи нового файлу Android Package Kit і передбачить з певною ймовірністю, чи є програма доброякісною чи шкідливою.

Повторювані нейронні мережі (RNN), включаючи їх варіанти, такі як мережі довгострокової короткочасної пам’яті, добре підходять для послідовного аналізу даних. Дозволи Android часто мають певний порядок або шаблон залежно від функцій програми. RNN можна використовувати для вивчення цих шаблонів і виявлення аномалій, які можуть вказувати на зловмисну поведінку.

Повторювані нейронні мережі були введені для усунення обмежень традиційних нейронних мереж, коли справа доходить до обробки послідовних даних: він долає ці обмеження, вводячи повторювані з’єднання, які дозволяють інформації перетікати від одного часового кроку до наступного, створюючи певний контекст в аналізі.

Це періодичне з’єднання дозволяє RNN підтримувати внутрішню пам’ять, де вихідні дані кожного кроку передаються назад як вхідні дані для наступного кроку, дозволяючи мережі отримувати інформацію з попередніх кроків і використовувати її на поточному кроці, дозволяючи моделі вивчати часові залежності та обробка введення змінної довжини. Наприклад, якщо певна послідовність дозволів є незвичною або недоречною, Метод повторюваних нейроннихмереж може позначити її як потенційно небезпечну. LSTM особливо корисні, оскільки вони можуть обробляти довгострокові залежності та пом’якшувати проблему зникнення градієнта, роблячи їх

ефективними для більш розширених послідовностей запитів дозволів.

Автокодери – це моделі глибокого навчання без контролю, які можна використовувати для виявлення аномалій. Вони працюють шляхом кодування вхідних даних у представлення нижчої розмірності, а потім декодування їх назад до вихідного введення.

У контексті дозволів Android автокодер можна навчити на наборі даних, що містить запити на дозволи від надійних і доброякісних програм. Цей набір даних слугує «нормальною» поведінкою, яку автокодер навчається відтворювати. Модель по суті кодує дозволи в компактне представлення у своєму латентному просторі, а потім декодує їх, щоб передбачити вихідний набір дозволів. Процес навчання гарантує, що автокодувальник стане високоефективним у реконструкції запитів на дозвіл, які є поширеними та доброякісними. Після навчання автокодер можна використовувати для оцінки нових наборів дозволів із програм Android. Якщо новий набір дозволів значно відхиляється від вивченого представлення, модель не зможе його точно реконструювати. Ця недостатня точність свідчить про те, що новий набір дозволів може бути аномальним і потенційно небезпечним. Наприклад, якщо програма запитує комбінацію дозволів, яка є незвичайною або підозрілою, автокодер може позначити це як аномалію, вказуючи на те, що програма може виконувати шкідливі дії, як-от доступ до конфіденційних даних без законної причини.

Автокодери особливо корисні для виявлення загроз нульового дня або невідомих типів зловмисної поведінки, які можуть бути відсутні в навчальних даних. Оскільки вони не покладаються на дані з мітками (як моделі навчання під наглядом), вони можуть ідентифікувати шаблони або комбінації дозволів, які раніше не зустрічалися, що є важливою можливістю для боротьби з новими загрозами. Здатність виявляти незвичайні комбінації дозволів без попереднього знання того, що є зловмисною поведінкою, робить автокодувальники важливим інструментом у сфері безпеки та аналізу дозволів Android.

3 ІНШІ ПІДХОДИ ВИЯВЛЕННЯ ЗЛОВМИСНИХ ДОЗВОЛІВ

3.1 Статичний аналіз

Підходи статичного аналізу виконують швидку розбірку та декомпіляцію коду без необхідності виконання двійкового коду. Статичні методи залежать від статичних функцій, які витягуються з файлу APK, таких як запитувані дозволи, API, байт-коди, коди операцій [15-16].

Методи статичного аналізу є швидкими та охоплюють усі шляхи виконання аналізованої програми. Однак цей підхід підривається використанням різних методів перетворення коду. Термін «статичний аналіз коду» або просто «статичний аналіз» означає процес аналізу програми для виявлення її властивостей без її фактичного виконання. Ця стратегія відрізняється від динамічного аналізу, представленого в наступному розділі, де програма, що знаходиться під спостереженням, запускається для спостереження за її поведінкою під час виконання.

Статичний аналіз охоплює багато методів. Можливо розглядати це як набір підходів до міркувань про програми, включаючи читання програмного коду, а також автоматизовані стратегії, такі як аналіз потоку керування та аналіз потоку даних, спрямованих на розуміння порядку, у якому програма виконує інструкції, і того, як дані проходять через її змінні і пам'ять.

Існують також більш просунуті методи статичного аналізу, такі як перевірка моделі (використовується для підтвердження або спростування властивостей фрагмента коду) та абстрактна інтерпретація (спосіб дослідження станів програми за допомогою моделювання виконання), але ми не будемо розглядати ці просунуті методи у цій книзі. Наступні підрозділи містять деякі загальні вказівки для підвищення ефективності статичного аналізу [17].

Можливо розділити методи, засновані на статичному аналізі, на декілька категорій.

Аналіз на основі підпису: цей аналіз має справу з особливостями виділеного синтаксичного шаблону. Створюється унікальна сигнатура, що відповідає певній шкідливій програмі. Однак такий підпис не може працювати з новими варіантами існуючих відомих зловмисних програм. Крім того, базу даних підписів слід оновити для обробки нових варіантів. AndroSimilar було запропоновано для виявлення варіантів нульового дня відомого шкідливого програмного забезпечення. Це автоматизований метод виявлення зловмисного програмного забезпечення на основі сигнатур статистичних функцій.

Аналіз на основі ресурсів: файл маніфесту містить важливі метадані про компоненти, наприклад дії, служби, отримувачів тощо, а також необхідні дозволи. Існують методи, які були запропоновані для вилучення такої інформації та піддавання її аналізу.

Аналіз на основі дозволів базується на виявленні непотрібних запитів на дозвіл, які можуть призвести до зловмисної діяльності. Деякими дослідниками було запропоновано інструмент сертифікації, який визначає набір правил для виявлення зловмисного програмного забезпечення шляхом ідентифікації комбінацій запитуваних дозволів [18].

Існують підходи, які аналізують байт-код Dalvik, який є семантично насиченим і містить інформацію про тип, таку як класи, методи та інструкції. Крім того, таку інформацію можна використовувати для аналізу графів керування та потоку даних, які виявляють витік конфіденційності та зловживання послугами телефонії.

У професійному середовищі мета аналізу визначає, коли закінчите. Якщо метою є класифікувати програму як зловмисне програмне забезпечення та якомога швидше захистити користувачів, аналіз шкідливого програмного забезпечення може бути надзвичайно поверхневим. Наприклад, для фішингової програми можна переглядати зразок менше хвилини, знайти

докази того, що застосунок націлений на банківські програми, отримати це та зробити висновок [19-20].

Якщо мета полягає в тому, щоб задокументувати зловмисну функцію у звіті або якщо аналіз є реакцією на інцидент на сайті клієнта, можливо, необхідно піти глибше та витратити дні чи тижні на вибірку.

Досвід показує, що аналітики зловмисного програмного забезпечення повинні також рухатися швидко, щоб підтвердити, що програма є зловмисним програмним забезпеченням, щоб можна було вжити заходів, щоб її вимкнути, або проводити більш детальний аналіз, щоб поглиблено дослідити її та попутно дізнатися, як покращити інструменти та процеси.

3.2 Динамічний аналіз

Динамічні методи використовують функції, отримані від виконання програми. Вони більш стійкі до обфускацій коду, ніж методи статичного аналізу. Однак такі методи спричиняють додаткові витрати з точки зору обробки та пам’яті для запуску програми. Крім того, методи захисту від емуляції, такі як виявлення пісочниці та затримка виконання зловмисного програмного забезпечення, можуть уникнути методів динамічного аналізу.
Є два види динамічного аналізу - на основі використання ресурсів та на основі зловмисної поведінки.

На основі використання ресурсів деякі шкідливі програми можуть спричиняти атаки типу «відмова в обслуговуванні» (DoS), надмірно використовуючи обмежені апаратні ресурси. Низка параметрів, таких як використання процесора, статистика використання пам’яті, модель мережевого трафіку, використання акумулятора та системні виклики для безпечних і шкідливих програм, збираються з підсистеми Android. Потім використовуються методи автоматичного аналізу разом із технікою машинного навчання [21].

На основі зловмисної поведінки: це пов’язано з ненормальною

поведінкою, такою як витік конфіденційних даних і надсилання SMS/електронної пошти.

Моніторинг поведінки програми передбачає відстеження різних аспектів її операцій під час виконання. Це включає виклики API, потоки даних, взаємодію із системними ресурсами та мережеву діяльність. Інструменти та методи моніторингу можуть фіксувати детальні журнали цих взаємодій. Відстежуйте виклики API, зроблені програмою, щоб виявити будь-які, які можуть отримати доступ до конфіденційних даних або виконати підозрілі дії.

Аналізуйте потік даних, щоб виявити будь-який неавторизований збір або передачу даних. Відстежуйте взаємодію із системними ресурсами, такими як файлова система, мережа та датчики.

Приклади інструментів для аналізу:

- DroidBox – це інструмент динамічного аналізу, призначений для аналізу поведінки програм Android, він надає докладні журнали та візуалізацію поведінки програми під час виконання, що допомагає ідентифікувати зловмисну діяльність і вразливі місця безпеки;

- TaintDroid – це система моніторингу конфіденційності в реальному часі для ОС Android, яка відстежує потік конфіденційних даних через програми сторонніх розробників, він використовує динамічний аналіз забруднення, щоб відстежувати, як програми обробляють конфіденційну інформацію, і гарантує, що дані не витікають без згоди користувача.

Ці інструменти відрізняються за своїм підходом і можливостями: від статичного аналізу, зосередженого на перевірці коду без виконання, до динамічного аналізу, який передбачає запуск програм і моніторинг їх поведінки в режимі реального часу. Залежно від ваших конкретних потреб (тестування безпеки, оптимізація продуктивності, аналіз конфіденційності), інші інструменти можуть бути більш підходящими. Інтеграція кількох інструментів у робочий процес аналізу може забезпечити більш повне розуміння безпеки та конфіденційності програми Android.

Виявлення аномалій передбачає виявлення відхилень від нормальних

моделей поведінки. Цього можна досягти за допомогою евристик або моделей машинного навчання, навчених на наборах даних про звичайну та шкідливу поведінку. Аномалії можуть свідчити про зловмисну діяльність, як-от несанкціонований доступ до даних або несподіване мережеве спілкування.

Виявлення на основі евристики мають на меті створення або використання правил та шаблонів, які характеризують нормальну та зловмисну поведінку. Виявлення на основі машинного навчання – це навчайте моделі за допомогою наборів даних доброякісної та шкідливої поведінки програм [22].

3.3 Підходи до гібридного аналізу

Гібридні методи використовують як статичні, так і динамічні характеристики. У попередніх категоріях запропоновані системи зосереджені головним чином на завданнях виявлення, у яких відокремлюється зловмисне програмне забезпечення та безпечні програми. У цій категорії запропоновані системи зосереджені на завданні атрибуції сімейства зловмисних програм, окрім завдання виявлення, як мети аналізу. Атрибуція сімейства зловмисного програмного забезпечення має на меті віднести зловмисне програмне забезпечення до його фактичного сімейства.

Щоб захистити системи Android, деякі методи зосереджені на виявленні варіантів відомих сімейств. Інші пропозиції застосовують підхід до неконтрольованого навчання для пошуку сімейства подібних програм. Ці пропозиції припускають, що дві або кілька програм із подібним кодом, ймовірно, належать до однієї сім’ї шкідливих програм.

Таким чином, вони перевіряють, чи програми використовують подібний шкідливий код (тобто виявлення сімейства зловмисного програмного забезпечення), або вони перевіряють повторне використання коду тієї самої оригінальної програми (тобто виявлення повторного використання коду).

У програмному аналізі та реверсивному проектуванні термін динамічний

аналіз або динамічний аналіз коду стосується застосування методів аналізу, які розкривають властивості програми, що знаходиться під спостереженням, шляхом виконання її коду. Це контрастує зі статичним аналізом, метою якого є виявлення властивостей програми шляхом аналізу її коду та структури без її виконання.

Звичайно, запуску програми недостатньо, щоб зрозуміти, що вона робить. Динамічний аналіз включає цілий арсенал інструментів, які відстежують програму та взаємодіють із нею, включаючи налагоджувачі та програмне забезпечення для перехоплення викликів API, створення дампа пам’яті або перевірки мережевого трафіку.

Інші інструменти можуть реалізовувати способи взаємодії з графічним інтерфейсом програми або автоматично перевіряти властивості безпеки програми на потенційні вразливості. Коли ці інструменти працюють разом, вони повинні створити картину того, як програма взаємодіє з пристроєм. Чим більше інструментів використовується для моніторингу пристрою, тим повнішим стає ваше розуміння програми. Однак розгортання та підтримка всіх цих інструментів може зайняти значний час і гроші [23].

3.4 Динамічний проти статичного аналізу

Динамічний і статичний аналіз доповнюють одне одного. Щоб отримати повне уявлення про функціональність програми, потрібно використовувати обидві форми аналізу, і всі професійні програми аналізу зловмисного програмного забезпечення роблять це. Прикладом різких відмінностей між статичним і динамічним аналізом є кількість зусиль, необхідних для їх налаштування. Для статичного аналізу вам потрібно лише завантажити програму в jadx (jadx – це інструменти командного рядка та графічного інтерфейсу для створення вихідного коду Java із файлів Android Dex і APK).

З іншого боку, динамічний аналіз вимагає спочатку налаштування пристрою (реального чи віртуального), який може виконувати програму, а

потім переконатися, що є можливість перехоплювати та реєструвати системні виклики, мережевий трафік, зміни файлової системи та будь-які інші модифікації пристрою, які застосунок може зробити. Нарешті, необхідно запустити програму та взаємодіяти з нею в надії запустити шкідливі функції. Це може бути доволі складно, оскільки програми зловмисного програмного забезпечення часто використовують безліч трюків для захисту від аналізу та відмовляються запускатися, коли вважають, що під час аналізу в тестовому середовищі дослідника безпеки.

Коли ця перешкода буде подолана застосунок буде готовий до тестування на запущеному емуляторі, однак динамічний аналіз може досягти прогресу набагато швидше, ніж за допомогою статичного аналізу, оскільки є можливість спостерігати, що робить програма, і намагатися змусити її виконання в будь-якому напрямку, який цікавить.

Система аналізу реєструватиме всі конфіденційні виклики API, мережевий трафік і інформацію про навколишнє середовище та помістить деталі у звіт, який ви зможете вивчити пізніше. Немає потреби довго переглядати весь код, як зі статичним аналізом.

Ще одне місце, де динамічний і статичний аналіз доповнюють одне одного з точки зору покриття коду, міри того, скільки коду може проаналізувати техніка аналізу. У статичному аналізі весь код програми доступний для аналізу. У динамічному аналізі можна враховувати лише виконаний код. Різниця між обидвома може бути величезною. Навіть найкращий динамічний аналіз програми не може виконати більше 5-10 відсотків коду програми. Решта від 90 до 95 відсотків залишаються таємницею і можуть бути розкриті лише шляхом статичного аналізу [24].

Статичний аналіз забезпечує основу для швидкого виявлення відомих проблем, тоді як динамічний аналіз виявляє приховану поведінку, яка може проявлятися лише під час виконання.

Файл AndroidManifest.xml є важливим компонентом будь-якої програми Android. Він містить важливу інформацію про програму, включаючи назву її

пакета, компоненти (такі як дії, служби, приймачі трансляції та постачальники вмісту), а також дозволи, які вона запитує. Спочатку нелбхідно витягнути файл маніфесту. Першим кроком у статичному аналізі є парсинг файлу AndroidManifest.xml із файлу APK (Android Package Kit). Це можна зробити за допомогою таких інструментів, як Androguard або APKTool, які декомпілюють APK і роблять його вміст доступним.

Після вилучення файл маніфесту потрібно прочитати та проаналізувати. Це передбачає перевірку структури XML для ідентифікації різних тегів і атрибутів, зокрема тегів <uses-permission>, які оголошують дозволи, запитувані програмою.

Після вилучення та аналізу дозволів із файлу маніфесту наступним кроком є класифікація цих дозволів. Дозволи Android класифікуються на основі їх впливу на конфіденційність користувачів і безпеку системи. Оскільки розрізняється декілька видів дозволів їх необхідно розділити на певні категорії.

Щоб покращити виявлення потенційно шкідливих програм, ідентифіковані дозволи можна порівняти з базою даних відомих шкідливих дозволів. Цей крок передбачає порівняння дозволів, запитуваних програмою, з тими, які зазвичай пов’язані зі зловмисною поведінкою. Це порівняння допомагає визначити дозволи, які можуть становити значні ризики для безпеки. Перш за все потрібне створення або використання існуючої бази даних, де необхідно скласти список дозволів, які часто використовуються зловмисним програмним забезпеченням. Цей список можна отримати з досліджень безпеки, звітів і баз даних, які підтримують організації з кібербезпеки.

3.5 Емулятор пристрою Android та методи антианалізу

Першим інструментом, який необхідно налаштувати для динамічного аналізу, є середовище виконання, у якому виконується програма. Можна

використовувати або справжній пристрій Android, або емулятор. Далі потрібно вибрати тип і конфігурацію пристрою та, якщо використовується емулятор, чи використовувати емулятор за замовчуванням, який постачається з Android SDK, чи сторонній.

Використання емулятора дешеве і дозволяє швидко скинути аналіз, якщо щось піде не так. З іншого боку, більшість зловмисного програмного забезпечення Android намагається визначити, чи працює воно в емуляторі, і поводиться по-іншому, якщо вважає, що його аналізують, що може призвести до втрати великої кількості часу. Якщо ви використовуєте справжній пристрій, ви пройдете ці перевірки, але потрібно розуміти, що під час виконання роботи на власному пристрої його може бути пошкоджено і просто скинути конфігурацію, як в прикладі з емулятором, може не вирішити проблему.

Сотні методів динамічного антианалізу, опублікованих в Інтернеті, намагаються виявити емулятори. Ці техніки варіюються від досить простих до досить складних. Наприклад, стандартний емулятор Android не намагається приховати себе. Скоріше він транслює, що це емулятор, за допомогою властивостей системи, таких як модель пристрою та емульований мобільний оператор (налаштований на Android).

Зловмисне програмне забезпечення може легко виявити, що воно працює на цьому емуляторі, перевіривши ці властивості системи або подивившись на архітектуру ЦП пристрою, на якому воно працює. Справжніх Android-пристроїв x86 майже не існує, тому щоразу, коли програма працює на процесорі x86, вона, ймовірно, на емульованому пристрої.

Але програми не повинні покладатися на ці значення за замовчуванням або властивості апаратного забезпечення для виявлення емуляторів. Деякі можуть перевірити, чи встановлено на пристрої такі популярні програми, як Facebook. Facebook можна знайти майже на всіх реальних пристроях, але рідко на емуляторах. Інші додатки перевіряють, чи SMS і історія веб-перегляду користувача схожі на дані справжнього користувача чи порожні, як у нещодавно створеному емуляторі [25-26].

У ще більш екстремальних випадках програми можуть запускати код для оцінки часових властивостей доступу до пам’яті. Емульована пам'ять поводиться інакше на апаратному рівні, ніж справжня фізична пам'ять.

Багато загальнодоступних методів антианалізу націлені на інструменти динамічного аналізу, які часто встановлюються на емуляторах. Зазвичай ці методи намагаються виявити файли, процеси або інші властивості системи, які присутні лише тоді, коли встановлено інструмент динамічного аналізу.

Зокрема, зловмисне програмне забезпечення може різними способами виявити потужний інструмент Frida, який використовується в цьому розділі. У професійній лабораторії аналізу зловмисного програмного забезпечення гра в кішки-мишки між застосунками, які намагаються виявити інструменти динамічного аналізу, та розробниками лабораторії, які намагаються їх приховати, є одним із аспектів професії, що забирає найбільше часу.

Ці програми перераховують системні властивості ваших пристроїв і надсилають інформацію розробникам зловмисного програмного забезпечення, які потім створюють методи антианалізу спеціально для вашого обладнання.

3.6 Тестування застосунків за допомогою хеш-коду

Тестування програм Android за допомогою хеш-коду – це метод аналізу для виявлення потенційно шкідливого програмного забезпечення. Цей підхід передбачає обчислення хешів для файлів або компонентів програми та їх порівняння з відомими базами даних шкідливих хешів.

Хеш-код – це унікальний ідентифікатор, створений за допомогою алгоритму хешування (наприклад, MD5, SHA-1 або SHA-256), який представляє вміст файлу як рядок фіксованої довжини. Змінити файл значущим чином, зберігаючи його хеш незмінним, є обчислювально складним завданням. Хеш-коди широко використовуються для перевірки автентичності файлів та порівнянням цих хешів з відомими шкідливими файлами та їх хешами в базах даних.

Програма для тестування отримується у форматі файлу APK. Алгоритми хешування, як-от MD5, SHA-1 або SHA-256, використовуються для обчислення хеш-коду для файлу APK або його компонентів (наприклад, файлів .dex, бібліотек).

Для отримання відповідних хешів програм можна використовувати величезну кількість вже створених бібліотек та пакетів, які повертають хеші файлів. Є також пакети, які навіть автоматично проводять порівняння з базами даних. Розрахований хешкод порівнюється з відомими базами даних, VirusTotal (база даних з усіма «вірусами»), сховища хешів, які підтримують розробники та локальними сховищами відомих шкідливих хешів.

Хешування є швидким і ефективним методом для виявлення відомих шкідливих програм, але він має ряд обмежень, які важливо враховувати під час розробки системи безпеки. Крім того, хешування не враховує контекст або поведінку програми, що може бути критичним для визначення, чи є програма справжньою загрозою чи ні. Хеш-коди надають лише інформацію про точну копію файлу, але не про його можливу шкідливу діяльність або здатність до змін.

Якщо хеш збігається з відомим шкідливим файлом, програма позначається як шкідлива. Цей метод досить легкий як в розробці так і в швидкості проведення аналізуобчислення хеш-кодів відбувається швидко, навіть для великих програм. Якщо відповідний хеш майже гарантує ідентифікацію шкідливого файлу якщо хеш уже є в базі даних, його можна миттєво ідентифікувати.

Проте, з головної переваги цього методу випливає його головний недолік – неефективність для невідомих загроз. Модифіковані шкідливі програми (наприклад, використання упаковки або обфускації) призведуть до різних хешів, також застарілі хеш-бази даних можуть пропускати нові загрози. Хеш-коди не дають уявлення про функціональність або поведінку програми. Тому цей метод необхідно використовувати лише для комплексної перевірки.

4 РОЗРОБКА СИСТЕМИ ВИЯВЛЕННЯ ПОТЕНЦІЙНО ЗЛОВМИСНИХ ДОЗВОЛІВ

Розробка програми є завершальним етапом аналізу, наведеного в попередніх розділах, щоб перетворити цей аналіз на інноваційну концепцію на функціональну та надійну програму.

Цей розділ заглиблюється в технічні тонкощі програми, наголошуючи на основних компонентах, рамках та методологіях, які використовуються під час її розробки. Проект розпочався з поглибленого аналізу вимог і цілей, після чого було проведено всебічне дослідження дозволів, які запитують програми Android, і їх потенційного впливу на безпеку та конфіденційність. Ця основа заклала основу для розробки детальної архітектури та інтерфейсу користувача, окреслюючи структуру, модулі та взаємодію всередині програми.

Використовуючи Android SDK і набір сторонніх бібліотек, було забезпечено ефективність, продуктивність і зручність програми. Ці технології сприяли створенню надійного та масштабованого рішення, здатного ефективно оцінювати ризик, пов’язаний із дозволами програм.

Програма також має добре розроблений інтерфейс користувача, який спрощує процес аналізу дозволів програми. Особливу увагу було приділено створенню інтуїтивно зрозумілого та зручного досвіду, що робить складне завдання оцінки безпеки доступним для користувачів із різним рівнем знань. Інтерфейс дозволяє користувачам переглядати детальну інформацію про дозволи кожної програми та пов’язану з нею оцінку ризику, надаючи чітку та практичну інформацію. Крім того, програма містить надійний модуль аналізу хешів і підписів, який порівнює підписи файлів APK із спеціальною базою даних відомих шкідливих і надійних програм.

Цей додатковий рівень аналізу підвищує точність і надійність оцінки ризиків, пропонуючи користувачам комплексний інструмент оцінки безпеки. Загалом «Система виявлення зловмисних програм Android за їхніми

дозволами» являє собою складне та орієнтоване на користувача рішення, розроблене, щоб надати користувачам знання для прийняття обґрунтованих рішень щодо програм, які вони встановлюють. Поєднання передових аналітичних методів, продумано розробленого інтерфейсу користувача та надійних серверних систем підкреслює прагнення програми забезпечити безпечну та безперебійну роботу користувача.

4.1 Розробка статичного методу

Процес статичного аналізу починається з визначення встановлених програм на пристрої. Це включає запит до API PackageManager, який надає вичерпну інформацію про всі встановлені пакети в системі. Це включає як встановлені користувачем додатки, так і системні додатки. Отримані дані можуть містити такі деталі, як назва пакета програми, значок, код версії та назва версії.

Після збору списку встановлених програм наступний етап включає вилучення дозволів, які кожна програма декларує у своєму файлі маніфесту. Ці дозволи визначають доступ додатка до конфіденційних системних ресурсів або даних користувача, таких як місцезнаходження, камера, контакти та сховище. Використовуючи такі методи, як getPackageInfo() з PackageManager, можливо отримати цю інформацію програмним шляхом (лістинг 4.1).

Лістинг 4.1 – Отримання відомостей щодо дозволів програм

```
private fun getInstalledApps(): List<PackageInfo> {
    val pm = requireContext().packageManager
    return
pm.getInstalledPackages(PackageManager.GET_PERMISSIONS)
}
```

Потім кожен дозвіл можна класифікувати на основі його чутливості або потенційних ризиків, наприклад визначити, чи є це «небезпечним» дозволом, який потребує схвалення користувача під час виконання, чи «звичайним»

дозволом, що надається автоматично. Аналізуючи оголошені дозволи, можна виявити потенційно надмірні або зловмисні запити на доступ. Для безпосередньої реалізації цього функціоналу було додано Data Class (лістинг 4.2).

Лістинг 4.2 – Клас даних дозволів

```
 data class Permission(
    val id: String,
    val alias: String,
    val group: String,
    val score: Int,
    val description: String
)
```

Класи даних у Kotlin в основному використовуються для зберігання даних. Для кожного класу даних компілятор автоматично генерує додаткові функції-члени, які дозволяють друкувати екземпляр для читання, порівнювати екземпляри, копіювати екземпляри тощо, тому цей клас підходить для зберігання даних про дозволи та їх потенційну загрозу.

Лістинг 4.3 – Встановлення рейтингу загрози

```
val permissions = getPermissionsForApp(app.packageName, pm)
val (totalDangerScore, averageDangerScore) =
calculateAppDangerMetrics(permissions)
if (averageDangerScore > 5 ){
    holder.appScore.text = "Average Danger Score:
%.2f".format(averageDangerScore)
}else{
    holder.appScore.text = "Average Danger Score:
%.2f".format(averageDangerScore)
    holder.appScore.setTextColor(Color.parseColor("#000000"))
}
```

Після того, як дозволи отримано, аналіз можна продовжити, класифікуючи їх за різними рівнями ризику, або просто розрахувавши середній рівень їх загрози (лістинг 4.4). Ця категоризація допомагає виявити

шаблони, коли програма може запитувати дозволи, не пов’язані з її основною функціональністю, що може вказувати на потенційні зловмисні наміри або надмірні дозволи.

Лістинг 4.4 – Розрахунок середнього значення небезпеки дозволів

```
fun calculatePermissionScore(permissionId: String): Int {
    // Check if permission exists in the list
    val permission = permissions.find { it.id == permissionId }

    // Transform the score (10 becomes 0, 1 becomes 9, etc.) or
default to 0 if not found
    return permission?.let { 10 - it.score } ?: 9
}
```

Представлення результатів аналізу в чіткій та лаконічній формі є важливим. Дозволи для кожної програми можна відобразити у форматі списку, де вказано назву програми, її оголошені дозволи та їхні відповідні рівні ризику.

Для комерціалізації проєкту можна провести ключення візуальних індикаторів, таких як кольорові коди або піктограми, для виділення конфіденційних дозволів може ще більше покращити зручність використання результатів. Процес статичного аналізу можна автоматизувати, щоб періодично перевіряти наявність змін у дозволах встановлених програм. Це гарантує, що користувачі отримають сповіщення, якщо програма оновиться та почне запитувати нові дозволи, які можуть викликати занепокоєння. Інтеграція цієї функції в систему може забезпечити безперервний моніторинг і покращити безпеку пристрою з часом. Виконуючи ці кроки, статичний аналіз дозволів надає потужний інструмент для розуміння та керування наслідками безпеки встановлених програм. Це дозволяє проактивно ідентифікувати потенційно шкідливі або непотрібні дозволи та покращує загальну обізнаність користувачів про безпеку.

Щоб зробити аналіз ефективним, інтерфейс може надавати користувачам варіанти наступних кроків на основі відображених результатів.

Наприклад, користувачі можуть видалити програми безпосередньо з інтерфейсу, якщо знайдуть програми з надмірними дозволами, або детально переглянути дозволи для певних програм. Надання навчальних матеріалів або рекомендацій, як-от пояснення, чому певні дозволи можуть бути ризикованими, може допомогти користувачам приймати зважені рішення.

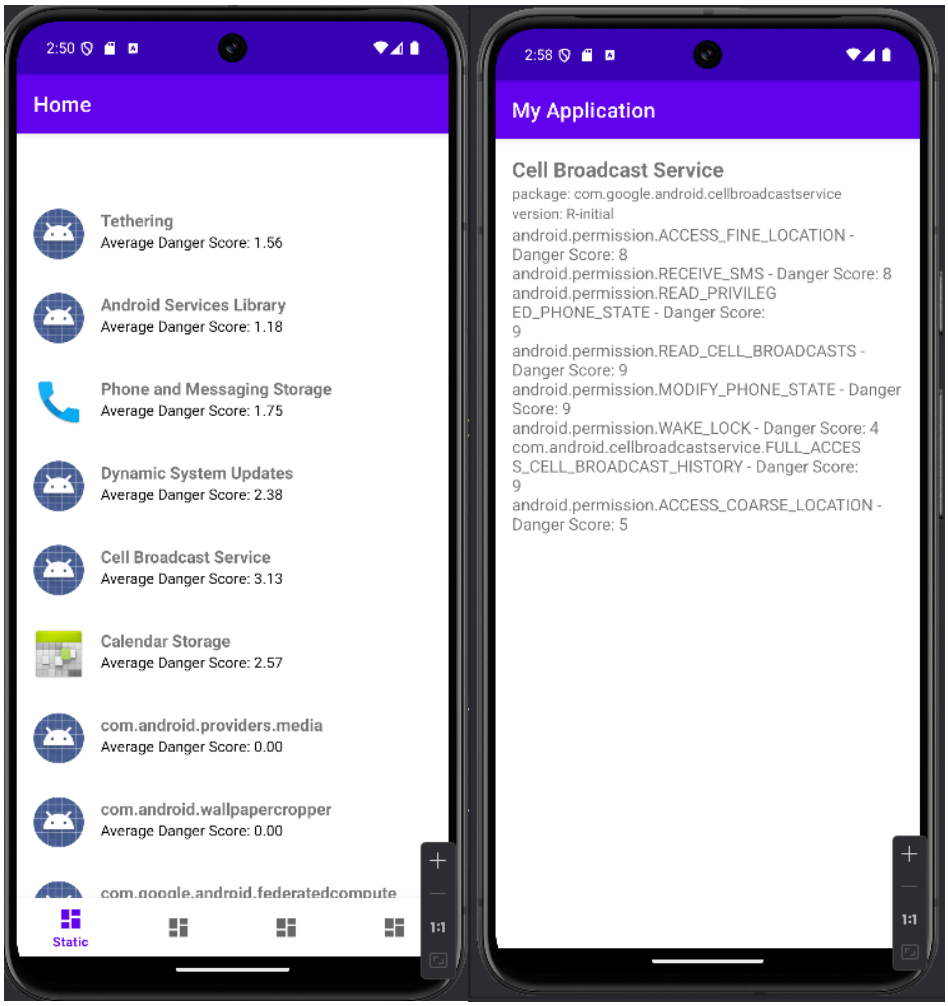

Рисунок 4.1 – Інтерфейс статичного методу

Процес статичного аналізу можна автоматизувати, щоб періодично перевіряти наявність змін у дозволах встановлених програм. Це забезпечує сповіщення користувачів, якщо програма оновлюється та починає запитувати нові дозволи, які можуть викликати занепокоєння. Інтеграція цієї функції в систему може забезпечити постійний моніторинг і покращити безпеку

пристрою з часом. Виконуючи ці дії, статичний аналіз дозволів надає потужний інструмент для розуміння та керування наслідками безпеки встановлених програм. Це дозволяє завчасно виявляти потенційно шкідливі або непотрібні дозволи та покращує загальну обізнаність користувачів про безпеку.

4.2 Розробка методу перевірки за допомогою хеш-кодів

У контексті виявлення потенційно шкідливих програм хешування є важливим інструментом для виявлення та перевірки цілісності програм. Хешування – це процес перетворення будь-яких вхідних даних (наприклад, файлу APK програми) у рядок фіксованої довжини, який однозначно представляє вміст цього файлу.

Хеш-коди (наприклад, MD5, SHA-1 або SHA-256) можна використовувати для порівняння хешу файлу APK програми з базою даних відомих шкідливих хешів для виявлення загроз. Хоча MD5 (128-біт) швидший і створює коротший хеш, він вважається менш безпечним через такі вразливості, як колізії хешів.

Однак це все ще корисно для швидких порівнянь і перевірок у деяких контекстах. Хешування SHA-256: SHA-256 (256-біт) є більш безпечним і має меншу ймовірність зіткнутися з хеш-колізіями, що робить його кращим варіантом для криптографічного використання. Однак він потребує більшої обчислювальної потужності, що може бути компромісом з точки зору продуктивності.

Перш за все потрібно зібрати список зловмисних хеш-кодів на форумах та ресурсах, виявилося, що знайти необхідний список зловмисних хешів – не просте завдання і вимагає багато часу і ручної рутинної роботи, яку майже неможливо автоматизувати, тому як базу даних було використано лише хешкоди для MD5. Відомості про хеш програм MD5 частіше знаходився на форумах та відповідних ресарсах ніж SHA-1 або SHA-256.

Потім необхідно створити або клас даних аба базу даних для того щоб зберігати хеші в локальному файлі або онлайн-базі даних, яка постійно оновлюється. База даних може являти собою простий список рядків, що представляють хеші відомих шкідливих програм. Цей список може включати хеші раніше виявлених зловмисних програм або хеш-значення, які були позначені дослідниками безпеки.

Лістинг 4.4 – Список хешів зловмисних програм

```
val hashes: List<Hashes> = listOf(
    Hashes(id = "fa35b0a2acd5565ade6d3e1af64a94"),
    Hashes(id = "6ffa35b0a2acd5565ade6d3e1af64a94"),
    Hashes(id = "12b5cc085c974ac955c37b1f84dda8ce"),
    Hashes(id = "e9b2d68cd1de41c6278776f2d1249676"),   Hashes(id
= "1c2a2d1853aafec963e5a62264f68134"),
    Hashes(id = "11b8e669e71e956c138324272200e099"),
    Hashes(id = "28ddb9b5b14fcca82c4b53af8ba0e03c"),
    Hashes(id = "89dfd0590058f9021aac24e69a3132fe"),
    Hashes(id = "5080cd5a0a56c99022a3fdfde6107f6b"),
//etc}
```

Другий крок у розробці – отримати хеш встановленої програми. Як правило, можна обчислити хеш файлу APK з каталогу інсталяції програми. Для цього знадобиться отримати доступ до файлу APK програми, прочитати його вміст і створити хеш. Розроболена функція читає файл APK і генерує хеш за вказаним алгоритмом.

Лістинг 4.5 – Розрахунок хешу MD5

```
import java.security.MessageDigest
fun generateMD5Hash(input: String): String {
    val md = MessageDigest.getInstance("MD5")
    val digest = md.digest(input.toByteArray())
    return digest.joinToString("") { "%02x".format(it) }
}
```

Після того, як функція обчислила хеш файлу APK програми, наступним

кроком потрібно провести порівняння його зі списком відомих шкідливих хешів, який додано до локального файлу. Якщо збіг знайдено необхідно позначити програму як потенційно шкідливу. Ця проста функція (лістинг 4.6) перевіряє, чи існує обчислений хеш у списку відомих шкідливих хешів.

Лістинг 4.6 – Перевірка хешу

```
val isMalicious = hashes.any { it.id == appHash }

if (isMalicious) {
    holder.appScore.text ="Calculated hash is $appHash, there is
this hash in the malicious database, IT IS DANGEROUS"}
else{
    holder.appScore.setTextColor(Color.parseColor("#000000"))
    holder.appScore.text = "Calculated hash is $appHash, there
is no this hash in the malicious database"
}
```

Якщо це так, програма позначається як потенційно шкідлива. Можливо використовувати цю функцію в логіці програми, щоб динамічно обчислювати та перевіряти хеш програм. Таким чином, кожного разу, коли програма потрапляє в список, її хеш обчислюватиметься та перевірятиметься зі списком шкідливих хешів.

Останнім кроком є чітке відображення результатів для користувача. Якщо хеш програми збігається з відомим шкідливим записом, можна відобразити попередження або надати можливість видалити чи заблокувати програму. І навпаки, якщо програма безпечна, можна відобразити повідомлення про те, що хеш не відповідає жодній відомій загрозі.

Щоб посилити заходи безпеки програми, важливо регулярно оновлювати шкідливі хеші – періодично отримувати нові хеші з надійних джерел або використовувати хмарні бази даних, які зберігають оновлені списки сигнатур шкідливих програм.

Виконуючи ці кроки, можна використовувати хеш-коди як ефективний метод ідентифікації шкідливих програм у вашому проекті Android. Цей процес

використовує хешування для обчислення сигнатур програми та порівнює їх із базою даних відомих шкідливих хешів, гарантуючи, що ваша програма залишається захищеною від поширених загроз.

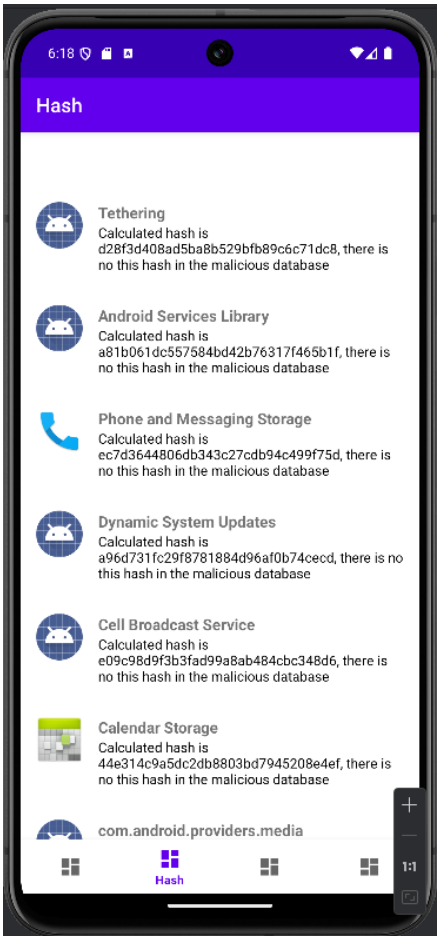

Рисунок 4.2 – Інтерфейс метод перевірки за допомогою хешкодів

Для подальшого підвищення безпеки необхідна інтеграція механізмів для регулярного оновлення хеш-бази даних із надійних джерел. Використовуючи зручні інтерфейси для відображення результатів і виконання відповідних дій, таких як видалення позначених програм або перегляд дозволів, система дає користувачам можливість проактивно захищати свої пристрої.

У поєднанні з іншими методами, такими як аналіз дозволів і моніторинг часу виконання, цей метод не тільки підвищує точність виявлення шкідливих

програм, але й створює довіру та впевненість у безпеці пристроїв Android. Регулярні оновлення, зосередженість на взаємодії з користувачем і багаторівневий підхід до безпеки є ключовими для максимізації переваг цієї техніки.

4.3 Розробка динамічного методу

За один із варіантів розробки моніторинг мережевого трафіку. Динамічний метод стосується аналізу або перевірки чогось під час роботи програми. У цьому випадку метою є динамічне відстеження використання мережевих даних шляхом моніторингу метрик у реальному часі, пов’язаних із надісланими та отриманими даними.

На відміну від статичних методів, які аналізують дані в один момент часу або на основі попередньо визначених умов, динамічні методи коригуються та оновлюються в реальному часі на основі поточної поведінки програми. В Android існує кілька підходів для досягнення цього динамічного моніторингу мережевого трафіку. Деякі з найпоширеніших методів покладаються на системні служби, такі як TrafficStats і NetworkStatsManager.

Клас TrafficStats в Android є частиною пакета android.net і надає простий спосіб відстежувати використання мережевих даних для програми. Він надає такі методи, як getUidRxBytes і getUidTxBytes, щоб отримати отримані та передані байти для певного UID (ідентифікатора користувача), пов’язаного з цією програмою.

Це один із найпростіших способів динамічного моніторингу того, скільки даних передається програмою. Ці значення оновлюються динамічно, і їх можна опитувати через регулярні проміжки часу (наприклад, кожну секунду), щоб постійно відстежувати використання даних.

Клас TrafficStats простий у використанні та не потребує явного дозволу, окрім звичайних пов’язаних з Інтернетом дозволів.

Однак важливо зауважити, що, починаючи з Android 10 (рівень API 29),

програми можуть отримувати доступ до статистики трафіку лише для власного UID. Для програм, націлених на старіші версії, можна було отримати доступ до статистики мережі в усій системі, включаючи інші програми.

У разі націлювання на Android Q або новішої версії, якщо потрібно отримати доступ до використання даних для інших програм, вам потрібно буде використовувати інший підхід, зазвичай залучаючи NetworkStatsManager.

Клас NetworkStatsManager доступний, починаючи з Android 6.0 (рівень API 23), і він розроблений, щоб дозволити програмам отримувати доступ до детальної статистики використання мережі. Основний метод, querySummaryForDevice, дозволяє запитувати використання мережі через різні мережеві інтерфейси, такі як Wi-Fi і мобільні дані.

Для програм, що працюють на Android Q (рівень API 29) або пізнішої версії, NetworkStatsManager є рекомендованим способом моніторингу вхідного та вихідного мережевого трафіку на кожному пристрої без обмежень, пов’язаних з UID програми.

Лістинг 4.7 – Перевірка API

```
private fun startNetworkMonitoring() {
    monitoringJob = CoroutineScope(Dispatchers.IO).launch {
        Log.d("RATARA", "Coroutine started")

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
            val networkStatsManager =
requireContext().getSystemService(Context.NETWORK_STATS_SERVICE)
as NetworkStatsManager
            monitorTotalTraffic(networkStatsManager)
        } else {
            val uid = Process.myUid()
            monitorAppTraffic(uid)
        }
    }
}
```

Для доступу до мережевих даних на пристрої NetworkStatsManager потрібен дозвіл PACKAGE_USAGE_STATS. Це означає, що якщо програма націлена на рівень API 29 та вище, користувач повинен надати дозвіл на

доступ до статистики використання програми. Якщо користувач не ввімкнув цей дозвіл, програма не зможе отримати дані, і в таких випадках має з’явитися підказка, щоб направити користувача до відповідного екрана налаштувань, щоб увімкнути його.

Лістинг 4.8 – Меню дозволу PACKAGE_USAGE_STATS

```
private fun requestUsageAccessPermission() {
    val intent = Intent(Settings.ACTION_USAGE_ACCESS_SETTINGS)
    startActivity(intent)
}
```

Функція updateUI призначена для оновлення інтерфейсу користувача даними мережевого трафіку. Він виконується асинхронно за допомогою корутин та дозволяє програмі призупиняти та відновлювати своє виконання, що робить його ідеальним для операцій, які потребують багато часу, як-от моніторинг мережі, без блокування основного потоку інтерфейсу користувача.

Лістинг 4.9 – Функція оновлення даних

```
private suspend fun updateUI(received: Long, transmitted: Long)
{
    withContext(Dispatchers.Main) {
        // Updating the UI with received and transmitted values
in KB
        val textView: TextView =
view?.findViewById(R.id.text_notification) ?: return@withContext
        textView.text = "Received: ${received / 1024} KB, Sent:
${transmitted / 1024} KB"
        Log.d("RATARA", "Received: ${received / 1024} KB, Sent:
${transmitted / 1024} KB")
    }
}
```

Функція приймає два параметр, що представляють отримані та передані пристроєм дані, виміряні в байтах. Метою цієї функції є відображення цих

значень користувачеві в більш зручному для читання форматі, зокрема в кілобайтах (КБ), оскільки дані спочатку представлені в байтах. Функція використовує блок withContext(Dispatchers.Main), щоб переключити виконання на головний потік. Це важливо, оскільки оновлення інтерфейсу користувача мають відбуватися в основному потоці в Android.

Рисунок 4.3 – Приклад інтерфейсу динамічного методу

Що стосується налаштування систем для динамічного аналізу, наведених в розділі 2, то їх надзвичайно складно ітегрувати до власного застосунку. DroidBox і TaintDroid розроблені як інструменти системного рівня, які працюють на рівні ОС Android. Вони вимагають модифікації фреймворку Android, що ускладнює їх інтеграцію в окремі програми без глибоких змін у самій операційній системі.

DroidBox і TaintDroid зазвичай потребують спеціального ПЗП (постійний запам'ятовувальний пристрій) або модифікованого емулятора Android. Це означає, що їх не можна просто об’єднати або вбудувати в стандартні програми Android. Натомість їм потрібне спеціалізоване середовище для належної роботи. Ці інструменти часто потребують підвищених дозволів, які не надаються звичайним програмам. Це включає доступ до системних журналів, низькорівневих потоків даних і конфіденційних внутрішніх процесів, які виходять за межі можливостей типових дозволів додатків Android.

У сукупності ці фактори роблять непрактичним безпосередню інтеграцію DroidBox або TaintDroid в окремі програми. Натомість вони використовуються як зовнішні інструменти в середовищі контрольованого аналізу.

Для налаштування DroidBox доступний на GitHub. Для початку необхідно клонувати репозиторій на свою машину, інсталювати необхідні залежності для запуску DroidBox, включаючи Python, ADB і емулятор Після запуску потрібно помістити APK, який потрібно проаналізувати, у папку DroidBox. Запустіть DroidBox, щоб проаналізувати APK. Після чого DroidBox створюватиме журнали, які відображатимуть поведінку програми, включаючи мережеву активність, операції з файлами тощо.

Інтеграція TaintDroid TaintDroid, в свою чергу, відстежує потоки конфіденційних даних у режимі реального часу, щоб виявляти потенційні порушення конфіденційності в програмах Android.

TaintDroid інтегровано в модифікований образ ОС Android. TaintDroid надає журнали, які вказують, як дані проходять через програму, чи залишають

4.4 Розробка методу аналізу за допомогою ML

Одним з варіантів розробки методу аналізу за допомогою машинного навчання є використання сторонніх API, такий як OpenAI API. OpenAI – це

компанія з дослідження та впровадження штучного інтелекту, яка зосереджена на тому, щоб штучний загальний інтелект (AGI) приносив користь всьому людству. Компанія відома розробкою передових технологій ШІ, таких як моделі GPT (Generative Pretrained Transformer), включаючи GPT-3 і GPT-4, які зробили революцію в області обробки природної мови.

Зараз OpenAI є однією з провідних організацій у галузі штучного інтелекту, яка просуває як теорію, так і застосування моделей штучного інтелекту в різних сферах, включаючи створення мови, навчання з підкріпленням і робототехніку.

На додаток до своєї роботи в обробці природної мови OpenAI зробив внесок у навчання з підкріпленням, коли моделі навчаються через взаємодію з середовищем. Це особливо важливо при розгляді потенційного майбутнього впливу AGI, який може перевершити людський інтелект і створити ризик, якщо не узгоджуватиметься з людськими цінностями. Щоб зменшити ці ризики, OpenAI прийняла підхід «безпека на першому місці», який включає проведення ретельних досліджень щодо безпеки та етики ШІ. Компанія також співпрацює із зовнішніми організаціями, урядами та інституціями, щоб розробити політику та рамки для відповідального розвитку ШІ.

OpenAI API та комерційне використання OpenAI зробив свої моделі, включаючи GPT-3, доступними для розробників через API (інтерфейс прикладного програмування), дозволяючи компаніям, дослідникам і розробникам інтегрувати потужні можливості ШІ у свої програми. API підтримує широкий спектр варіантів використання, від розмовних систем штучного інтелекту та чат-ботів до створення вмісту, узагальнення та навіть створення коду.

Пропонуючи ці послуги через платний API, OpenAI дозволяє широко використовувати свої технології, одночасно забезпечуючи сталий розвиток своїх досліджень. API OpenAI використовувався багатьма компаніями та організаціями для створення інноваційних продуктів, зокрема ботів для обслуговування клієнтів, розробників персоналізованого контенту та навіть

розробників ігор на основі ШІ. API є значним кроком у місії OpenAI щодо демократизації доступу до передового ШІ та надання можливості більшій кількості людей використовувати потужність цих моделей. Проте, головним недоліком цього API є платне використання.

Для початку необхідно зайти на офіційний сайт та згенерувати ключ, який потрібен для побудови запиту. Згенерувавши його, його необхідно розмістити безпосередньо в програмі та створити запит до цього API. Оскільки запит робиться за допомогою Інтернету, то необхідно використати залежності (лістинг 4.10) для використання запитів.

Лістинг 4.10 – Додавання залежностей для роботи з API

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'com.squareup.okhttp3:logging-
interceptor:4.9.1'
}
```

Після того, необхідно створити типовий запит до API за допомогою встановлених залежностей. Для виконання запитів до API було використано Retrofit. Retrofit – це надійний, безпечний HTTP-клієнт. Він спрощує процес взаємодії з RESTful API, перетворюючи кінцеві точки HTTP API в інтерфейси Java.

Ця трансформація допомагає розробникам уникнути шаблонного коду та підвищує зручність обслуговування програми завдяки дотриманню принципів об’єктно-орієнтованого програмування. Однією з видатних особливостей Retrofit є його здатність обробляти серіалізацію та десеріалізацію даних. Він автоматично перетворює відповіді JSON або XML в об’єкти Java.

Модернізація робить мережеві виклики ефективними та простими завдяки використанню таких анотацій, як @GET, @POST, @PUT і @DELETE. Ці анотації допомагають визначити методи HTTP для конкретних кінцевих

точок API, роблячи код більш читабельним і організованим. Крім того, він підтримує як синхронні, так і асинхронні мережеві запити, що дає розробникам у виборі відповідно до вимог додатків.

Лістинг 4.11 – Типовий запит до API

```
object RetrofitInstance {
    private const val BASE_URL = "https://api.openai.com/"

    private val client = OkHttpClient.Builder().addInterceptor {
chain ->
        val originalRequest: Request = chain.request()
        val newRequest = originalRequest.newBuilder()
            .header("Authorization", "Bearer
${ApiConstants.OPENAI_API_KEY}")
            .build()
        chain.proceed(newRequest)
    }.build()

    val api: OpenAIService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .client(client)  // Add the OkHttp client with the
interceptor
            .addConverterFactory(GsonConverterFactory.create())
            .build()
            .create(OpenAIService::class.java)
    }
}
```

Об’єкт RetrofitInstance — це синглтон, який налаштовує конфігурацію Retrofit для викликів API до OpenAI API. В основі цього налаштування лежить BASE_URL, який визначає кореневу URL-адресу для запитів API. Усі мережеві виклики, здійснені за допомогою цього екземпляра, починатимуться з базової URL-адреси . Це гарантує, що виклики спрямовуються до правильної кінцевої точки API.

Об’єкт також визначає налаштований OkHttpClient. Цей клієнт необхідний для керування мережевими запитами та відповідями. До клієнта додається перехоплювач, який перехоплює кожен вихідний запит для додавання заголовка авторизації. Цей заголовок містить маркер носія,

необхідний для автентифікації запитів API. Маркер динамічно отримується із попередньо визначеної константи ApiConstants.OPENAI_API_KEY, що гарантує безпечну автентифікацію викликів API без ручного включення маркера в кожен запит. Екземпляр Retrofit використовується для створення реалізації інтерфейсу OpenAIService. Цей інтерфейс визначає різні кінцеві точки OpenAI API, з якими може взаємодіяти програма. Використовуючи це налаштування, розробники можуть здійснювати мережеві виклики OpenAI API структурованим і ефективним способом, з належною автентифікацією та перетворенням даних, інтегрованим у процес. Використавши цей код необхідно перевірити чи повертаються якісь дані від API. Якщо все добре, то необхідно перевірити, які саме коди повертаються, що вони означають та що потрібно, щоб їх вирішити. В контексті платних API варто розглядати також помилку 429 –insufficient_quota, яке означає, що використання OpenAI API перевищило дозволену квоту для плану. Зокрема, OpenAI повідомля, що було перевищено доступні кредити або ліміти використання API, пов’язані з обліковим записом.

Функція displayPermissionsDialog призначена для відображення діалогового вікна, яке містить список дозволів програми та додаткове пояснення щодо цих дозволів (лістинг 4.12), яке сформовано запитом до ШІ (лістинг 4.13)

Лістинг 4.12 – Візуалізація відповіді ШІ

```
private fun displayPermissionsDialog(permissions: List<String>,
permExplanation: String) {
    // You can use an AlertDialog or any UI component to show
the permissions list
    val permissionsText = permissions.joinToString("\n") { it }
    AlertDialog.Builder(requireContext())
        .setTitle("App Permissions")
        .setMessage(permissionsText + permExplanation)
        .setPositiveButton("OK", null)
        .show()
    Log.d("TAGTAG",  permissionsText))
}
```

Функція displayPermissionsDialog – це функція, призначена для відображення діалогового вікна, яке містить список дозволів програми та додаткове пояснення щодо цих дозволів. Він приймає два параметри: дозволи, список рядків, що представляють дозволи, і permExplanation, рядок, який надає пояснення щодо цих дозволів. Усередині функції список дозволів перетворюється в один рядок під назвою permissionsText. Це робиться за допомогою методу joinToString, який об’єднує елементи списку за допомогою символу нового рядка.

Лістинг 4.13 – Безпосередній запит до API

```
val call =
RetrofitInstance.api.getPredictionExplanation(request)
        call.enqueue(object : Callback<OpenAIResponse> {
            override fun onResponse(call: Call<OpenAIResponse>,
response: Response<OpenAIResponse>) {
                if (response.isSuccessful) {
                    val explanation =
response.body()?.choices?.firstOrNull()?.message?.content ?: "No
explanation"
                    Log.d("Explanation:", explanation)

                } else {
                    Log.d("API ERROR", "Error:
${response.code()} - ${response.errorBody()?.string()}")
                }
            }

            override fun onFailure(call: Call<OpenAIResponse>,
t: Throwable) {
                Log.d("Network Error", "${t.message}")
                println("Network Error: ${t.message}")
            }
        })
```

Таким чином, кожен дозвіл відображатиметься в новому рядку результуючого рядка, що полегшить читання списку в діалоговому вікні. Потім функція створює AlertDialog за допомогою AlertDialog.Builder.

Заголовок діалогового вікна встановлено на «Дозволи програми». Повідомлення діалогового вікна поєднує в собі permissionsText і

permExplanation, відображаючи список дозволів із додатковим поясненням. У діалоговому вікні є одна кнопка «OK», яка закриває діалогове вікно після натискання. Нарешті, діалогове вікно відображається за допомогою методу show(). Функція також записує рядок permissionsText на консоль за допомогою Log.d, що корисно для цілей налагодження. Цей журнал може допомогти розробникам побачити, які дозволи відображаються в діалоговому вікні.

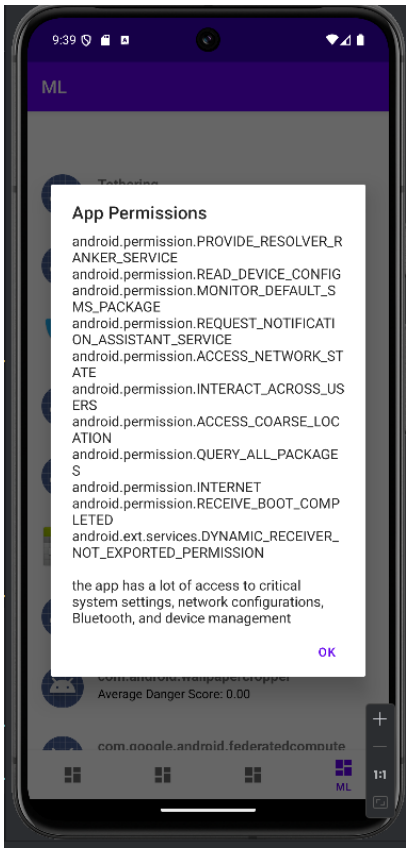

Рисунок 4.4 – Приклад інтерфейсу динамічного методу

Важливо зауважити, що кожне натискання в адаптері на застосунок в списку призводить до нового запиту до Open AI API, що в свою чергу спричиняє списування коштів з балансу користувача. Можна додатково додати локальну базу даних, яка буде перевіряти чи проводився запит до API та зберігати його значення, перевіряючи версію програми. Якщо версія програми була змінена, значить потенційно нові дозволи могли бути застосовані та потрібно робити новий аналіз, зберігаючи його.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було розглянуто небезпеку, що виникає при використанні програм, що функціонують на операційній системі Android. Надавання великої кількості та відсутність керування дозволами застосунками є ризиком, що дані не будуть вкрадені або пошкоджені. Своєчасна перевірка дозволів може забезпечити безпеку та конфіденційність даних користувача пристрою в екосистемі Android.

Метою аналізу було дослідження дозволів та складання рейтингу їх небезпечності на підставі досліджень профільних організацій та власного досвіду щодо варіантів використання дозволів в тих чи інших програмах, які потенційно можуть бути шкідливими. Було встановлено найбільш небезпечні одиниці, приклади їх використання та порівняльну характеристику з іншими небезпечними дозволами. Наведено статистику частоти використання цих дозволів та загальну динаміку по створенню зловмисних застосунків на базі цієї операційної системи. Також метою аналізу було також встановлення факту поведінки застосунку при його роботі та комбінації декількох різних дозволів і загальний аналіз для встановлення більшої вірогідності небезпеки. Було досліджено прийняті шляхи боротьби з небезпечними застосунками, які використовують дозволи для своєї роботи, а також безпосередньо проведено аналіз сучасних алгоритмічних рішень, які використовуються для пошуку підозрілих дозволів користувачів:

- статичний метод;
- перевірка за хеш кодами;
- динамічний;
- за допомогою машинного навчання, який в свою чергу розділений також на декілька категорій, оскільки даний підхід складний.

В ході аналізу було встановлено яким саме чином можна використовувати дані засоби для боротьби, які засоби вже є в програмах, які

можуть виникнути проблеми при використанні емулятора та реального пристрою, та ін. Досліджено методи пошуку і встановлено, що кожен підхід має як свої значні переваги так і свої значні недоліки.

Також розглянуто приклади, як саме дозволи налають доступ зловмисникам для інтегрування свої злочинних систем в пристрої та наслідки їх дій для користувачів. Було наведено методи виявлення та аналізу зловмисних дозволів та надані рекомендації як не допустити доступу зловмисникам до дозволів операційної системи.

Одним з найважливіших аспектів розробки програми, що може бути розроблена для пошуку небезпечних дозволів може бути використання всіх проаналізованих методів в одному застосунку для отримання найкращого результату: статичного, динамічного методів, за допомогою ML (машинного навчання) та хешкодів. Було надано інформацію щодо вже реалізованих засобів для кожного наведеного підходу, щодо засобів для декомпіляції коду на операційній системі Android, щодо вже навчених моделей машинного навчання. В процесі використання моделей було виявлено що різні моделі машинного навчання, змінюючи розмір набору даних, а також генерацію та сімейство шкідливих програм Android перевершили інші моделі у виявленні зловмисного програмного забезпечення для різних конфігурацій. У процесі проектування було використано інформацію та розроблено набори даних за допомогою аналізу статичної поведінки Android із різних сімейств шкідливого програмного забезпечення Android та різних часових проміжків.

Було розроблено програму, що можуть бути використана як приклад для майбутньої розробки комерційно значущого застосунку-відстежувача на ОС Android відповідно до нових викликів, які провокуються зі сторони зловмисників.

В результаті даної роботи покладено початок для розробки застосунку, що використовує найкращі практики нативного програмування під операційну систему Android для відстеження застосунків, які потенційно можуть використовувати дозволи для передачі даних користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. NAMRUD, Zakeya; KPODJEDO, Sègla; TALHI, Chamseddine. AndroVul: A repository for Android security vulnerabilities. In: Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering. 2019. p. 64-71. (дата звернення: 01.08.2024).

2. Android Vulnerabilities. URL: http://www.androidvulnerabilities.org/ (дата звернення: 01.08.2024).

3. Ratul, Qudrat E Alahy & Chowdhury, Naseef & Soliman, Hamdy & Chaity, Moshrefa Sultana & Haque, Ahshanul. (2020). Android Malware Detection in Large Dataset: Smart Approach. 10.1007/978-3-030-39445-5_58. (дата звернення: 01.08.2024).

4. Mohamad Arif J, Ab Razak MF, Awang S, Tuan Mat SR, Ismail NSN, Firdaus A. A static analysis approach for Android permission-based malware detection systems. PLoS One. 2021 Sep 30;16(9):e0257968. doi: 10.1371/journal.pone.0257968. PMID: 34591930; PMCID: PMC8483345. (дата звернення: 01.08.2024).

5. Almomani, Iman M., and Aala Al Khayer. "A comprehensive analysis of the android permissions system." Ieee access 8 (2020): 216671-216688. (дата звернення: 01.08.2024).

6. DOĞRU, İbrahim Alper; ÖNDER, Murat. AppPerm analyzer: malware detection system based on android permissions and permission groups. International Journal of Software Engineering and Knowledge Engineering, 2020, 30.03: 427-450. (дата звернення: 01.08.2024).

7. AKBAR, Fahad, et al. Permissions-based detection of android malware using machine learning. Symmetry, 2022, 14.4: 718. (дата звернення: 01.08.2024).

8. RATHORE, Hemant, et al. Identification of significant permissions for efficient android malware detection. In: International conference on broadband communications, networks and systems. Cham: Springer International Publishing,

2020. p. 33-52. (дата звернення: 01.08.2024).

9. DHARMALINGAM, Varna Priya; PALANISAMY, Visalakshi. A novel permission ranking system for android malware detection—the permission grader. Journal of Ambient Intelligence and Humanized Computing, 2021, 12: 5071-5081. (дата звернення: 01.08.2024).

10. HSIAO, Shou-Ching; LI, Shih-Wei; HSIAO, Hsu-Chun. Risky Cohabitation: Understanding and Addressing Over-privilege Risks of Commodity Application Virtualization Platforms in Android. In: Proceedings of the Fourteenth ACM Conference on Data and Application Security and Privacy. 2024. p. 253-264. (дата звернення: 01.08.2024).

11. Android 10. Highlights. URL: https://developer.android.com/about/versions/10/privacy (дата звернення: 01.09.2024).

12. Open AI overviewURL: https://platform.openai.com/docs/overview (дата звернення: 01.09.2024).

13. Mobile security. URL: https://www.gdata.de/help/en/consumer/Products/MobileSecurity/Android/msa_index/ (дата звернення: 01.09.2024).

14. Mobile security. URL: https://pts-project.org/guides/g5/ (дата звернення: 01.09.2024).

15. Mobile security. URL: https://www.virustotal.com/gui/home/search (дата звернення: 01.09.2024).

16. Mobile security. URL: https://www.hybrid-analysis.com/ (дата звернення: 01.09.2024).

17. Mobile security. URL: https://www.apkmirror.com/ (дата звернення: 01.09.2024)

18. Liu, Mei, and Qun Wang. "Research on android user privacy permission analysis and protection mechanism under big data environment." MATEC Web of Conferences. Vol. 395. EDP Sciences, 2024. (дата звернення: 01.06.2024).

19. Enck, William, et al. "Understanding Android Security." IEEE Security & Privacy. Vol. 7, No. 1, 2009, pp. 50-57. (дата звернення: 01.06.2024).

20. Felt, Adrienne Porter, et al. "Android Permissions: User Attention,

Comprehension, and Behavior." Proceedings of the Eighth Symposium on Usable Privacy and Security. ACM, 2012. (дата звернення: 01.06.2024).

21. Garg, Swapnil. "Android Application Security Essentials." Packt Publishing, 2013. (дата звернення: 01.06.2024). Nguyen, Triet, and Murat Kantarcioglu. "Protecting User Privacy in Android App Ecosystem with Permission Usages." Lecture Notes in Computer Science. Springer, 2014. (дата звернення: 01.06.2024).

22. Wei, Fuchun, and Tao Zhu. "Improving Android Security with Fine-Grained Permission Control." Security and Privacy in Mobile Information and Communication Systems. Springer, 2012. (дата звернення: 01.06.2024).

23. Vidas, Timothy, and Nicolas Christin. "Evading Android Runtime Analysis via Sandbox Detection." Proceedings of the Ninth Annual ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. ACM, 2013. (дата звернення: 01.06.2024).

24. Permissions. URL: https://developer.android.com/guide/topics/permissions/overview (дата звернення: 01.09.2024)

25. Releases. URL: https://developer.android.com/tools/releases/platforms (дата звернення: 01.09.2024)

26. Android 10. Highlights. URL: https://developer.android.com/about/versions/10/highlights (дата звернення: 01.09.2024)