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

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

«На правах рукопису»
УДК 004.4_____

До захисту допущено:
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2023 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою
«Системне програмування і спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в ОС Android»

Виконав:
студент ІІ курсу, групи КВ-11мн
Кравчук Олександр Віталійович _____

Науковий керівник:
професор кафедри СПСКС, д.т.н., професор,
Терейковський Ігор Анатолійович_____ _____

Рецензент:
_____
_____
_____

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

Київ – 2023 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп‘ютерна інженерія»

Освітньо- наукова програма «Системне програмування і спеціалізовані комп‘ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Віталій РОМАНКЕВИЧ

«___»_____2023 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Кравчуку Олександру Віталійовичу

1. Тема дисертації «Спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в ОС Android», науковий керівник дисертації Терейковський Ігор Анатолійович, д.т.н., професор, затверджені наказом по університету від «30» 03. 2023 р. №. 1359-С.

2. Термін подання студентом дисертації 18.05.2023.

3. Об’єкт дослідження: виявлення зловмисного програмного забезпечення в застосунку операційної системи Android.

4. Вихідні дані: спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в операційній системі Android.

5. Перелік завдань, які потрібно розробити: розробити та описати спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в операційній системі Android.

6. Перелік графічного (ілюстративного) матеріалу: презентація.

7. Перелік публікацій:

- XIV науково-практична конференція магістрантів та аспірантів ПМК-2021 факультету прикладної математики;
- LXVII міжнародна науково-практична конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення»;
- Міжнародна мультидисциплінарна наукова інтернет-конференція «Світ наукових досліджень. Випуск 18»;

8. Дата видачі завдання 10.10.2020.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Узгодження технічного завдання	10.10.2020	
2	Дослідження та вивчення літератури	01.09.2021	
3	Розробка структури МД	01.10.2021	
4	Аналіз існуючих рішень	01.12.2021	
5	Написання першого розділу МД	01.01.2022	
6	Написання другого розділу МД	01.02.2022	
7	Розробка та реалізація алгоритму	01.03.2022	
8	Написання третього розділу МД	21.01.2023	
9	Написання четвертого розділу МД	15.02.2022	
10	Написання п’ятого розділу МД	10.04.2023	
11	Підготовка графічного матеріалу	03.05.2023	
12	Розгляд МД на кафедрі	04.05.2023	

Студент Олександр КРАВЧУК

Науковий керівник Ігор ТЕРЕЙКОВСЬКИЙ

РЕФЕРАТ

Актуальність теми. Мобільні пристрої є невід'ємною частиною повсякденного життя людства. Від соціальних мереж до банківських транзакцій мобільні пристрої користуються довірою мільйонів людей по всьому світі. В той же час існує тенденція зростання кількість шкідливого програмного забезпечення, котра є актуальною для всіх популярних мобільних платформ.

На даний момент операційна система Android є найпоширенішою мобільною платформою у світі. Відповідно до цього факту, на ній щороку розгортається незліченна кількість зловмисних застосунків, що мають на меті нашкодити роботі пристрою та отримати доступ до конфіденційних даних користувача. Виходячи з вище зазначеної інформації, проблема виявлення таких додатків є вкрай актуальною на сьогоднішній день і саме тому потребує нових ідей та підходів.

Об’єктом дослідження є виявлення зловмисного програмного забезпечення в операційній системі Android базуючись на методі статичного аналізу програмного забезпечення.

Предметом дослідження є програмне забезпечення, що реалізовує запропонований спосіб виявлення зловмисного програмного забезпечення в застосунку операційної системи Android.

Мета роботи: дослідити методологію виявлення шкідливого програмного забезпечення та метод статичного аналізу зокрема; проаналізувати архітектуру та засоби безпеки операційної системи Android, розробити та реалізувати алгоритм для виявлення зловмисного програмного забезпечення шляхом статичного аналізу застосунку в операційній системі Android.

Наукова новизна полягає в наступному: запропоновано гнучкий спосіб виявлення шкідливого програмного забезпечення на основі статичного аналізу застосунку в операційній системі Android та реалізовано програму, котра використовує вище зазначений спосіб.

Практична цінність отриманих в роботі результатів полягає в тому, що запропонований спосіб дає змогу більш ефективно в плані гнучкості та детальності отриманої інформації проводити аналіз застосунків на предмет наявності ознак зловмисного програмного забезпечення в рамках операційної системи Android. Розроблене програмне забезпечення дає можливість аналізувати додатки та запобігти потраплянню зловмисних програм на персональний мобільний пристрій.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювались на XIV науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2021 (Київ, 17-19 листопада 2021 р.), на LXVII міжнародній науково-практичній конференція «Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення» (Тернопіль, 12 травня 2022 р.) та на міжнародній мультидисциплінарній науковій інтернет-конференції «Світ наукових досліджень. Випуск 18» (м.Тернопіль (Україна) - м.Переворськ (Польща), 20-21 квітня 2023 року).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, п'ятьох розділів та висновків.

У вступі подано загальну характеристику роботи, зроблено оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку досліджень, сформульовано мету і задачі досліджень, показано наукову новизну і практичну цінність роботи.

У першому розділі більш детально розглянуто постановку проблеми, проведено аналіз існуючих рішень та інструментів для аналізу безпеки

мобільних додатків, визначено основні вимоги до програмного забезпечення та виявлено певні закономірності у розробці програм, що базуються на статичному аналізі, для виявлення зловмисного програмного забезпечення в операційній системі Android. Запропоновано власний підхід до вирішення зазначеної проблеми.

У другому розділі розглянуто архітектуру та модель безпеки операційної системи Android з метою кращого розуміння існуючих механізмів та способу їх використання у виявленні зловмисного програмного забезпечення.

У *третьому розділі* розглянуто методики та стратегії статичного аналізу програмного забезпечення, їхні переваги та недоліки, а також вказано, наскільки зазначені методи можуть сприяти виявленню потенційних проблем з безпекою в мобільних додатках для ОС Android. Окрім того, розібрано алгоритм аналізу додатків в цій системі та представлено послідовність етапів цього процесу.

У четвертому розділі наведено запропонований спосіб виявлення зловмисного програмного забезпечення на базі статичного аналізу, описано його логіку та алгоритм роботи, а також детально розібрано кожен з його етапів.

У п'ятому розділі наведено результати тестування програмного забезпечення, котре реалізовує запропонований в даній роботі спосіб. Результати було наведено на прикладі повного аналізу тестового застосунку, а також на прикладі окремо розібраних тестових випадків.

У висновках представлено підсумок проведеної роботи.

Робота представлена на 83 аркушах та містить посилання на список використаних літературних джерел.

Ключові слова: статичний аналіз, зловмисне програмне забезпечення, вразливості, операційна система Android.

ABSTRACT

Actuality of theme. Mobile devices are an integral part of the everyday life of mankind. From social media to banking transactions, mobile devices are trusted by millions of people around the world. At the same time, there is an increasing trend in the number of malicious software that is relevant for all popular mobile platforms.

At the moment, the Android operating system is the most common mobile platform in the world. Due to this fact, countless malicious applications are deployed on it every year, with the aim of harming the device and gaining access to the user's confidential data. Based on the above information, the problem of detecting such applications is extremely relevant today and therefore requires new ideas and approaches.

The object of the study is a detection of malicious software in the Android operating system based on the method of static analysis of software.

The subject of the study is the software that implements the proposed method of detecting malicious software in the application of the Android operating system.

Purpose: to investigate the methodology for detecting malicious software and the method of static analysis in particular; analyze the architecture and security of the Android operating system, develop and implement an algorithm to detect malicious software by static analysis of the application in the Android operating system.

The scientific novelty is the following: a flexible method of detecting malicious software based on static analysis of an application in the Android operating system is proposed, and a program using the above-mentioned method is implemented.

The practical value of the results obtained in this work is that the proposed method makes it possible to analyze applications for the presence of signs of malicious software within the Android operating system more efficiently in terms of

flexibility and details of the information that is obtained during analysis. The developed software makes it possible to analyze applications and prevent malicious programs from entering a personal mobile device.

Approbation of work. The main provisions and results of the work were presented and discussed at the XIV scientific conference of undergraduates and graduate students "Applied Mathematics and Computing" PMK-2021 (Kyiv, November 17-19, 2021), at the LXVII International Scientific and Practical Conference "Information Society: technological, economic and technical aspects of formation "(Ternopil, May 12, 2022) and at the at the international multidisciplinary scientific Internet conference "The World of Scientific Research. Issue 18" (Ternopil (Ukraine) - Perevorsk (Poland), April 20-21, 2023).

Structure and scope of work. The master's dissertation consists of an introduction, five chapters and conclusions.

In the introduction, a general overview of the work is presented, an assessment of the current state of the problem is made, the relevance of the research direction is substantiated, the aim and objectives of the research are formulated, the scientific novelty and practical value of the work are demonstrated.

In the first section the statement of the problem is considered in more detailed way, an analysis of existing solutions and tools for analyzing the security of mobile applications is carried out, the main requirements for the software are defined, and certain regularities in the development of programs based on static analysis for the detection of malicious software in the Android operating system are identified. An own approach to solving the specified problem is proposed.

The second section examines the architecture and security model of the Android operating system in order to get better understanding of existing mechanisms and how they are used to detect malware.

In the third section methods and strategies of static software analysis are considered, their advantages and disadvantages are discussed, and it is indicated to what extent these methods can contribute to the detection of potential security issues

in mobile applications for the Android OS. In addition, the algorithm for analyzing applications in this system is dissected, and the sequence of stages of this process is presented.

In the fourth chapter, the proposed method of detecting malicious software based on static analysis is introduced, its logic and working algorithm are described, and each of its stages is thoroughly analyzed.

In the fifth chapter, the results of testing the software that implements the method proposed in this work are presented. The results were demonstrated using the example of a full analysis of a test application, as well as separately analyzed test cases.

The conclusion presents the results of the work.

The work is presented on 83 sheets and contains references to the list of used literature sources.

Keywords: static analysis, malware, vulnerabilities, Android operating system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

ОС – операційна система;

ПЗ – програмне забезпечення;

API - (Application Programming Interface) - це набір інструкцій, структур даних та інших елементів ПЗ, які використовуються іншим ПЗ за допомогою визначеного загального інтерфейсу;

APK (Android application package) – стандартний формат для зберігання та розповсюдження мобільних додатків для операційної системи Android;

Intent – об'єкт, який використовується в Android для взаємодії між різними компонентами додатка або між додатками;

Filtered intent – це Intent, котрий використовується для запуску компонентів додатка або додатків, які можуть відповісти на певний запит зі спеціальною фільтрацією.

Dalvik – віртуальна машина, котра використовується в ОС Android для виконання додатків.

URI (Uniform Resource Identifier) – рядок символів, котрий ідентифікує назву або місце зберігання ресурсу в Інтернеті чи в локальній мережі.

HAL (Hardware Abstraction Layer) – шар абстракції між операційною системою Android і апаратним обладнанням пристрою.

IPC (Inter-Process Communication) – технологія, що дозволяє процесам взаємодіяти між собою в операційній системі.

MAC (Mandatory Access Control) - тип контролю доступу до ресурсів системи, який використовується для забезпечення вищого рівня безпеки в комп'ютерних системах.

ВСТУП

В сучасному світі смартфонами користуються мільярди людей. Така кількість споживачів, кожен з яких має різні погляди, інтереси та захоплення, ініціює появу все більшої кількості різноманітних застосунків, котрі поширюються як на офіційних, так і на тіньових площадках, деякі з яких не завжди забезпечують достатній рівень захисту користувача. Це, в свою чергу, заохочує розробників зловмисного програмного забезпечення до пошуку вразливостей в найпопулярніших мобільних операційних системах з метою створення шкідливих застосунків, котрі дозволяли б отримати доступ до персональних пристроїв та даних користувачів.

Станом на початок 2023 року Android являється найбільш поширеною операційною системою націленою на кінцевого користувача, а також займає найбільшу частку ринку мобільних ОС у всьому світі, котра становить 71% від загального обсягу [1]. За усередненими даними кількість активних пристроїв, котрі працюють на основі ОС Android, наразі сягає близько 2,5 мільярдів одиниць щомісяця. Дані числа досягаються за допомогою різноманітності типів пристроїв, котрі можуть використовувати ОС Android (наприклад, телефони, планшети, телевізійні системи, системи розумного дому, автомобілі та інші), а також за допомогою існування широкого діапазону функцій, котрі надаються - спілкування, споживання медіа, розваги, контроль здоров'я та фізичних показників тіла, можливість використання Інтернет-сервісів.

Для забезпечення даних можливостей існує велика кількість застосунків, розроблених під ОС Android. Однак, разом з перевагами, кожна з цих програм являється потенційним джерелом шкоди для безпеки та конфіденційності користувача. Існує велика різноманітність вразливостей ОС Android, котрі можуть з’являтися на всіх рівнях безпеки ОС, починаючи з рівня додатків та закінчуючи рівнем ядра.

Типовими прикладами вразливостей ОС Android є: атаки типу «Відмова в обслуговуванні», переупакування програм для введення шкідливого коду,

зловживання дозволами та несанкціонований доступ між програмами. Все це може бути використано не тільки зловмисниками, з прямим наміром нанести шкоду користувачеві, але й доброчесними постачальниками послуг, ставши наслідком ненавмисних помилок в коді або недоліків архітектурного дизайну додатку. Так чи інакше наведені ситуації лише доводять той факт, що операційна система Android не є ідеально захищеною та має власні труднощі пов'язані з аспектом безпеки.

Одним з підходів до виявлення загроз у програмному забезпеченні є статичний аналіз коду. Статичний аналіз коду – це аналіз програмного забезпечення, що проводиться без реального виконання досліджуваних програм. Даний підхід ідеально підходить для перевірки мобільного застосунку на наявність потенційно шкідливих секцій коду з двох причин. По-перше, мобільні додатки поширюються у форматі архівованого виконуваного файлу застосунку, що дає змогу отримати доступ до коду додатку після певних маніпуляцій з ним. По-друге, даний підхід під час виконання не несе загрози пристрою та даним кінцевого користувача, так як потенційно шкідлива програма при цьому не запускається.

Метою цієї магістерської дисертації є детальніше дослідження моделі безпеки платформи Android, а також розробка та реалізація способу виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в даній операційній системі.

Ця дисертація буде актуальною для професіоналів та дослідників, які працюють у сфері кібербезпеки, мобільної безпеки та безпеки ОС Android, а також для організацій і окремих осіб, котрі користуються платформою Android. Дослідження може сприяти розробці більш якісних методів виявлення та запобігання поширенню шкідливого програмного забезпечення в екосистемі Android, тим самим підвищуючи загальну безпеку платформи.

1. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості та аналіз предметної області

Інтерес до галузі безпеки програмного забезпечення ОС Android ґрунтується на зростаючій загрозі, котра надходить від зловмисного ПЗ, що розробляється з прицілом на мобільні пристрої. Разом з широким використанням смартфонів і все більшою залежністю від мобільних пристроїв як для особистих цілей, так і для бізнесу, безпека цих пристроїв стала серйозною проблемою. Операційна система Android основною мішенню для зловмисників, зокрема через відкритість її коду та велику кількість пристроїв, на яких вона працює.

Шкідливе (зловмисне) програмне забезпечення може мати широкий спектр негативних наслідків для користувача, таких як викрадення конфіденційної інформації, порушення нормального функціонування пристрою та можливість віддаленого керування пристроєм зловмисником. Зростаюча складність шкідливого програмного забезпечення та труднощі з його виявленням зробили його серйозною проблемою для дослідників та практиків у сфері безпеки. Зловмисне програмне забезпечення може приймати різні види, включаючи віруси, трояни, шпигунське та рекламне ПЗ. Крім того, зловмисне програмне забезпечення може бути замаскованим або зашифрованим, що ускладнює його виявлення за допомогою традиційних інструментів безпеки.

Інструментом оцінки безпечності програми є використання методики аналізу ПЗ, котра базується на перевірці семантики коду програми (до або під час її виконання) з ціллю визначення наявності в програмі вразливостей або потенційно небезпечної поведінки. Дана методика аналізу ПЗ поділяється на дві категорії: динамічний та статичний аналіз.

Динамічний аналіз – це вид аналізу ПЗ, котрий застосовується під час виконання коду програми для спостереження за її поведінкою та виявлення відхилень. Даний тип аналізу має довгу історію використання у якості методу виявлення шкідливого програмного забезпечення, однак має кілька істотних

обмежень. Серед них можна виділити такі як потреба в контрольованому середовищі, трудомісткий характер аналізу та можливість шкідливого програмного забезпечення уникати виявлення шляхом визначення фіктивності середовища виконання та приховування шкідливої поведінки.

В той же час, статичний аналіз – це вид аналізу ПЗ, котрий передбачає перевірку коду програми без його виконання та показує широкі перспективи у ролі методу виявлення шкідливого програмного забезпечення. Під час дослідження коду статичний аналіз здатний виявляти моделі та типи поведінки, котрі є характерними для зловмисного програмного забезпечення, навіть якщо вони є замаскованими або зашифрованими. Крім того, статичний аналіз ПЗ може бути виконаний за короткий проміжок часу, не потребує контрольованого середовища та не потребує великого обсягу ресурсів пристрою (таких як центральний процесор, оперативна пам’ять, використання мережі та сховища), що робить його більш ефективним методом виявлення шкідливого програмного забезпечення, а також дає змогу проводити аналіз на мобільних пристроях із низькими фізичними характеристиками. Саме тому більшість існуючих робіт з виявлення зловмисного програмного забезпечення в ОС Android засновані саме на статичному аналізі застосунку.

Виходячи з вище зазначеної інформації тема безпеки програмного забезпечення в ОС Android є актуальної і на сьогоднішній день та користується попитом у людей, котрі залучені в сферу безпеки ПЗ. Це пов’язано з постійним розвитком предметної області, оскільки зловмисники постійно винаходять нові методи розповсюдження та впровадження шкідливого програмного забезпечення. Найбільш поширеним інструментом виявлення зловмисних застосунків є статичний аналіз коду ПЗ, котрий має багато переваг над його конкурентами у даному напрямку.

1.2. Аналіз та оцінка існуючих рішень

Серед наявних існуючих рішень найближчими за тематикою є DroidEnsemble, DroidSwan та Apposcopy. Надалі пропонується до ознайомлення детальніший розгляд кожного з цих підходів.

1.2.1. DroidEnsemble

У даній роботі автори представили програмне забезпечення під назвою DroidEnsemble [2], котре базується на статичному аналізі коду. Дане ПЗ використовує рядкові та структурні особливості застосунку з метою систематичного огляду та надання подальшої характеристики поведінки додатку Android. Таким чином, даний підхід дає змогу побудувати більш точну модель для виявлення зловмисних застосунків у ОС Android. Дана програма зчитує рядкові представлення застосунку, котрий аналізується, зокрема використані дозволи, апаратні функції, filtered intents, використання обмежених API викликів, шаблони коду, а також структурні особливості, такі як граф виклику функції. Після цього, з метою аналізу результату, застосовуються три алгоритми машинного навчання, а саме: Support Vector Machine (SVM) [3], k-Nearest Neighbor (kNN) [4] і Random Forest (RF) [5]. Загальну структуру ПЗ DroidEnsemble можна побачити на рисунку 1.

Процес роботи DroidEnsemble розділений на чотири кроки:

1. Отримання на вхід застосунку у форматі APK (Android application package).
2. Проведення двох-етапного статичного аналізу застосунку.
3. Створення трьох моделей машинного навчання.
4. Перевірка ефективності програми на тестових даних.

З метою аналізу цього програмного забезпечення в рамках даної магістерської роботи, достатньо розглянути лише другий крок, а саме проведення двох-етапного статичного аналізу застосунку.

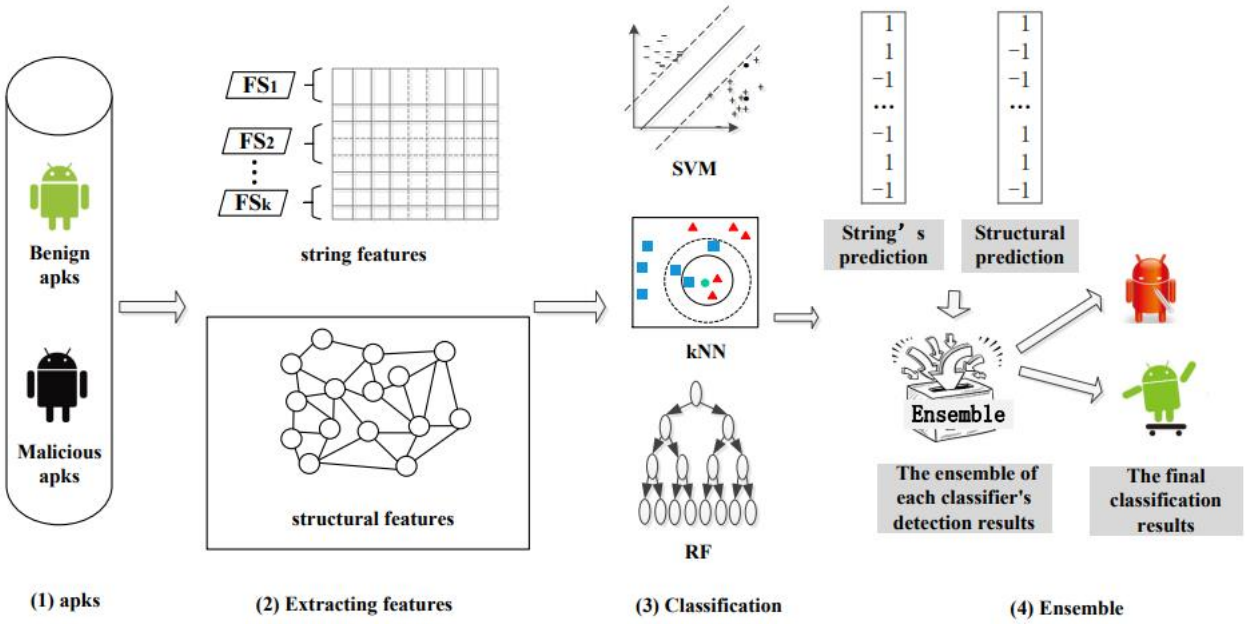


Рисунок 1 – Загальна структура роботи ПЗ DroidEnsemble

Першим етапом статичного аналізу застосунку є аналіз тексту програми. Під час даного етапу аналізуються наступні компоненти:

1. Запитувані дозволи. Кожна програма містить файл маніфесту, що містить мета-інформацію, котра використовується при встановленні та подальшому виконанні програми. Для аналізу автори використовують всі дозволи, котрі оголошені у файлі маніфесту додатку.
2. Характеристики обладнання. У системі Android вимоги до апаратного та програмного забезпечення визначають види доступу застосунків до системних ресурсів. Для аналізу автори використовують визначену у файлі маніфесту інформацію про апаратне та програмне забезпечення.
3. Filtered intents. Intents забезпечують зв'язок між компонентами застосунку шляхом обміну відповідними об'єктам. В цій роботі використовуються filtered intents, котрі представлені в маніфесті додатку елементами <intent-filter>.
4. Використання обмежених API викликів. Система дозволів ОС Android обмежує доступ до серії критичних викликів API, котрі визначають, як застосунок взаємодіє з системою/пристроєм.

5. Використані дозволи. Якщо програма робить запит на дозвіл на певну функціональність, то вона фактично отримує доступ до відповідних ресурсів пристрою.
6. Шаблони коду. ОС Android не надає належного механізму автентифікації та захисту від зовнішніх завантажених ресурсів. У такому разі виникають ризики при роботі з підключеними до застосунку бібліотеками, так як вони можуть бути модифіковані шкідливим кодом або приховувати небезпечні ділянки коду всередині. Тому відбувається перевірка, чи виконує програма команди оболонки ОС, чи вона динамічно завантажує зовнішні файли.

Другим етапом являється аналіз структури викликів. DroidEnsemble генерує граф викликів функцій програми, котрі будуються на основі інструкцій Dalvik. Вузли функцій класифікуються на ті, що викликають інший функціонал та ті, що були викликані ним. Кожному з них призначається хеш-код на основі підходу [6], що використовується в DroidEnsemble. Після цього використовуючи результати аналізу тексту програми обчислюється подібність між двома програмами, підраховуючи кількість однакового кодування між їх представленнями.

В даному програмному забезпеченні можна виділити певні недоліки.

По-перше, DroidEnsemble не має можливостей аналізу коду під час виконання. Деякі застосунки використовують методи обфускації, щоб запобігти вилученню функцій, або динамічно завантажують код, щоб перешкодити статичній перевірці. Щоб зменшити вплив відсутності динамічного аналізу DroidEnsemble витягує з додатків як рядкові, так і структурні особливості. У рядкових функціях перевіряється, чи програма динамічно завантажує зовнішні виконувані файли чи використовує код Linux, котрий може відображати деякі шкідливі дії. Однак, незважаючи на те, що структурні особливості компенсують певні недоліки задіяного підходу, вилучення структурних ознак займає дуже багато часу.

По-друге, точність алгоритмів, котрі спираються на машину, залежить від набору даних, який використовується для навчання моделі. Таким чином, якість набору даних важлива для визначення того, чи є моделі виявлення в цілому ефективними.

1.2.2. DroidSwan

DroidSwan [7] також являється програмним забезпеченням, котре при виявленні шкідливого ПЗ в ОС Android базується на статичному аналізі.

Так як застосунки для ОС Android встановлюються на пристрій за допомогою APK файлу, DroidSwan починає обробку застосунку саме з його аналізу. Даний файл декомпозується з метою виокремлення певних особливостей програми, котрі разом утворюють набір особливих рис ПЗ, який потім використовуватися під час класифікації.

На рисунку 2 показано, як саме відбувається декомпозиція файлу APK. Ресурси застосунку, такі як файли опису розміщення елементів інтерфейсу додатку та його зображення, зберігаються в каталозі «/res», котрий автори шкідливого ПЗ використовують для вставки зловмисних двійкових файлів (наприклад, файли з розширенням «.sh», «.elf» або «.exe») всередину зображень або інших файлів, котрі використовуються застосунком. У більшості випадків ці файли знаходять вбудованими у файли зображень (файли з розширенням «.jpg» та «.png»).

В свою чергу, файл «AndroidManifest.xml», котрий також знаходиться всередині APK файлу, містить назву, номер версії та права доступу до функцій системи, котрі має застосунок. Так як «AndroidManifest.xml» являється XML файлом, котрий зберігається у двійковому вигляді всередині APK файлу, DroidSwan використовує ПЗ ApkParser для його перетворення у файл XML придатний для читання.

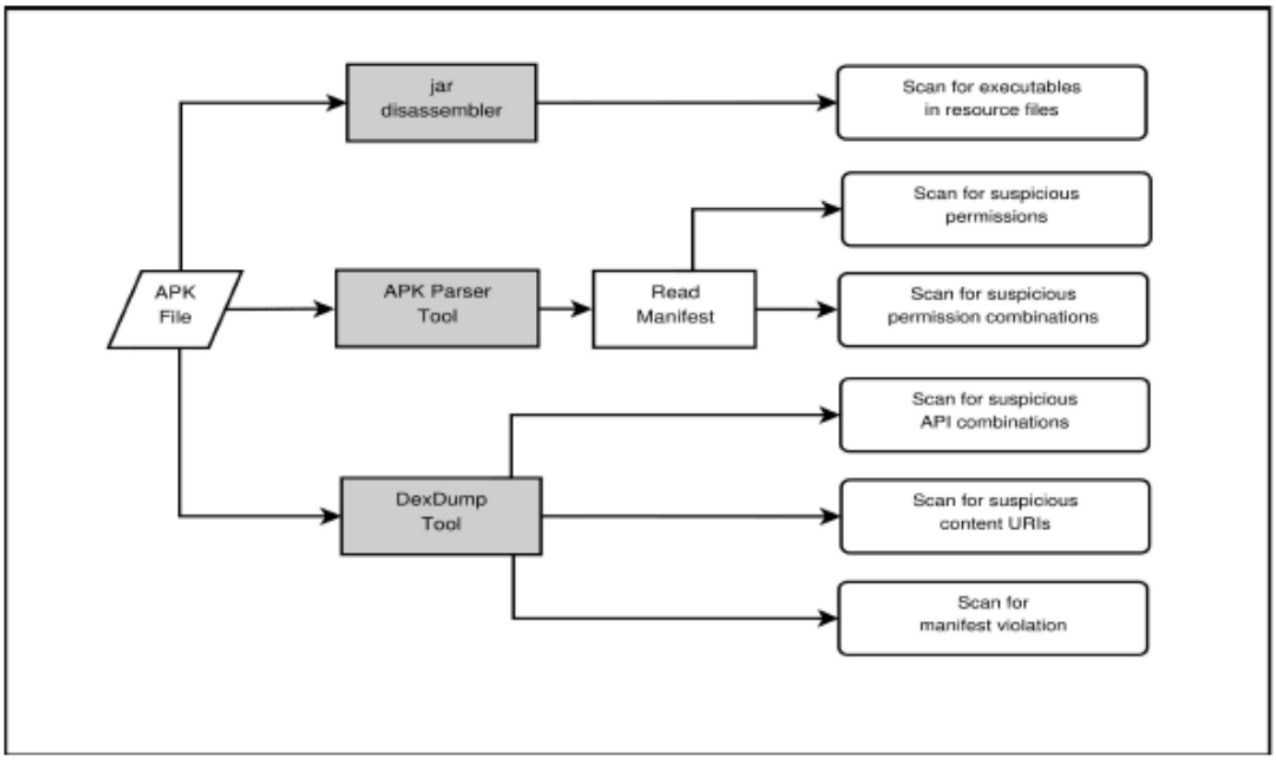


Рисунок 2 – Аналіз APK від DroidSwan

На першому етапі статичного аналізу застосунку, DroidSwan, аналогічно до DroidEnsemble, виконує аналіз програми на видані дозволи. Однак, в даному випадку до розгляду беруться лише певні комбінації дозволів, котрі на думку авторів дозволяють DroidSwan виявляти небезпечні секції застосунку, що становлять загрозу для даних та конфіденційності користувача.

На другому етапі відбувається перевірка викликів API ОС, які використовуються застосунком, котрий розглядається. Інформацію про виклики API, з метою подальшого їх аналізу, DroidSwan отримує шляхом декомпозиції файлу «classes.dex». Виклики API класифіковані авторами відповідно до їх використання програмою. Зі списку API, які зустрічаються у великій кількості зразків шкідливих програм, були отримані комбінації, які можуть становити загрозу для користувача. Розглядається два основних типи загроз – фінансові втрати та витік особистої інформації користувача.

Крім цього, DroidSwan виконує перевірку на надлишкові дозволи та на вміст URI. Вміст URI можна назвати підозрілим, якщо за допомогою цього URI є можливість отримати доступ до персональних даних користувача або до інших

даних цього застосунку. Наприклад, програма може отримати доступ до контактів користувача за допомогою наступного URI: «content://com.Android.contacts». Зразки URI такого типу DroidSwan вважає небезпечними.

На кожному етапі, кожному набору знайдених підозрілих ознак застосунку призначається вага, яка відображає вплив, який наявність чи відсутність того чи іншого набору має на класифікацію.

Процес роботи у DroidSwan поділяється на два етапи (рисунок 3):

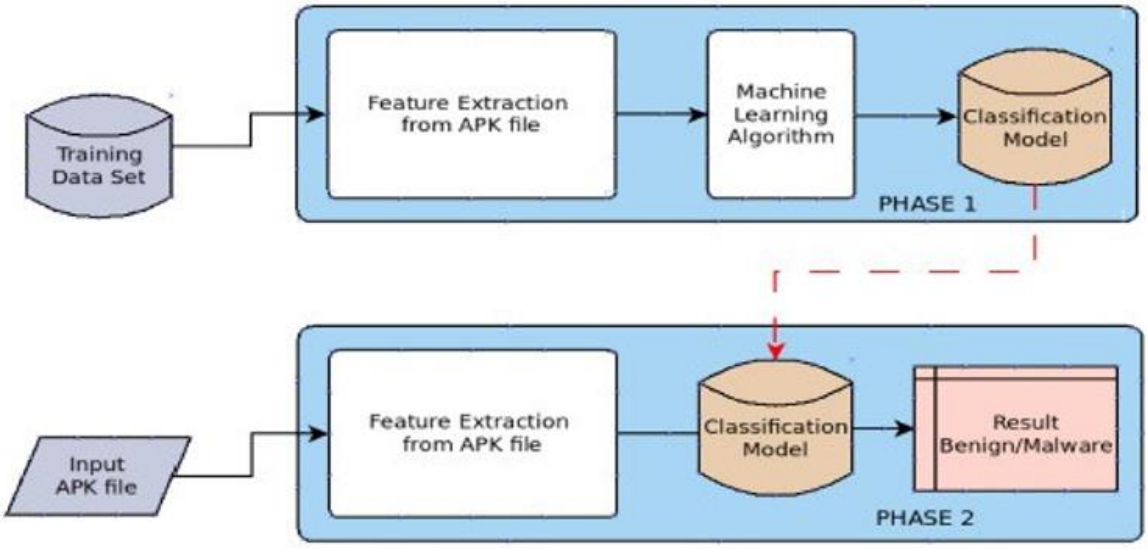


Рисунок 3 – Архітектура ПЗ DroidSwan

Етап перший – це фаза формування знань про застосунок. На цьому етапі DroidSwan виокремлює програмні компоненти та створює набір функцій зі зразків із навчального набору. Ці набори функцій потім подаються в алгоритм машинного навчання за допомогою реалізації алгоритмів машинного навчання WEKA, за допомогою чого створюється двокласна модель класифікатора. Дана модель може використовуватися для класифікації зразків без оновлення кожного разу, коли новий зразок надається для аналізу.

Етап другий – це фаза класифікації. На цьому етапі витягуються програмні особливості з застосунку, котрий необхідно класифікувати, і створюється відповідний набір функцій. Надалі цей набір функцій надається моделі

класифікації, створеної під час першої фази. У підсумку, модель класифікації виконує аналіз переданого набору та визнає його як зловмисний або доброякісний.

До недоліків даного підходу можна віднести постійну вагу для кожного окремого класу ознак. Це може призвести до некоректних результатів аналізу, так як, наприклад, одні дозволи можуть нести більшу загрозу, ніж інші і давати їм однакову вагу в такому разі не правильно. Крім цього, дозволи можуть працювати в комбінації з викликами API, переслідуючи одну зловмисну ціль. Даний момент упускається в розглянутій роботі.

1.2.3. Apposcopy

Останнім зі схожого програмного забезпечення для визначення шкідливого ПЗ в ОС Android є Apposcopy [8]. На відміну від розглянутих вище робіт, автори даного проекту пропонують спосіб виявлення шкідливих застосунків в ОС Android саме на основі семантики, використовуючи при цьому статичний аналіз.

Підхід Apposcopy заснований на семантиці має на меті виявлення шкідливих застосунків в ОС Android, котрі викрадають особисту інформацію користувача. Використовуючи уявлення про наявні переваги детекторів зловмисного ПЗ, що створені на основі шаблонів та аналізаторів, Apposcopy включає специфікацію мови високого рівня для опису семантичних характеристик сімейства шкідливих програм Android та статичний аналіз для прийняття рішень у випадку, коли розглянута програма відповідає сигнатурі сімейства шкідливих програм.

Мова специфікації на основі шаблонів, що пропонується Apposcopy, дозволяє визначити два типи семантичних властивостей програм у ОС Android – потік керування і потік даних. Прикладом семантичних властивостей потоку керування є випадок, коли зловмисне програмне забезпечення містить широкомовний приймач, котрий запускає службу після завершення деякої

системної події. Прикладом семантичних властивості потоку даних є випадок, коли зловмисне програмне забезпечення зчитує деякі приватні дані пристрою та надсилає їх зловмиснику певним способом (наприклад, за допомогою Інтернету, SMS-повідомлення тощо).

Щоб мати змогу визначати шаблони, описані у мові Apposcopy, статичний аналіз Apposcopy опирається на два ключових моменти. Перш за все, створюється нове представлення програми Android високого рівня, яке називається міжкомпонентним графіком викликів (inter-component callgraph - ICCG) та використовується для визначення того, чи відповідає програма Android властивостям потоку керування, зазначеним у шаблоні. По-друге, Apposcopy включає в себе статичний аналіз підозрілих ділянок коду, який використовується для визначення того, чи відповідає дана програма заданій властивості потоку даних.

Мова шаблонів шкідливого програмного забезпечення Apposcopy базується на мові програмування Datalog [9] (рисунок 4). Datalog — це декларативна логічна мова програмування, яка синтаксично є підмножиною мови програмування Prolog. В останні роки Datalog знайшов нове застосування в інтеграції даних, роботі з інформацією, мережах, аналізі програм, безпеці, хмарних обчисленнях і машинному навчанні.

```
<program> ::= <fact> <program> | <rule> <program> | ε
<fact> ::=  <relation> "(" <constant-list> ")."
<rule> ::= <atom> ":-" <atom-list> "."
<atom> ::= <relation> "(" <term-list> ")"
<atom-list> ::= <atom> | <atom> "," <atom-list>
<term> ::= <constant> | <variable>
<term-list> ::= <term> | <term> "," <term-list>
<constant-list> ::= <constant> | <constant> "," <constant-list>
```

Рисунок 4 – Синтаксис мови Datalog

В Apposcopy Datalog доповнюється вбудованими предикатами. Для кожного сімейства шкідливих програм користувач визначає унікальний предикат, який служить сигнатурою для цього сімейства шкідливих програм. Користувач може також визначити додаткові допоміжні предикати, що використовуються в підписі.

Щоб побудувати міжкомпонентний графік викликів і відстежувати потік інформації, Apposcopy починає з виконання аналізу вказівника, який обчислює набір абстрактних об’єктів області пам’яті, на які може вказувати кожна змінна. Оскільки точність базового аналізу вказівників є критичною для виявлення шкідливого програмного забезпечення з невеликою кількістю помилкових спрацювань, Apposcopy використовує контекстно-залежний аналіз покажчиків у стилі Андерсена [10].

Як і будь-яке рішення на основі шаблонів, Apposcopy не є ідеальним. Дуже важко розробити будь-яку схему на основі сигнатур, яку неможливо обійти належним чином налаштованим автоматичним інструментом. Зокрема, як і будь-яка система на основі статичного аналізу, Apposcopy можна обійти за допомогою таких методів, як динамічне завантаження коду та використання відображення в поєднанні зі зміною імен методів або класів. Однак такі спроби уникання виявлення зловмисних ділянок коду, ймовірно, будуть вважатися підозрілими та потребуватимуть подальшої перевірки.

По-друге, оскільки Apposcopy виконує глибокий статичний аналіз для виявлення семантичних властивостей програми, він може бути непридатним для сценаріїв, які вимагають миттєвого виявлення шкідливого програмного забезпечення.

1.3. Постановка задачі магістерської дисертації

Ціль даної роботи полягає в тому, щоб зробити внесок у сферу безпеки ОС Android, запропонувавши метод виявлення шкідливого програмного забезпечення на основі статичного аналізу програм в ОС Android. Запропонований метод повинен враховувати унікальні характеристики та проблеми ОС Android, включаючи використання байт-коду Dalvik, різноманітність конфігурацій апаратного та програмного забезпечення та децентралізований характер екосистеми додатків Android. Запропонований метод має бути ефективним та масштабованим, що дозволило б йому бути цінним інструментом для інших працівників даної сфери.

Крім того, запропонований метод також повинен мати на увазі такі виклики як потреба в точних і своєчасно оновлених базах даних зловмисного програмного забезпечення, труднощі з виявленням нового та невідомого зловмисного програмного забезпечення та частоту помилкових позитивних результатів існуючих методів.

У підсумку, запропонований метод виявлення зловмисного програмного забезпечення на основі статичного аналізу додатків в ОС Android має бути спрямований на усунення обмежень існуючих методів та сприяти кращому розумінню проблем і можливостей використання статичного аналізу для виявлення зловмисного програмного забезпечення. Результати цієї роботи повинні мати на меті не лише підвищення безпеки мобільних пристроїв, а й сприяння кращому розумінню проблем та можливостей використання статичного аналізу як методу виявлення шкідливого програмного забезпечення в ОС Android.

Висновки до першого розділу

В даному розділі було наведено загальні відомості та проведено аналіз предметної області даної роботи. Тема безпеки в операційній системі Android є надзвичайно актуальною відтоді як платформа зайняла передове місце по кількості пристроїв, котрі її використовують. Одним з інструментів забезпечення даного аспекту є статичний аналіз програмного забезпечення, котрий також користується популярністю як один з найкращих підходів до аналізу програмного коду з метою виявлення зловмисного ПЗ.

При аналізі існуючих рішень було виявлено певні закономірності у роботах, котрі базуються на статичному аналізі ПЗ. Наприклад, обов’язковими для аналізу частинами кожного застосунку є перевірка набору його дозволів та послідовностей викликів всередині програми. Це спричиняє наявність схожих мінусів у кожного з проаналізованих підходів. У інших компонентах аналізу автори пропонують власний підхід для аналізу отриманих даних та їх оцінці на наявність шкідливого програмного забезпечення.

Також, у даному розділі було поставлено задачу розробити метод виявлення зловмисного програмного забезпечення на основі статичного аналізу у застосунках ОС Android, котрий покликаний покращити певні аспекти існуючих методі, а також підвищення загальний рівень безпеки мобільних пристроїв.

2. АРХІТЕКТУРА ОПЕРАЦІЙНОЇ СИСТЕМИ ТА МОДЕЛЬ БЕЗПЕКИ ПЛАТФОРМИ ANDROID

2.1. Архітектура операційної системи Android

Android – це операційна система, що розробляється та підтримується компанією Google. ОС Android створена на основі ядра Linux та складається з проміжного програмного забезпечення, бібліотек та API, написаних на мові програмування С, а також з прикладного програмного забезпечення, яке включає в себе бібліотеки, сумісні з мовою програмування Java. Вихідний код Android розповсюджується компанією Google під ліцензіями відкритого вихідного коду.

Архітектура ОС Android побудована як набір різних шарів. Кожен шар має свої цілі та обов'язки. Верхня шар — це програми, котрі відносяться до простору взаємодії з користувачем, такі як стандартні програми та програми, встановлені користувачем. В свою чергу нижній шар – це шар роботи ядра.

Загальна архітектура ОС Android (рисунок 5) складається з шести основних компонентів (знизу вверх) [11]:

- ядро Linux;
- рівень апаратної абстракції (HAL);
- сирцеві бібліотеки C/C++;
- середовище виконання Android;
- фреймворк Java API;
- системні та користувацькі застосунки.

ОС Android побудована на ядрі Linux, що дозволяє переносити її різні платформи за допомогою апаратної абстракції. Сирцеві бібліотеки Android на мовах програмування С та С++ компілюються апаратною архітектурою, що використовується пристроєм, та знаходяться над ядром Linux. Рівень самої платформи Android складається з віртуальної машини Dalvik (DVM) та Java бібліотек. Рівень застосунків платформи дозволяє контролювати життєвий цикл програм, обмінюватися даними між ними, надає відомості про такі дані як, наприклад, географічне розташування, а також керує сповіщеннями.

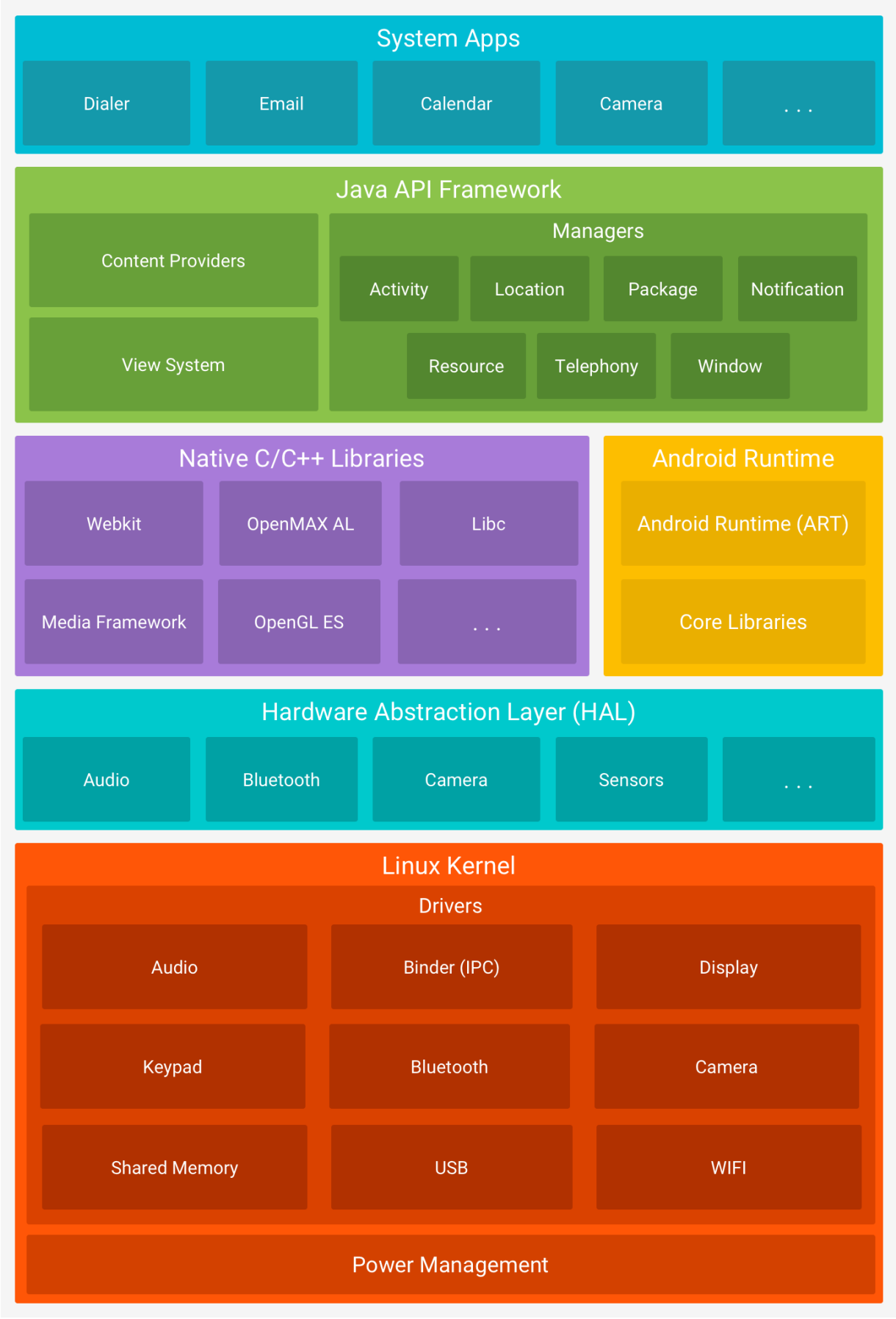


Рисунок 5 – Архітектура ОС Android

Ядро. Ядро Linux є фундаментальним компонентом усієї системи. Дана частина налаштована спеціально для взаємодії з пристроєм, що складається з обмежених ресурсів. Вся операційна система Android побудована на основі ядра Linux з деякими подальшими архітектурними змінами, внесеними компанією Google. Цей шар також діє як рівень абстракції між апаратним та програмним рівнями. Ядро Linux надає основні системні функції, такі як керування процесами, управління пам'яттю та керування пристроями. Ядро Linux також надає набір драйверів, які роблять можливим процес взаємодії ОС Android із периферійними пристроями. Використання ядра Linux дозволяє Android використовувати переваги ключових функцій безпеки та надає можливість виробникам пристроїв розробляти драйвера для стандартизованого ядра.

Рівень апаратної абстракції. Рівень апаратної абстракції (HAL) надає стандартні інтерфейси, які дають змогу для фреймворку Java API взаємодіяти з апаратною частиною пристрою. HAL складається з кількох бібліотечних модулів, кожен з яких реалізує інтерфейс для певного типу апаратного компонента, такого як камера або модуль Bluetooth. Коли API фреймворку здійснює виклик для доступу до апаратного забезпечення пристрою, система Android завантажує модуль бібліотеки для цього апаратного компонента.

Сирцеві бібліотеки C/C++. Дані бібліотеки розроблені для взаємодії з ядром Linux і тому написані мовами програмування C/C++. Вони допомагають обробляти дані вищого порядку перетворюючи їх в більше прості структури для їх подальшого аналізу ядром Linux. Серед сирцевих бібліотек можна знайти багато таких, що реалізовують різні корисні функції, які потім використовує ОС Android. Наприклад, існують наступні сирцеві бібліотеки, котрі можна зустріти в ОС Android: для безпечного доступу в Інтернеті використовуються бібліотеки SSL, для надання різних медіа-кодеків використовується медіа-фреймворк, для показу 2D або 3D графічного вмісту використовується бібліотека OpenGL, для зберігання даних застосунку використовується двигун бази даних SQLite, а для відображення вмісту HTML використовується механізм веб-браузера WebKit.

Середовище виконання Android. Ключовим елементом середовища виконання Android являється Dalvik Virtual Machine (DVM), яка є інтерпретатором байт-коду, що був розроблений на основі віртуальної машини Java (JVM), котру вдосконалили спеціально для операційної системи Android з урахуванням особливостей мобільних пристроїв. DVM не може запускати байт-код Java (файли з розширенням «.class») безпосередньо. Рідним форматом даних, що подаються на вхід для DVM, є Dalvik Executable (DEX). Йому відповідають дані, упаковані у файли з розширенням «.dex». У свою чергу, файли з розширенням «.dex» упаковуються або в системні бібліотеки Java (файли JAR), або в програми Android безпосередньо. Серед основних функцій ядра Linux DVM використовує такі переваги, як керування пам’яттю, багатозадачне середовище виконання та багатопотокове виконання, що дуже важливо для мови Java. DVM забезпечує контроль над застосунками та їх запуск у формі процесу із власною віртуальною машиною та ресурсами. Так як DVM використовує JVM, яка надає користувачам групу Java API та бібліотек для проектування та створення застосунків для ОС Android, їх розробка відбувається переважно за допомогою мови програмування Java.

Фреймворк Java API. На рівні фреймворку Java API існує група служб, які спільно формують середовище, в якому контролюються та виконуються програми. Це реалізовує концепцію створення застосунків для ОС Android на основі багаторазових, взаємозамінних та гнучких компонентів. Крім цього дана концепція створює умови для повторного використання даних та механізмів одного застосунку іншими. Рівень фреймворку Java API включає в себе наступні ключові служби:

- менеджер активності – адмініструє стек активності та життєвого циклу програми;
- постачальник вмісту – дозволяє програмам розкривати інформацію та обмінюватися інформацією з іншими програмами;
- менеджер ресурсів – надає доступ до вбудованих ресурсів, таких як тексти, кольори та макети інтерфейсу користувача;

- менеджер сповіщень – керує сповіщеннями та їх передачею користувачу;
- система представлень – розширений набір представлень, який використовується для створення інтерфейсу застосунків;
- менеджер пакетів – інструмент, за допомогою якого програми можуть ідентифікувати інформацію про інші програми, які зараз встановлені на пристрої;
- менеджер телефонії – надає програмі інформацію про послуги телефонії, доступні на пристрої, наприклад інформацію про стан зв'язку пристрою абонента;
- менеджер локації – надає доступ до служб локації, дозволяючи програмі отримувати оновлення про зміну місцезнаходження пристрою користувача;

Системні та користувацькі застосунки. Найвищий рівень архітектури в ОС Android складається безпосередньо з застосунків. За замовчуванням ОС Android надає встановлений набір основних програм, таких як електронний клієнт, SMS, веб-браузер, календар, менеджер контактів, карти тощо. Дані програми можуть взаємодіяти з іншими встановленими програмами незалежно від того, хто є їх розробником. Зазвичай застосунки розробляються з використанням мови програмування Java або Kotlin. Дані та ресурси додатку архівуються у файл пакету Android за допомогою комплекту розробки програмного забезпечення (SDK). Цей файл розглядається як один застосунок і використовується для встановлення додатку на будь-якому пристрої з ОС Android. Після встановлення на пристрій, ОС Android розглядає кожну програму як окремого користувача, що робить ОС Android багатокористувацькою системою Linux. Система призначає застосунку ідентифікатор користувача та встановлює дозволи на всі файли, які потрібні програмі. Це робиться для того, щоб тільки користувач саме з цим ідентифікатором мав доступ до них. Так як кожна програма працює у своєму власному процесі Linux і кожен процес має свою віртуальну машину, то код програми виконується незалежно від інших програм. За замовчуванням кожна

програма має доступ лише до своїх компонентів і не може отримати доступ до жодної частини системи без дозволу. Однак застосунок може отримати доступ до інших застосунків системних служб [12].

2.2. Структура та процес створення файлу застосунку в ОС Android

Застосунки для ОС Android розповсюджуються як файли з розширенням APK (Android Package Kit). Файли APK в загальному являються архівними файлами, подібними до файлів JAR, які використовуються для пакування бібліотек Java. Файл APK містить ключові компоненти додатку, такі як код застосунку у форматі файлу DEX, бібліотеки, ресурси, тощо.

Детальніше вміст пакету APK зображений на рисунку 6 [13]:

Рисунок 6 – Структура APK файлу

Файл «AndroidManifest.xml». Це маніфест застосунку, що представлений файлом у форматі XML. Він містить метадані застосунку – наприклад, його назву, версію, дозволи, що потрібні для запуску цього застосунку та дозволи, необхідні для доступу до його інформації іншим додаткам. Інформація, що міститься в цьому файлі, використовується для визначення чи дозволено встановлення застосунок на даному пристрої. Для цього перевіряються розміри екрану, доступне апаратне обладнання та інші речі (наприклад, наявність сенсорного екрану).

Каталог «assets/». Дана директорія використовується для зберігання будь-яких ресурсів, що не являються кодом програми, котрий виконується. Найчастіше це файли шрифтів або дані гри, такі як мапи рівнів та текстури, а також будь-які інші дані на кшталт відео, аудіо, документів, тощо.

Файл «resources.arsc». Файл, що містить у собі всі попередньо скомпільовані ресурси, такі як тексти, кольори або стилі.

Каталог «res/». Це директорія з усіма ресурсами, які не скомпільовані у файл «resources.arsc». Даний каталог містить більшість ресурсів застосунку у форматі XML (наприклад, макети інтерфейсу) і малюнків (наприклад, PNG, JPEG) у піддиректоріях з різними позначеннями, такими як «-mdpi» і «-hdpi», котрі відповідають щільності пікселів на екрані пристрою, «-sw600dp» або «-large», котрі відповідають розмірам екрану та «-en», «-de», «-pl», котрі необхідні для збереження локалізованих текстів застосунку. Будь-які файли у форматі XML у каталозі «res/» файлу APK перетворюються в більш компактне двійкове представлення під час компіляції, аналогічно до малюнків.

Каталог «lib/». Це директорія зі скомпільованими сирцевими бібліотеками, які використовуються застосунком. Даний каталог містить у собі кілька інших – по одній директорії для кожної підтримуваної архітектури центрального процесору. Будь-які бібліотеки (файли з розширенням «*.so») будуть розміщені саме в тих піддиректоріях, названих на честь архітектури процесора (наприклад, «x86», «x86_64», «armeabi-v7a»), на які вони націлені.

Каталог «META-INF/». Це директорія з метаданими APK файлу, такими як його підпис. Ця папка існує лише в підписаних APK файлах і містить список усіх внутрішніх APK файлів з їхніми підписами. Наразі підписання застосунку в Android працює так, що система перевіряє підписи щодо нестисненого вмісту файлу з архіву один за іншим.

Файл «classes.dex». Файл із кодом програми, що зберігається у виді байт-коду. У випадку використання підходу multidex у APK файлі можуть існувати додаткові файли з розширенням DEX (з назвою «classes2.dex» тощо).

Процес створення APK файлу складається з п’яти етапів:

1. Процес починається з вихідного коду програми, написаної мовою програмування Java або Kotlin, разом з відповідним залежними бібліотеками. Даний вихідний код компілюється в байт-код Java за допомогою відповідного компілятора. Варто зауважити, що залежні бібліотеки застосунку вже скомпільовані і не потребують додаткової обробки на даному етапі. В даний перелік входять як залежності Android, що поставляються як файли формату AAR, так і залежності, що стосуються лише мови Java (файли формату JAR).

2. Байт-код Java, створений на попередньому етапі разом з усіма залежними бібліотеками, перетворюється у файл з розширенням DEX, що містить байт-код Dalvik, який може запускатися на пристроях з операційною системою Android. Це створює файл «classes.dex» (або декілька файлів «classes*.dex», якщо застосовується підхід multidex).

3. Ресурси програми архівуються за допомогою Android Asset Packaging Tool. Цей інструмент перетворює всі ресурси застосунку у двійковий формат, оптимізований для використання в ОС Android. Результатом є файл APK з усіма ресурсами, крім коду.

4. Отриманий на другому етапі код програми з усіма файлами «classes.dex» та сирцевими бібліотеками запаковується у файл APK.

5. Проводиться оптимізація файлу APK, що гарантує, що всі нестиснені дані у цьому файлі вирівнюються за чотирьохбайтовою межею, щоб покращити продуктивність під час доступу до великих даних, таких як зображення.

6. На останньому етапі файл APK підписується цифровим підписом, що дає змогу перевірити та встановити його на будь-якому пристрої Android.

2.3. Основні компоненти застосунку в ОС Android

ОС Android надає платформу з відкритим вихідним кодом та середовище для виконання програм на мобільних пристроях. Основними будівельними блоками будь-якого застосунку в ОС Android є:

- «AndroidManifest.xml»: файл «AndroidManifest.xml» — це керуючий файл, який повідомляє системі, що робити з усіма компонентами верхнього рівня (зокрема, Activity, Service та Broadcast Receiver описаними нижче) у програмі. Даний файл також визначає, які дозволи необхідні для роботи застосунку.
- Activity: загалом Activity — це код для кожної окремої дії, орієнтованої на користувача. Зазвичай це включає в себе відображення інтерфейсу для користувачів, але не завжди – деякі Activity ніколи не відображають інтерфейс користувача. Прийнято, що одна з наявних у застосунку Activity повинна бути вхідною точкою програми.
- Service: Service — це програмний об’єкт, який працює у фоновому режимі. Він може виконуватися як у власному процесі, так і в контексті процесу іншої програми. Інші компоненти застосунку «прив'язуються» до Service та викликають у ньому методи за допомогою віддалених викликів процедур. Прикладом Service є медіапрогравач: навіть коли користувач виходить із інтерфейсу вибору медіа, користувач, ймовірно, все ще має намір продовжувати відтворення музики. Service підтримує музику навіть після завершення роботи інтерфейсу.
- Broadcast Receiver: BroadcastReceiver – це програмний об’єкт, який створюється, коли операційна система або інша програма видає механізм, відомий як Intent. Наприклад, програма може зареєструвати приймача для повідомлення про низький заряд акумулятора та змінити свою поведінку на основі цієї інформації.

2.4. Модель безпеки платформи Android на рівні операційної системи

Одним з двох рівнів моделі безпеки платформи Android є рівень операційної системи. Основними складовими даного рівня є інструменти забезпечення безпека ядра системи та засоби між процесного зв'язку, або IPC (inter-process communication). IPC використовується для забезпечення безпечного зв'язку між програмами, що працюють у різних процесах. Зазначені складові безпеки на рівні операційної системи гарантують, що простір виконання будь-якого коду обмежується лише приватним простором програми. Незалежно від того, чи викликається код під час роботи запущеного застосунку, чи має інше походження, дана система призначена для запобігання шкоди іншим програмам, системі Android або самому пристрою [14].

2.4.1. Рівень ядра

Основою платформи Android є ядро Linux. Ядро Linux широко використовується протягом багатьох років і використовується в мільйонах чутливих до безпеки середовищ. За свою історію, коли тисячі розробників постійно досліджували, атакували та виправляли, Linux став стабільним і безпечним ядром, якому довіряють багато корпорацій та фахівців з безпеки.

Як основа для мобільного обчислювального середовища, ядро Linux надає Android кілька ключових функцій безпеки, зокрема:

- модель дозволів на основі ідентифікаторів користувачів;
- ізоляція процесів;
- розширюваний механізм для безпечного між процесного зв'язку;
- можливість видалення непотрібних і потенційно небезпечних частин ядра;

Будучи багатокористувацькою операційною системою, основною метою організації безпеки ядра Linux є ізоляція ресурсів користувачів один від одного. Саме тому ОС Linux:

- забороняє користувачу А читати файли користувача В;
- гарантує, що користувач А не вичерпує пам'ять користувача В;
- гарантує, що користувач А не вичерпує ресурси ЦП користувача В;
- забезпечує, щоб користувач А не виснажував пристрої користувача В (наприклад, телефонію, GPS і Bluetooth);

2.4.2. Середовище виконання застосунків

Безпека додатків Android забезпечується приватним простором застосунку, який ізолює додатки один від одного та захищає їх та систему від шкідливих програм.

Платформа Android використовує переваги користувацького захисту Linux для ідентифікації та ізоляції ресурсів програми. Це ізолює програми один від одного та захищає програми та систему від шкідливих програм. Для цього Android призначає унікальний ідентифікатор користувача (UID) кожній програмі Android і запускає її в власному процесі.

ОС Android використовує UID для налаштування приватного простору програми на рівні ядра. Ядро забезпечує безпеку у взаємодії між програмами та системою на рівні процесу за допомогою стандартних засобів Linux, таких як ідентифікатори користувачів і груп, які призначаються програмам. За замовчуванням програми не можуть взаємодіяти один з одним і мають обмежений доступ до ОС. Якщо додаток А намагається зробити щось шкідливе, наприклад, прочитати дані програми В або набрати номер телефону без дозволу, йому заборонено це зробити, оскільки він не має відповідних привілеїв користувача за замовчуванням. Приватний простір застосунку є досить простим механізмом, котрий піддається аудиту та заснований на десятилітньому досвіді розділення користувачів у стилі UNIX між процесами та дозволами на файли.

Оскільки приватний простір застосунку знаходиться в ядрі, ця модель безпеки поширюється як на власний код, так і на програми ОС. Усе програмне забезпечення над ядром, таке як бібліотеки ОС, фреймворки додатків,

середовище виконання програми та всі програми, запускаються в окремому приватному середовищі. На деяких платформах розробники обмежені певною структурою розробки, набором API або мовою програмування. В ОС Android же немає обмежень щодо написання програм, необхідних для забезпечення безпеки; у цьому відношенні «рідний» для ОС код настільки ж закритий, як і інтерпретований код.

Як правило, щоб вийти з приватного простору застосунку на належним чином налаштованому пристрої, потрібно поставити під загрозу безпеку ядра Linux. Однак, як і інші функції безпеки, індивідуальні засоби захисту, що застосовуються, не є невразливими, тому поглиблений захист надає достатньо механізмів для запобігання того, що окремі вразливості призведуть до компрометації ОС або інших програм [15].

2.4.3. Міжпроцесна комунікація

Ізоляція процесів покращує безпеку програм, які запущені на пристрої. Тим не менш, є деякі випадки, коли одному процесу може знадобитися надати певну послугу іншим процесам. ОС Android пропонує механізм між процесного зв’язку (IPC) за допомогою віддаленого виклику процедур (RPC), в якому метод викликається за допомогою Intent або іншим компонентом програми, але виконується в іншому процесі, при цьому будь-який результат повертається назад до сторони, що зробила виклик. Intent надає можливість зв’язувати код з різних програм під час виконання.

Зокрема, екземпляри класу Intent містять інформацію, яку ОС Android використовує, щоб вибрати відповідний компонент для використання, наприклад точне ім’я компонента або категорію компонента для отримання наміру. Вони також містять інформацію, яка визначає загальну дію, яку потрібно виконати, будь то перегляд чи вибір дії. Наприклад, якщо програма хоче переглянути веб-сторінку, вона виражає свій «намір» переглянути URL-адресу, створюючи екземпляр класу Intent та передає його системі. Система знаходить

інший фрагмент коду, у даному випадку веб-браузер, який знає, як поводитися з цим наміром, і запускає його. Використовуються дві форми намірів, а саме явні та неявні. Перші вказують компонент, який надає точний клас для запуску. Останні, неявні наміри, не вказують безпосередньо програмний компонент; замість цього вони повинні включати достатньо інформації, щоб система могла визначити, який із доступних компонентів найкраще використовувати для цього.

2.4.4. Покращена модель безпеки ОС Linux

Як частину моделі безпеки ОС Android використовує Security-Enhanced Linux (SELinux) для забезпечення обов'язкового контролю доступу (MAC) до всіх процесів, навіть тих, що виконуються з правами root/superuser. Багато компаній та організацій зробили внесок у впровадження SELinux для Android. За допомогою SELinux Android може краще захищати й обмежувати системні служби, контролювати доступ до даних програм і системних журналів, зменшувати вплив шкідливого програмного забезпечення та захищати користувачів від потенційних недоліків у коді на мобільних пристроях.

Security Enhanced Linux (SELinux) — це обов'язкова система контролю доступу (MAC) для операційної системи Linux. Як система MAC, вона відрізняється від знайомої для Linux системи контролю доступу (DAC). У системі DAC існує концепція власності, згідно з якою власник певного ресурсу контролює пов'язані з ним права доступу. В той же час, система MAC консультується з центральним органом для прийняття рішення щодо всіх спроб доступу.

SELinux був реалізований як частина модуля безпеки Linux (LSM), який розпізнає різні об'єкти ядра та конфіденційні дії, що виконуються з ними. У момент, коли буде виконано кожну з цих дій, викликається функція LSM, щоб визначити, чи слід дозволити дію на основі інформації про неї, що зберігається в прихованому об'єкті безпеки. SELinux забезпечує реалізацію цих функцій і

керування цими об’єктами безпеки, які поєднуються з власною політикою, щоб визначити рішення щодо доступу.

Поряд з іншими заходами безпеки Android політика контролю доступу Android значно обмежує потенційну шкоду від пристроїв, що піддалися впливу зловмисного ПЗ, та облікових записів. Використання таких інструментів, як дискреційний та обов’язковий контроль доступу в ОС Android, надає структуру, яка гарантує, що програмне забезпечення працює лише на мінімальному рівні привілеїв. Це пом’якшує наслідки атак і зменшує ймовірність перезапису помилкових процесів або навіть передачі даних.

SELinux працює за принципом відмови за замовчуванням: все, що явно не дозволено – заборонено. SELinux може працювати в двох глобальних режимах:

- Вільний режим, у якому відмови в дозволах реєструються, але не застосовуються.
- Примусовий режим, у якому відмови в дозволах реєструються та застосовуються.

ОС Android включає SELinux у примусовому режимі та застосовує відповідну політику безпеки, яка працює за замовчуванням у AOSP. У примусовому режимі заборонені дії не виконуються, а всі спроби порушення реєструються ядром у вбудованих функціоналах – dmesg і logcat.

SELinux також підтримує вільний режим для кожного домену, в якому окремі домени (процеси) можна зробити дозволеними, а решту системи перевести в режим глобального застосування. Домен — це просто мітка, що ідентифікує процес або набір процесів у політиці безпеки, де всі процеси позначені тим самим доменом і обробляються політикою безпеки однаково [16].

Однак, ОС Android не використовує всі функції, надані SELinux:

- Більшість політик в AOSP визначаються за допомогою мови політики ядра. Існують деякі винятки для використання, такі як Common Intermediate Language (CIL).

- Ролі SELinux та контроль доступу на основі ролей. Використовуються лише дві ролі за замовчуванням: одна для суб’єктів та інша для об’єктів.
- Чутливість SELinux. За замовчуванням завжди встановлено чутливість s0.
- Логічні значення SELinux. Після створення політики для пристрою вона не залежить від стану пристрою. Це спрощує перевірку та налагодження політик.

2.4.5. Захищене завантаження системи

Захищене завантаження системи намагається гарантувати, що весь виконуваний код надходить із надійного джерела (зазвичай OEM-виробників пристроїв), а не від зловмисного виробника чи невідомого джерела. В ОС Android це реалізовано наступним чином: встановлюється повний ланцюг довіри, починаючи від апаратно захищеного довіреного кореня до завантажувача, завантажувального розділу та інших перевірених розділів, включаючи розділи системи, постачальника та, за бажанням, OEM (оригінальний виробник обладнання). Під час завантаження пристрою кожен етап перевіряє цілісність і справжність наступного етапу перед тим, як передати виконання. Схему захищеного завантаження ОС Android можна побачити на рисунку 7.

Крім стандартної перевірки на те, що на пристроях працює справна та безпечна версія ОС Android, захищене завантаження системи також перевіряє правильність версії Android із захистом від відкату. Захист від відкату допомагає запобігти встановлення старішої версії системи, забезпечуючи оновлення пристроїв лише до новіших версій ОС Android. На додаток до перевірки версій ОС, захищене завантаження системи також дозволяє пристроям з ОС Android повідомляти користувача про свій стан цілісності [17].

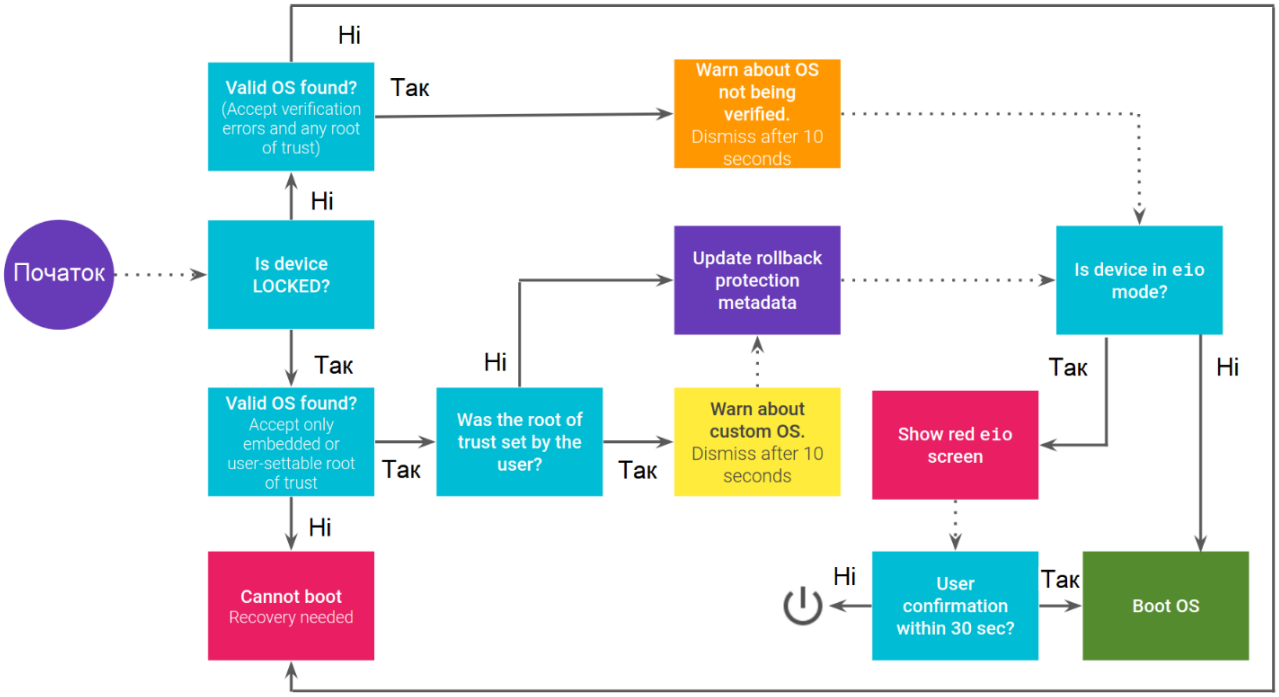


Рисунок 7 - Схема захищеного завантаження ОС Android

2.4.6. Адміністрування пристрою

За замовчуванням в ОС Android лише ядро та невелика підмножина основних програм запускаються з правами root. Android не забороняє користувачеві чи програмі з правами root змінювати операційну систему, ядро чи будь-яку іншу програму. Загалом, root має повний доступ до всіх програм і всіх даних програми. Користувачі, які змінюють дозволи на пристрої Android, щоб надати додаткам root-доступ, підвищують безпеку шкідливих програм і потенційних недоліків програм.

Можливість модифікувати власний пристрій Android важлива для розробників, які працюють із платформою Android. На багатьох пристроях Android користувачі мають можливість розблокувати завантажувач, щоб дозволити встановлення альтернативної операційної системи. Ці альтернативні операційні системи можуть дозволити власнику отримати root-доступ для цілей налагодження програм і системних компонентів або доступу до функцій, які не надаються додаткам API Android.

На деяких пристроях людина з фізичним контролем пристрою та кабелем USB може встановити нову операційну систему, яка надає користувачеві права root. Щоб захистити будь-які наявні дані користувача від компрометації, механізм розблокування завантажувача вимагає, щоб завантажувач стер усі наявні дані користувача як частину кроку розблокування. Root-доступ, отриманий через використання помилки ядра або діри в безпеці, може обійти цей захист.

Шифрування даних за допомогою ключа, що зберігається на пристрої, не захищає дані програм від користувачів root. Програми можуть додати рівень захисту даних за допомогою шифрування за допомогою ключа, що зберігається поза пристроєм, наприклад, на сервері або пароля користувача. Такий підхід може забезпечити тимчасовий захист, поки ключ відсутній, але в певний момент ключ повинен бути наданий програмі, а потім він стає доступним для користувачів root.

Більш надійний підхід до захисту даних від root-користувачів – використання апаратних рішень. OEM-виробники можуть вирішити впровадити апаратні рішення, які обмежують доступ до певних типів вмісту, наприклад DRM для відтворення відео, або довіреного сховища, пов’язаного з NFC, для гаманця Google.

У разі втрати пристрою шифрування файлової системи в ОС Android використовує пароль пристрою для захисту ключа шифрування, тому зміни завантажувача або операційної системи недостатньо для доступу до даних користувача без пароля пристрою користувача.

2.4.7. Шифрування

Шифрування – це процес кодування всіх даних користувача на пристрої з ОС Android за допомогою симетричних ключів шифрування. Після шифрування пристрою всі дані, створені користувачами, автоматично шифруються перед записом на диск, а всі зчитування автоматично розшифровуються перед

поверненням у процес виклику. Шифрування гарантує, що навіть якщо неавторизована сторона спробує отримати доступ до даних, вони не зможуть їх прочитати.

Android має два способи шифрування пристрою: шифрування на основі файлів і шифрування на весь диск.

Шифрування на основі файлів. Android 7.0 і новіших версій підтримують шифрування на основі файлів. Шифрування на основі файлів дозволяє шифрувати різні файли різними ключами, які можна розблокувати незалежно. Пристрої, які підтримують шифрування на основі файлів, також можуть підтримувати пряме завантаження, що дозволяє зашифрованим пристроям завантажуватися безпосередньо на заблокований екран, забезпечуючи швидкий доступ до важливих функцій пристрою, таких як служби доступності та сигналізації.

Завдяки шифруванню на основі файлів і API, які інформують програми про шифрування, програми можуть працювати в обмеженому контексті. Це може статися до того, як користувачі нададуть свої облікові дані, зберігаючи при цьому особисту інформацію користувача.

Android 9 пропонує підтримку шифрування метаданих, де є апаратна підтримка. За допомогою шифрування метаданих один ключ, присутній під час завантаження, шифрує будь-який вміст, не зашифрований FBE, наприклад, макети каталогів, розміри файлів, дозволи та час створення/модифікації. Цей ключ захищений Keymaster, який, у свою чергу, захищений перевіреним завантаженням.

Шифрування на весь диск. Android 5.0 до Android 9 підтримують повне шифрування диска. Воно використовує один ключ, захищений паролем пристрою користувача, щоб захистити весь розділ даних користувача пристрою. Після завантаження користувач повинен надати свої облікові дані, перш ніж будь-яка частина диска стане доступною (це може спричиняти недоступність певного функціоналу, такого як прийом дзвінків, будильники, а також служби доступу) [18].

2.5. Модель безпеки платформи Android на рівні застосунків

Застосунки ОС Android використовують сучасне апаратне та програмне забезпечення, а також локальні дані, взаємодія з яким відбувається за допомогою платформи. Захист відкритої платформи вимагає надійної архітектури безпеки та суворих програм безпеки. На рівні застосунків можна виділити два основних механізми – модель дозволів Android та схема підписання застосунку.

2.5.1. Модель дозволів Android

Усі програми в ОС Android працюють у своєму приватному просторі. За замовчуванням програма Android може отримати доступ лише до обмеженого діапазону системних ресурсів. Система керує доступом додатків Android до ресурсів, які в разі неправильного або зловмисного використання можуть негативно вплинути на роботу користувача, мережу або дані на пристрої.

Ці обмеження реалізуються в різних формах. Деякі можливості обмежені навмисною відсутністю API для чутливої функціональності (наприклад, немає API Android для безпосереднього маніпулювання SIM-карткою). У деяких випадках поділ ролей є заходом безпеки, так само як ізоляція сховища для кожної програми. В інших випадках конфіденційні API призначені для використання надійними програмами та захищені за допомогою механізму безпеки, відомого як дозволи.

Список таких API включає в себе:

- функції камери;
- дані про місцезнаходження (GPS);
- функції Bluetooth;
- функції телефонії;
- функції SMS/MMS;
- мережа/підключення даних.

Ці ресурси доступні лише через операційну систему. Щоб використовувати захищені API на пристрої, програма повинна визначити необхідні їй можливості у своєму маніфесті. Усі версії Android 6.0 і вище використовують модель дозволів під час виконання. Якщо користувач запитує функцію від програми, для якої потрібен захищений API, система відобразить діалогове вікно із пропозицією відмовити або надати дозвіл.

ОС Android класифікує дозволи на різні типи, включаючи дозволи на час встановлення, дозволи під час виконання та спеціальні дозволи. Кожен тип дозволу вказує на обсяг обмежених даних, до яких програма може отримати доступ, і обсяг обмежених дій, які може виконувати програма, коли система надає їй цей дозвіл.

Дозволи під час встановлення. Дозволи на час встановлення надають програмі обмежений доступ до специфічних даних і дозволяють додатку виконувати обмежені дії, які мінімально впливають на систему чи інші програми. Якщо для застосунку оголошено ряд дозволів, то під час встановлення система автоматично надає їх додатку. ОС Android включає кілька підтипів дозволів на час встановлення, включаючи звичайні дозволи та дозволи підпису. Звичайні дозволи надають доступ до даних і дій, які виходять за межі приватного простору додатка. Якщо ж програма оголошує дозвіл підпису, який визначив інший додаток, і якщо дві програми підписані одним сертифікатом, система надає дозвіл першій програмі під час встановлення. Інакше цій програмі не можна надати дозвіл.

Дозволи під час виконання. Дозволи під час виконання, також відомі як небезпечні дозволи, надають застосунку додатковий доступ до обмежених даних і дозволяють йому виконувати обмежені дії, які більш суттєво впливають на систему та інші програми. Тому спочатку необхідно виконувати запит на дозволи під час роботи додатку, перш ніж він зможете отримати доступ до обмежених даних або виконувати обмежені дії. Багато дозволів під час виконання мають доступ до приватних даних користувача, даних особливого типу, які містять потенційно конфіденційну інформацію. Приклади приватних даних користувача

включають місцезнаходження та контактну інформацію. Крім цього, мікрофон і камера забезпечують доступ до особливо конфіденційної інформації. Таким чином, система допомагає пояснити, чому додаток отримує доступ до цієї інформації. ОС Android призначає «небезпечний» рівень захисту дозволам під час виконання.

Звичайні дозволи. Це дозволи, котрі надають доступ до даних і дій, які виходять за межі приватного середовища застосунку. Однак дані та дії не представляють великого ризику для конфіденційності користувача та роботи інших програм. ОС Android призначає "звичайний" рівень захисту звичайним дозволам.

Спеціальні дозволи. Спеціальні дозволи відповідають конкретним операціям програми. Тільки платформа та OEM-виробники можуть визначати спеціальні дозволи. Крім того, платформа та OEM-виробники зазвичай визначають спеціальні дозволи, коли вони хочуть захистити доступ до особливо небезпечних дій, таких як малювання над іншими програмами. Сторінка доступу до спеціальної програми в системних налаштуваннях містить набір операцій, які можна перемикати користувачу. Багато з цих операцій реалізуються як спеціальні дозволи [19].

Після надання дозволи застосовуються до програми, доки вона встановлена. Щоб уникнути плутанини користувачів, система не сповіщає користувача знову про дозволи, надані програмі, а програми, які включені в основну операційну систему або входять у комплект OEM, не запитують дозволів у користувача. Дозволи видаляються, якщо застосунок видалено, тому наступна повторна інсталяція знову призведе до запиту дозволів.

У випадку, якщо застосунок намагається використати захищену функцію, яка не була оголошена в маніфесті, збій дозволу зазвичай призведе до повернення до програми винятку безпеки. Перевірка дозволів захищених API виконується на найнижчому можливому рівні. Приклад обміну повідомленнями між застосунком та даними пристрою, при спробі запиту доступу до захищених API, показано на рисунку 8 [20].

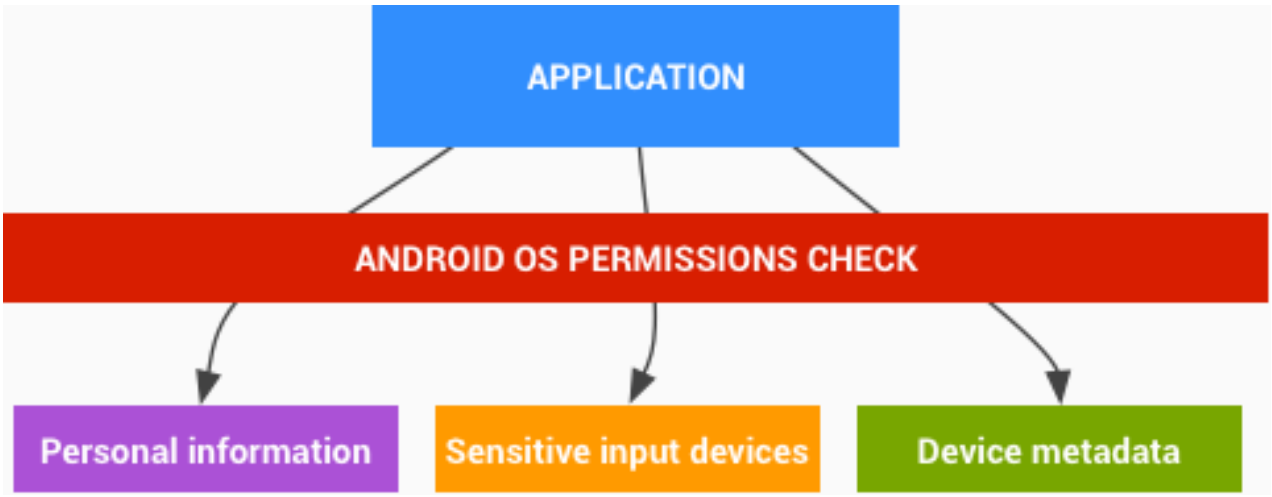


Рисунок 8 – Взаємодія застосунку та даних користувача

2.5.2. Підпис застосунку

Підписування коду дозволяє розробникам ідентифікувати автора програми та оновлювати свою програму без створення складних інтерфейсів та дозволів. Кожен додаток, який запускається на платформі Android, повинен бути підписаний розробником. Застосунки, які намагаються встановити без підпису, відхиляються Google Play або інсталятором пакетів на пристрої Android.

У Google Play підписання додатків поєднує довіру, яку компанія Google має до розробника, і довіру розробника до своєї програми. Розробники знають, що їх додаток надається без змін для пристрою з ОС Android і розробники можуть нести відповідальність за поведінку своєї програми.

В ОС Android підписання застосунку — це перший крок до розміщення програми. Наявний підпис додатку визначає, який ідентифікатор користувача пов'язаний з якою програмою; різні програми працюють під різними ідентифікаторами користувачів. Підписання програми гарантує, що одна програма не зможе отримати доступ до жодної іншої програми, крім як через чітко визначений IPC.

Коли застосунок (файл APK) інстальовано на пристрої з ОС Android, менеджер пакетів перевіряє, що файл APK належним чином підписаний сертифікатом, включеним у цей файл APK. Якщо сертифікат (або, точніше,

відкритий ключ у сертифікаті) збігається з ключем, який використовується для підписання будь-якого іншого файлу APK на пристрої, новий APK має можливість вказати в маніфесті, що він буде ділитися UID з іншим аналогічним чином підписаним додатком.

Програми також можуть оголошувати дозволи безпеки на рівні захисту підпису, обмежуючи доступ лише до програм, підписаних тим самим ключем, зберігаючи при цьому окремі ідентифікатори UID та приватний простір застосунку. Більш тісні відносини зі спільним підписом дозволяються за допомогою функції спільного UID, де два або більше додатків, підписаних одним ключем розробника, можуть оголосити спільний UID у своєму маніфесті [21].

Висновки до другого розділу

В даному розділі було розглянуто архітектуру операційної системи та модель безпеки платформи Android з метою більшого розуміння існуючих механізмів та способу їх використання у виявленні зловмисного програмно забезпечення.

Насамперед, було розглянуто архітектуру операційної системи Android, визначено її складові та описано їх більш детально. Найбільш інтерес для даної роботи представляє рівень застосунків та фреймворку Java API, так як саме вони зазвичай є індикаторами зловмисного програмного забезпечення. Для проведення аналізу застосунку необхідним є розуміння структури застосунку в ОС Android, тому дана інформація була представлена у відповідних розділах.

Крім того, у деталях було описано існуючу модель безпеки платформи Android включно з її складовими. На рівні операційної системи Android покладається на покращені засоби захисту ядра Linux, що надають можливість безпечно виконувати застосунки системи в різних потоках без ризику несанкціонованого доступу зі сторони будь-якого з них. Система також використовує можливості шифрування даних, безпечного запуску системи та

ізольованого середовища виконання додатків. Можна зробити висновок, що на рівні операційної системи платформа Android має надійний захист.

Що стосується рівня застосунків моделі безпеки Android, то на даному етапі з’являється більш широкий простір для проникнення зловмисного програмного забезпечення, так як велика відповідальність покладається на користувача. Перевірка підпису застосунку лише частково вберігає від поширення небезпеки для системи, так більшість додатків випускаються не приватними компаніями, а індивідуальними лицями. Тому основна відповідальність лягає на модель дозволів Android, так як саме при наявності некоректних, підозрілих, або надлишкових дозволів у комбінації з використанням метод Java API зловмисне програмне забезпечення може отримати інструменти для здобуття власних цілей. Як наслідок, проблеми даного рівня безпеки Android взяті для усунення в даній роботі.

3. СТАТИЧНИЙ АНАЛІЗ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Загальні відомості

Статичний аналіз — це важлива техніка, котра використовується в сучасній розробці програмного забезпечення та передбачає аналіз коду без його виконання. Метою статичного аналізу є виявлення та усунення помилок й потенційних проблем із продуктивністю у коді програми. Крім того, інструменти статичного аналізу часто використовуються для виявлення потенційних вразливостей у коді. Інструменти статичного аналізу можуть визначати вразливості в коді, якими можуть скористатися зловмисники, дозволяючи користувачам та розробниками вживати відповідні запобіжні заходи.

Статичний аналіз програми, як правило, включає автоматизований інструмент, який бере в якості вхідних даних вихідний код (або в деяких випадках об’єктний код) програми, досліджує цей код, не виконуючи його, та видає результат, перевіряючи структуру коду, послідовності операторів і те, як значення змінних обробляються під час викликів різних функцій. Основна перевага статичного аналізу полягає в тому, що аналізується весь код програми. Це відрізняється від динамічного аналізу, в якому частини коду можуть виконуватися лише за певних умов, які ніколи не можуть бути виконані у цьому аналізі.

Типовий процес статичного аналізу починається з представлення аналізованого коду програми в деяких абстрактних моделях (наприклад, графік викликів, графік потоку керування або діаграма класів/послідовностей UML) на основі цілей аналізу. Ці абстрактні моделі фактично забезпечують спрощений інтерфейс для підтримки клієнтського аналізу верхнього рівня. Більш глибокий аналіз, такий як аналіз значень змінних у різних частинах графу потоку управління, також може бути використаний за допомогою аналізу потоку даних з метою надання повної картини структури та якості коду програми.

Серед переваг статичного аналізу можна виділити наступні:

- можливість виявляти потенційні проблеми в коді до запуску програмного забезпечення, що дозволяє не ставити під загрозу цільовий пристрій;
- комплексний аналіз всього коду програми;
- швидкість виконання (порівняно з динамічним аналізом);
- використання невеликої кількості апаратних ресурсів (у порівнянні з динамічним аналізом);

За допомогою статичного аналізу можна виявити багато вразливостей, таких як витік приватних даних, несанкціонований доступ до захищених або приватних ресурсів. При аналізі застосунків в ОС Android, даний підхід використовується для виявлення неправомірного використання дозволів, споживання енергії, виявлення клонів і створення тестів, а також визначення проблем перевірки коду та криптографії.

3.2. Техніки статичного аналізу

Серед технік статичного аналізу можна виділити наступні [22]:

- Аналіз потоку керування. Це техніка, яка використовується в статичному аналізі для визначення можливих шляхів виконання програми. Вона перевіряє вихідний код програми, щоб визначити всі можливі шляхи, якими можна скористатися під час виконання програми. Зазвичай послідовності керування виражаються у вигляді графу потоку керування, в якому кожен вузол представляє основний блок коду (оператор або інструкцію), тоді як кожне спрямоване ребро вказує потенційний потік передачі керування між двома вузлами. Дана техніка є важливою для виявлення потенційних проблем у коді програми, таких як мертвий код, недоступний код та нескінченні цикли.
- Аналіз потоку даних. Це техніка, що використовується для обчислення набору можливих значень в кожній точці програми. Аналіз потоку даних передбачає перевірку маніпуляцій над даними в програмі, таких як присвоєння змінних, виклики функцій та блоки інструкцій, з метою визначення,

як дані використовуються та поширюються в ході виконання програми. Цей аналіз є важливим для виявлення потенційних проблем, таких як не присвоєні змінні, невикористані змінні та неправильне використання змінних.

- Точковий аналіз. Це техніка, яка використовується в статичному аналізі для визначення можливих значень, на які може вказувати вказівник у програмі під час виконання. Дана техніка перевіряє вихідний код програми, щоб визначити, як саме під час виконання програми використовуються та змінюються вказівники. Точковий аналіз є важливим для виявлення потенційних проблем, таких як звертання до нульового вказівника, витік пам’яті та переповнення буфера.
- Алгоритми на основі графів викликів:
    - аналіз ієрархії класів;
    - динамічний аналіз типів;
    - аналіз типів змінних;
    - алгоритм Андерсена [23];
    - алгоритм Стінсгарда [24];

3.3. Побудова графу викликів

Оскільки ОС Android підтримує об’єктно-орієнтовану схему програмування з мовою Java варто зосередитися на аналізі об’єктно-орієнтованих програм. Такі програми складаються з класів, кожен з яких представляє певну концепцію (наприклад, автомобіль), включає набір полів для представлення характеристики даної концепції (наприклад, кількість місць в автомобілі) та набір методів, що містять код для маніпулювання з об'єктами представленого класу (наприклад, початок чи зупинка руху автомобіля). Код методу може викликати інші методи для створення екземплярів вказаного класу або маніпулювати вже існуючими об'єктами.

Виконання програми починається з однієї точки входу, яку в мові програмування Java називають методом «main». Побудова графу викликів

починається з аналіз коду саме цього методу. При проходженні по ньому кожному виклику іншого методу присвоюється свій вузол, котрий прямує від вузла «main». Після цього запускається нова ітерація аналогічного проходу, однак вже по коду попередньо знайдених викликаних методів. Повторення цього алгоритму призводить до побудови орієнтованого графа (рисунок 9), широко відомого в аналізі програм як граф виклику:

```
class Main {
  public static void main(String[] args) {
    A();
    B();
  }
  public static int A(){
    C();
  }
  public static void B(){
      B();
  }
}
```

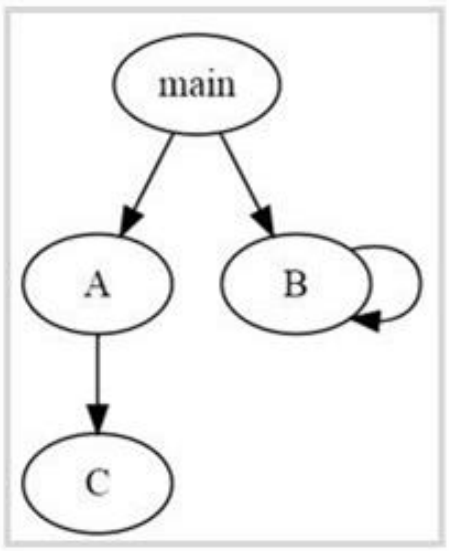


Рисунок 9 – Приклад графу викликів

Хоча концепція графів викликів є стандартною в підходах до статичного аналізу, різні проблеми, такі як вимоги до точності, можуть вплинути на алгоритми, що використовуються для побудови графу викликів програми. Для програм написаних мовою програмування Java було запропоновано ряд популярних алгоритмів, включаючи алгоритм аналізу ієрархії класів, алгоритм аналіз типів змінних алгоритм Андерсена тощо, кожен з яких в більшій чи меншій мірі чутливий до різних властивостей виконання програм. Нижче наведено детальний опис цих властивостей, щоб показати чітку відмінність між підходами. Властивості виконання програм проілюстровані на рисунку 10 за

допомогою фрагментів коду та двох графів викликів, один з яких чутливий до описаної особливості, в той час як інший – ні [25]:

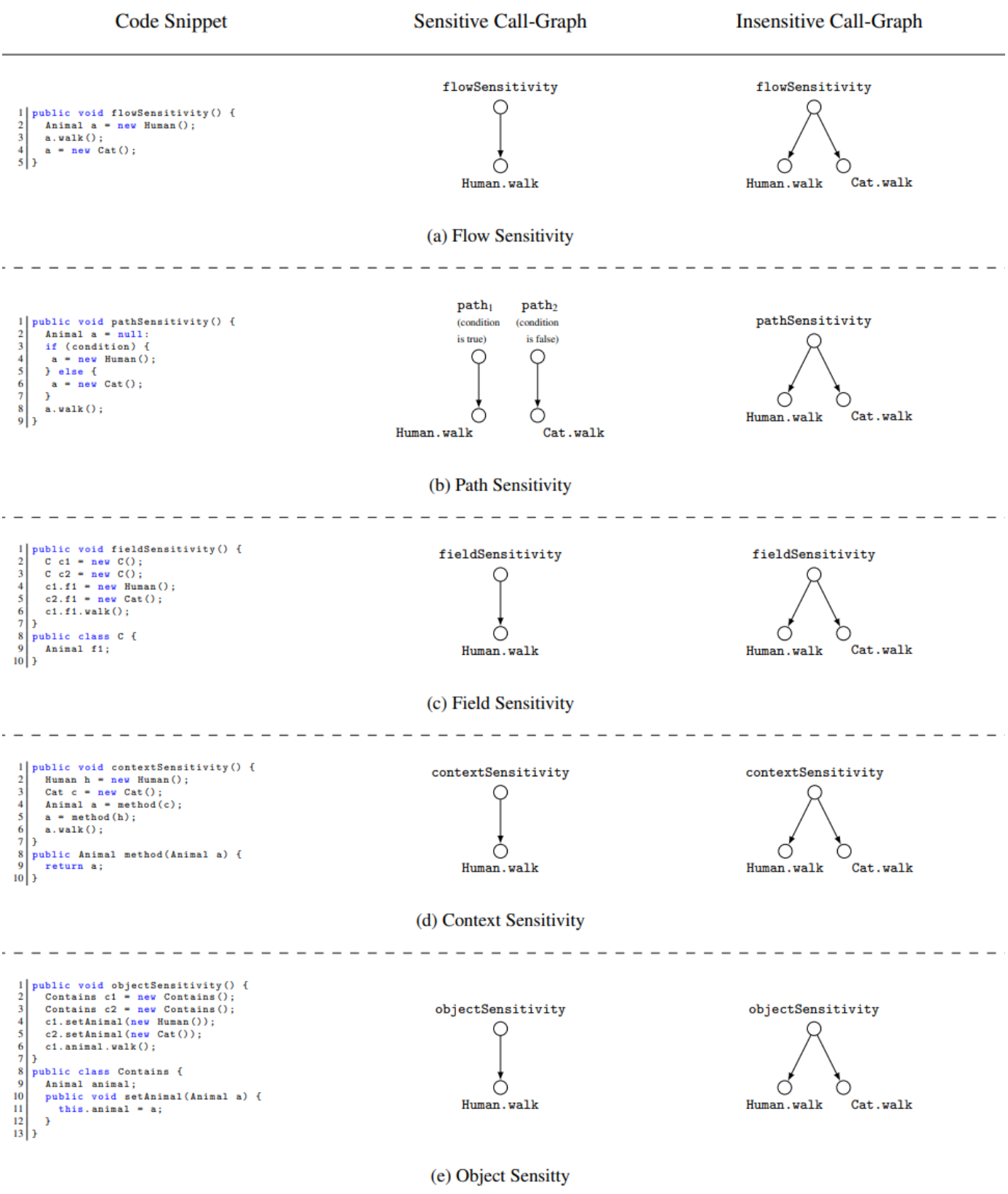


Рисунок 10 – Прикладів випадків чутливості в статичному аналізі об'єктно-орієнтованих програм

Чутливість до порядку виконання. Чутливий до порядку виконання граф викликів — це граф викликів, який знає порядок операторів програми. На прикладі рисунку 10 (a) спочатку створюється екземпляр класу «Human» та присвоюється змінній «a» класу «Animal». За допомогою даної змінний викликається метод «walk». Незважаючи на те, що змінна має клас «Animal», під час виконання даної строки коду буде викликано лише метод «Human.walk». Згодом, програма пов’язує змінну «a» з новим екземпляром, на цей раз класу «Cat». При побудові графу викликів насамперед цікавою є побудова орієнтованих графіків між викликами методів, тобто виклик методів «flowSensitivity» (рядок 1) та «walk» (рядок 3). У випадку, коли граф викликів чутливий до порядку виконання, він містить одне ребро, оскільки в рядку 3 змінна «a» може посилатися лише на об’єкт класу «Human». Якщо ж граф викликів нечутливий до порядку виконання, то при аналізі рядки 3 та 4 можуть бути змінені місцями. Таким чином, граф викликів повинен розглянути випадок, коли змінна «a» посилається на об’єкт класу «Cat» під час виклику «a.walk». Саме тому нечутливий до порядку виконання граф викликів містить два ребра: одне до вузла «Human.walk» та інше до «Cat.walk». Як наслідок, хоч в деяких випадках нечутливий до порядку виконання граф викликів і може бути достатнім, в інших випадках він спричиняє неточність, котра призводить до хибно-позитивних результатів в аналізі.

Чутливість до шляху. Граф викликів, котрий являється чутливим до шляху, враховує шлях виконання програми На прикладі рисунку 10 (b), залежно від значення умови в рядку 3, коли виконання досягає рядка 8, змінна «a» може посилатися на об'єкт класу «Human» або класу «Cat». Таким чином, якщо до уваги береться чутливість до шляху, то необхідно створити два графіки, по одному для кожного можливого шляху: перший шлях, у рядку 8, вказує на виклик методу «Human.walk», а другий – на виклик методу «Cat.walk». Разом з тим, під час створення нечутливого до шляху графу викликів, змінна «a» вказує як на об’єкт класу «Human», так і на об’єкт класу «Cat». Таким чином, графік буде містити виклики обох методів. Це призводить до того, що чутливість до

шляху створює проблему масштабованості для великих програм, в яких може бути експоненціальна кількість шляхів виконання.

Чутливість до полів. Чутливий до полів підхід моделює всі поля кожного об’єкту, що аналізується. На прикладі рисунку 10 (c), у рядках 2 і 3 полям «c1» і «c2» присвоюються об’єкти класу «C», котрі містять лише одну поле (типу «Animal»). У рядку 4 полю «c1.f1» присвоюється об’єкт класу «Human», а в рядку 5 полю «c2.f1» присвоюється об’єкт класу Cat. У результаті чутливого до полів аналізу, у рядку 6 модель поля «c1.f1» може вказувати лише на об’єкт класу «Human», і лише метод «Human.walk» буде знаходиться в такому графі викликів. Разом з тим, нечутливий до полів підхід моделює всі поля лише класів, що розглядаються. Це означає, що в наведеному прикладі, поля «c1.f1» та «c2.f1» мають однакову модель. Таким чином, у рядку 5 поле «f1» вказує на об’єкти класів «Human» та «Cat». Саме тому обидва методи – «Human.walk» та «Cat.walk» – знаходяться в результуючому графі викликів, котрий являється нечутливим до полів.

Чутливість до контексту. При виконанні контекстно-залежного аналізу, під час розбору викликів функцій завжди відстежується контекст. Ця інформація дозволяє з високою точністю переміщуватися назад і вперед від справжнього місця виклику методу, замість того, щоб поступово перевіряти всі можливі місця виклику цього методу в програмі. На прикладі рисунку 10 (d), у рядку 6 метод «walk» викликається об’єктом «a». Виходячи з підходу контекстно-залежного аналізу, кожен виклик методу моделюється незалежно. Тобто, для першого виклику методу (рядок 4) модель вказує на змінну «c», а для другого виклику методу (рядок 5) модель вказує на змінну «h». Однак, так як змінна «h» присвоєються для змінної «a» останньою, то до графіку викликів додається тільки метод «Human.walk». Разом з тим, контекстно-незалежний підхід до аналізу має лише одну модель параметра та одну модель значення, що повертається для даного методу. Тому, у контекстно-незалежному аналізі модель параметра вказує на змінні «c» та «h», так само як значення, що повертаються.

Таким чином, контекстно-незалежний підхід містить обидва методи «Human.walk» та «Cat.walk» у графі викликів.

Чутливість до об’єкту. Об’єктно-чутливий підхід – це контекстно-чутливий підхід, який розрізняє виклики методів, зроблені для різних об’єктів. На прикладі рисунку 10 (е), у 2-3 рядках створюються два об’єкти класу «Contains». Змінні «c1» та «c2» є об'єктами даного класу. Клас «Contains» має поле екземпляру типу «Animal» та метод екземпляра «setAnimal» для встановлення значення полю «animal». У рядку 4 метод «setAnimal» викликається з об’єкту «c1» з об’єктом класу «Human» як параметром. У рядку 5 метод «setAnimal» викликається з об’єкту «c2» з об’єктом класу «Cat» як параметром. Нарешті, у рядку 6, метод «walk» викликається через поле «animal» об’єкта «c1». У рядках 4-5, нечутливий до об'єктів підхід розглядатиме об’єкти «c1» та «c2» як один і той же отримувач. В результаті, виклики методу у рядках 4 та 6 не можуть відрізнити отримувача та модель об’єктів «c1» та «c2» як унікальний об’єкт типу «Contains». Таким чином, метод «walk», викликаний у рядку 6, представлений двома методами на графі викликів: «Human.walk» та «Cat.walk». Разом з тим, чутливий до об’єктів підхід мав би моделі «c1» та «c2» окремо для кожного виклику «setAnimal». Таким чином, виклик у рядку 6 буде представлений лише методом «Human.walk» на результуючому графі викликів.

3.4. Побудова графу потоку керування

Граф потоку керування відрізняється від графу викликів тим, що враховує не тільки виклики методів, а й інструкції кожного виклику методу. Такі графи часто використовується в аналізі ПЗ для представлення програми у формі, в якій оператори та їх послідовність зазвичай представлені як вузли та спрямовані ребра відповідно. Якщо як приклад взяти програму, показану на рисунку 11, на основі попереднього визначення можна було б побудувати три графи потоку керування, по одному для кожного методу, визначеного користувачем (рисунок 12). Наведені графи створені на основі однієї бази методів, що, таким чином,

недостатньо для представлення застосунків в ОС Android, які передбачають часті передачі викликів від одного методу до іншого.

```
1  void main() {
2    int x = secret();
3    int y = 0;
4    y = foo(x);
5    print(y);
6  }
7  int secret() {
8    int q = deviceid();
9    return q;
10 }
11 int foo(int p) { return p; }
```

Рисунок 11 – Програма для прикладу побудови графів потоку керування

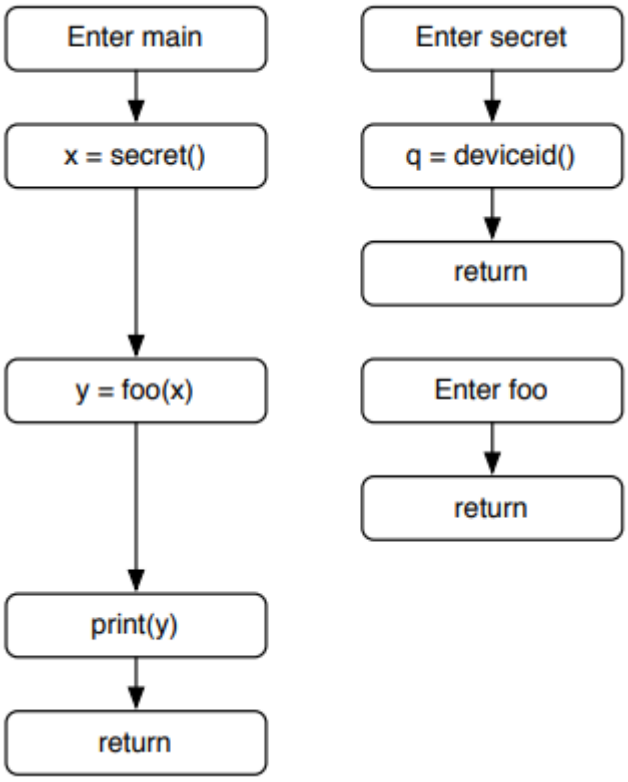


Рисунок 12 – Можливі варіанти графів потоку керування

Для запису механізму викликів методів було введено поняття міжпроцедурного графу потоку керування (МГПК) (рисунок 13):

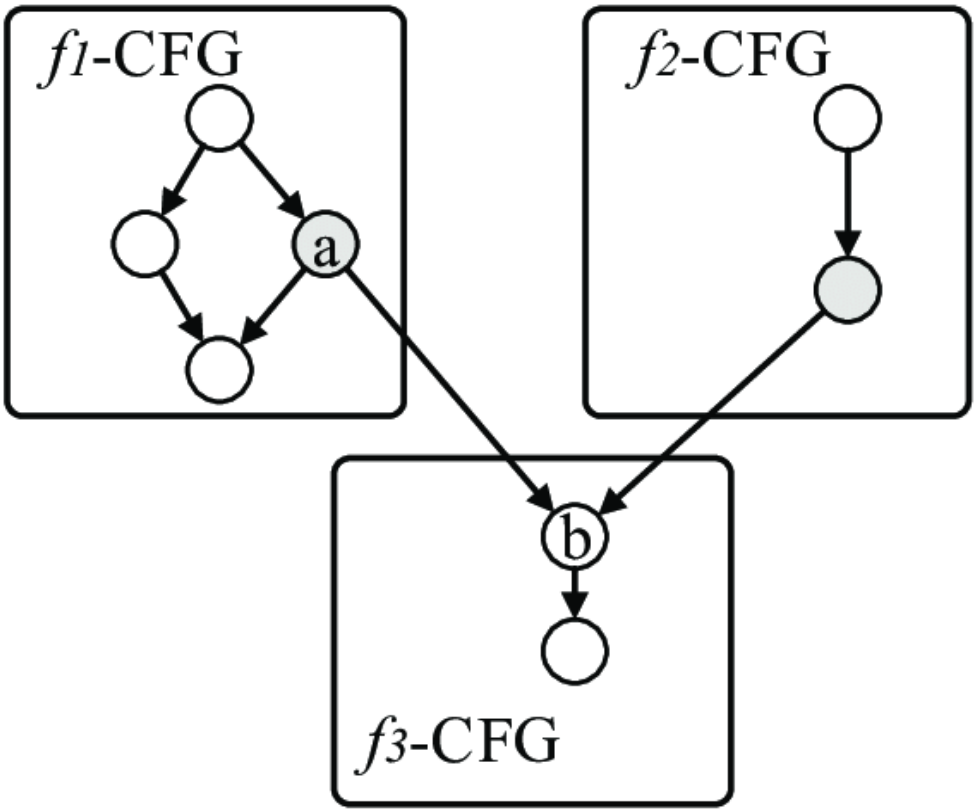


Рисунок 13 – Приклад зв’язку трьох різних графі у МГПК

На додаток до вихідних (звичайних) ребер, МГПК вводить три нові типи ребер для з’єднання внутрішньопроцедурних графів потоку керування:

- ребро виклику: позначає підключення вузла методу виклику до вузла викликаного методу та передачу між ними інформації, наприклад аргументів методу;
- ребро повернення інформації: позначає з’єднання вузла викликаного методу з вузлом методу виклику-повернення та передачу між ними зворотнього значення;
- ребро виклику для повернення: позначає підключення вузла методу виклику до вузла виклику-повернення та передача інформації від методу виклику до всіх його наступних вузлів. Це ребро зазвичай передає інформацію, яка не стосується викликаного вузла. Метою введення цього типу вузла разом із вузлом виклику до повернення є уникнення циклів. Уявляючи, що немає вузла

виклику-повернення, вузол повернення буде безпосередньо вказувати на вузол методу виклику, який може знову перейти в вузол методу виклику через вузол виклику, а потім назад до вузла методу виклику через зворотний вузол, в результаті у петлі.

Щоб краще представити ці введені ребра, МГПК додатково було введено новий вузол, а саме вузол виклику для повернення, для кожного вузла виклику методу.

3.5. Статичний аналіз застосунків в ОС Android

У контексті застосунків в ОС Android статичний аналіз використовується для виявлення потенційних вразливостей безпеки, проблем із продуктивністю та інших проблем із якістю коду. При використанні статичного аналізу для розборі застосунків великий акцент робиться на структурі та коді програми. На основі цього можна виділити наступні підходи:

- Аналіз підпису програми. Як було зазначено раніше, будь-яка програма Android має унікальний підпис. Рішення на основі сигнатур перевіряють вміст або шаблони програми за словником сигнатури шкідливих програм. Якщо під час аналізу у словнику було знайдено відповідність, то даний застосунок повинен розглядатися з високим ступенем недовіри. Даний метод дещо обмежений тим фактом, що він може ідентифікувати лише обмежену кількість нових загроз, наприклад загальні або надзвичайно поширені підписи.
- Аналіз дозволів. Цей механізм працює на основі наданих застосункам дозволів. Аналіз оцінює ризики від наданих дозволі та залежно від рівня ризику, виявляється ступінь шкідливості та небезпеки програми.
- Аналіз потоку керування У цьому підході статичного аналізу перш за все відбувається побудова графу потоку керування застосунку. На основі отриманого графу проводиться пошук наявних потенційних зловживань ресурсами та вразливостями в коді програми. На підставі виявлених загроз і вразливостей приймається рішення про шкідливість або вразливість застосунку.

В загальному, можна виділити наступні етапи статичного аналізу програм в ОС Android:

1. Отримання коду програми. Першим кроком у процесі статичного аналізу є отримання коду програми. Вихідний код програми для ОС Android знаходиться у файл APK у двійковому представленні, тому попередньо його необхідно декомпілювати у формат, котрий дозволяє проводити подальший аналіз коду. Цей процес передбачає вилучення вихідного коду, ресурсів та інших ресурсів із файлу APK. Існує кілька інструментів декомпіляції, які можна використовувати для вилучення коду, серед яких можна виділити «apktool» та «dex2jar».
2. Аналіз дозволів. Аналіз дозволів в ОС Android стосується процесу вивчення дозволів, запитуваних програмою, та оцінки їх потенційного впливу на конфіденційність і безпеку користувача. Це передбачає розуміння різних типів дозволів, доступних у платформі Android, як вони можуть використовуватися програмами та як користувачі можуть ними керувати. Виконуючи аналіз дозволів, приймаються рішення про те, які програми відповідають вимогам безпеки, а які можуть нести загрозу системі та пристрою.
3. Сканування коду. Для цього етапу використовуються розроблені або придбані інструменти автоматизованого сканування. За допомогою них можна виявити потенційні вразливості та інших проблем в рамках коду. Ці інструменти використовують різні методи, такі як зіставлення шаблонів, аналіз потоку даних та сканери вразливостей для виявлення потенційних проблем у коді. Інструменти автоматичного сканування можуть бути особливо корисними для виявлення проблем, які важко виявити вручну. Наприклад, сканери вразливостей можна використовувати для сканування коду на відомі вразливості та слабкі місця в системі безпеки програми. Інструменти автоматичного аналізу коду також можна використовувати для перевірки типових помилок

кодування, таких як винятки нульового покажчика та інші типові помилки.

4. Звітування. Після завершення процесу статичного аналізу створюється звіт, який узагальнює результати. Цей звіт містить відомості про будь-які потенційні вразливості, проблеми з продуктивністю або інші проблеми з якістю коду, які були виявлені під час аналізу. Звіт зазвичай містить опис кожної проблеми разом із рекомендаціями щодо вирішення проблеми. Звіт також може містити рейтинг серйозності для кожної проблеми, вказуючи рівень ризику, пов’язаного з уразливістю чи іншою проблемою.

Описані етапи є загальною структурою ПЗ для виявлення зловмисних застосунків в ОС Android, з метою забезпечення кращого досвід для користувачів та зниження ризику порушення його конфіденційності та безпеки його пристрою.

3.6. Проблеми статичного аналізу відносно особливостей ОС Android.

Байт-код Dalvik. Застосунки для ОС Android в більшості своїй розробляються мовою програмування Java та компілюються в байт-код. Байт-код надалі працює у віртуальній машині Dalvik, котра містить модель інструкцій на основі регістра. Незважаючи на те, що існують сховища з відкритим вихідним кодом, де знаходиться вихідний код багатьох програм, розробники використовують офіційні/комерційні ринки для розповсюдження своїх APK файлів. Таким чином, на практиці статичний аналізатор для застосунків ОС Android повинен бути здатний безпосередньо працювати з байт-кодом Dalvik або, принаймні, перекладати його в підтримуваний формат. Виходячи з цього, більшість класичних аналізаторів вихідного коду та байт-коду мови програмування Java потребують адаптації під екосистему Android.

Точка входу в програму. На відміну від програм більшості загальних мов програмування, таких як Java і C, програми для Android не мають основного

методу. Натомість кожна програма містить кілька точок входу, які викликаються фреймворком Android під час виконання. Отже, статичному аналізатору недостатньо побудувати глобальний графік викликів програми. Замість цього аналізатор повинен спочатку шукати всі точки входу та побудувати кілька графів викликів, не маючи впевненості в тому, як ці графи можуть з’єднуватися один з одним.

Життєвий цикл компонента. В ОС Android різні компоненти програми мають свій життєвий цикл. Кожен компонент реалізує свої методи життєвого циклу, які викликаються системою Android для запуску/зупинки/відновлення роботи компонента відповідно до потреб середовища. Наприклад, додаток у фоновому режимі (тобто з невидимим періодом життя) можна спочатку зупинити, коли система має нестачу пам’яті, а пізніше перезапустити, коли користувач спробує поставити застосунок на передній план. На жаль, оскільки ці методи життєвого циклу безпосередньо не пов’язані з потоком виконання, вони перешкоджають обґрунтованості деяких сценаріїв аналізу.

Користувач/системні події. Окрім методів життєвого циклу, методи обробки подій користувача (наприклад, дій UI) та системних подій (наприклад, ситуація з малим обсягом пам’яті або зміни місцезнаходження GPS) кидають виклик для статичного аналізу програм Android. Оскільки такі події можуть бути запущені в будь-який час, статичні аналізатори не можуть побудувати та підтримувати надійну модель подій. Крім того, розглянути всі можливі варіанти даної моделі занадто дорого в плані обмеженості ресурсів пристрою та користувача.

Міжкомпонентний зв'язок. ОС Android запровадила специфічний механізм, який дозволяє компонентам програми обмінюватися повідомленнями з компонентами тієї ж програми або інших програм. Це спілкування зазвичай ініціюється специфічними методами, які використовують спеціальний параметр, що містить всю необхідну інформацію, для визначення цільових компонентів і запитуваної дії. Подібно до методів життєвого циклу, ці методи фактично обробляються системою, яка відповідає за їх контроль під час виконання. Тому,

статичний аналізатор вважатиме небезпечним висування гіпотези про те, як компоненти з’єднуються один з одним, якщо не використовувати розширені евристики.

Бібліотеки. АРК — це окремий пакет, що містить байт-код Dalvik, що складається з фактичного коду програми та всіх бібліотечних наборів, таких як бібліотеки реклами та інші фреймворки. Ці бібліотеки можуть представляти тисячі рядків коду, що призводить до того, що розмір фактичного додатку буде значно меншим, ніж його ж з підключеними бібліотеками. Ця ситуація викликає дві труднощі: аналіз програми може витрачати більше часу на перевірку бібліотечного коду, ніж реального коду та результати аналізу можуть містити забагато помилкових результатів через аналіз «мертвого коду» бібліотеки. Наприклад, аналіз усіх викликів методів у файлі АРК для виявлення набору необхідних дозволів може призвести до списку дозволів, які насправді не потрібні для фактичного коду програми [25].

3.7. Цілі статичного аналізу в ОС Android

В ОС Android статичний аналіз застосовувався для вирішення різноманітних завдань. Серед іншого, даний аналіз застосовується для виявлення різних проблем безпеки (наприклад, витоку приватних даних або проблем з керуванням дозволами), для перевірки коду, порівняння програм для виявлення клонів, автоматизації створення тестових випадків або для оцінки ефективності коду з точки зору продуктивності та споживання енергії. Загалом можна визначити сім поширених цілей статичного аналізу в ОС Android.

Витоки приватних даних. Останнім часом занепокоєння щодо конфіденційності програм для ОС Android змусило дослідників зосередитися на витоках приватних даних.

Вразливі місця. Вразливості безпеки є ще однією проблемою для користувачів додатків, які повинні бути захищені від шкідливого програмного забезпечення, яке використовує дані та дозволи застосунків.

Зловживання дозволами. Перевірка дозволів є основою архітектури безпеки ОС Android. Модель безпеки Android на основі дозволів пов’язує конфіденційні ресурси з набором дозволів, які необхідно надати перед доступом. Зловмисне програмне забезпечення дійсно може використовувати дозволи (які не потрібні для основної функціональності програми) для досягнення своїх зловмисних цілей.

Споживання енергії. Час роботи акумулятора вже давно є проблемою для мобільних пристроїв. Великі екрани, які є в сучасних смартфонах, є найбільш енергоємними компонентами. Сучасні смартфони використовують OLED, який споживає більше енергії при відображенні світлих кольорів, ніж темних. Виходячи з різних розрахунків, споживання енергії можна було б зменшити на 40%, якщо створити більш ефективні веб-сторінки для мобільних систем (наприклад, із темним кольором фону).

Виявлення клонів. Крім наведеного вище, статичний аналіз можна також використовувати для виявлення клонів застосунків у ОС Android.

Генерація тестових випадків. Поширеність додатків для ОС Android підкреслила необхідність застосування різних методів автоматизованого тестування. Генерація тестових умов має на меті створити набір виконуваних тестових випадків, які можна використовувати для підтримки автоматичних і повторюваних тестів. Поширеним засобом створення тестових випадків є передача параметрів у застосунок з керівництвом деяких попередньо витягнутих моделей, щоб забезпечити доступність певних частин застосунку.

Перевірка коду. Перевірка коду має на меті забезпечити коректність заданого застосунку. Наприклад, пропонується перевірити, чи відповідають додати для ОС Android їхнім вимогам щодо конфіденційності, перш ніж встановлювати програму.

Висновки до третього розділу

В даному розділі було розглянуто статичний аналіз програмного забезпечення на якому базується спосіб виявлення зловмисного програмного забезпечення в даній роботі.

Насамперед було розглянуто загальну інформацію про даний підхід, його переваги, особливості та різновиди розвитку. Було наведено основні техніки проведення статичного аналізу, що базуються на графах програм, котрі підлягають аналізу. Це граф викликів, граф потоку керування, а також граф потоку даних. В результаті розгляду кожного з них було визначено, що найкращим варіантом для даної роботи є використання графу викликів.

Крім цього, було детальніше розглянуто особливості статичного аналізу застосунку в ОС Android, а саме можливі напрямки розвитку в сторону аналізу підписів, дозволі та потоку керування. Наведено також складнощі, котрі можуть виникати при проведенні аналізу в ОС Android, а також було показано сфери використання статичного аналізу.

Для даної роботи було обрано напрями аналізу дозволів та потоку виконання програми з використанням даних, що будуть отримані при виконанні реверсивної інженерії над файлом APK, а також після побудови графів виклику та контролю виконання.

4. ОПИС АЛГОРИТМУ ТА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ РІШЕННЯ

4.1. Загальний опис запропонованого рішення

В даній магістерській дисертації пропонується спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунків в ОС Android. В його основі лежить прагнення до реалізації гнучкого інструменту для аналізу та оцінки безпечності додатків, що дало б змогу легко змінювати конфігурацію аналізу залежно від поточних потреб, а також надавало б розширені відомості про застосунок, котрий аналізується.

На основі запропонованого способу було реалізовано програмне забезпечення, робочий алгоритм якого складається з наступні ключових етапів (рисунок 14):

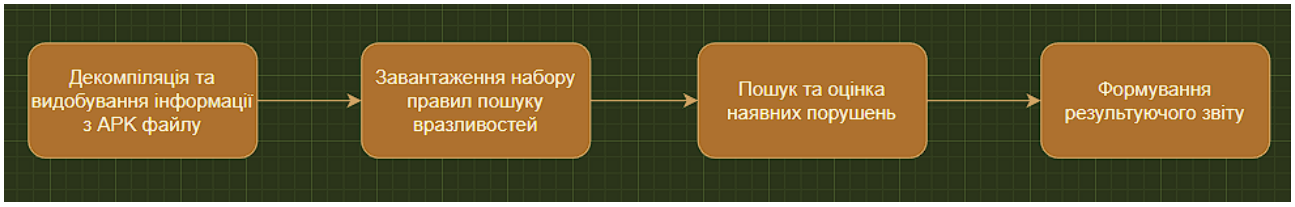


Рисунок 14 – Алгоритм запропонованого рішення

1. Декомпіляція та видобування інформації з APK файлу. На даному етапі на вхід ПЗ подається APK файл, аналіз якого необхідно провести. За допомогою інструментів статичного аналізу застосунків відбувається декомпіляція даного файлу та трансляція з текстового у програмне представлення застосунку (у вигляд визначених програмних об’єктів), доступне для подальшої взаємодії у середовищі виконання ПЗ.
2. Завантаження набору правил. Запропонований спосіб виявлення зловмисного програмного забезпечення базується на знаходженні визначених шаблонів поведінки у коді програм. В даному рішенні пропонується відділити набір шаблонів зловмисної поведінки у окремий модуль та завантажувати його для кожного окремого випадку

аналізу. Це дає змогу змінювати конфігурацію аналізу без зміни коду ПЗ та підлаштовуватись під поточні цілі без зайвих труднощів.

3. Пошук та оцінка наявних порушень. На даному етапі відбувається аналіз отриманої інформації про застосунок на базі переданого набору правил. Аналіз складається з послідовного виконання кількох етапів перевірок, котрі намагаються виявити небезпечні секції коду застосунку відповідно до правил. Кожному знайденому випадку присвоюється певна вага та оцінка ступеня впевненості.
4. Формування результуючого звіту. Після проведення аналізу та оцінки порушень, ПЗ формує докладний звіт, в який додає всі знайдені випадки та наводить додаткову інформацію про них. Даний етап дозволяє провести аналіз фінальної оцінку шкідливості застосунку та впевненості в результаті. Кінцевий користувач може погодитись з нею, або виокремити випадки помилкових спрацювань і прийняти рішення про шкідливість програми на власний розсуд.

Далі пропонується детальний розбір кожного з описаних етапів запропонованого рішення.

4.2. Декомпіляція та видобування інформації з APK файлу

Декомпіляція та видобування інформації з APK файлу є важливим етапом у процесі виявлення шкідливого програмного забезпечення на основі статичного аналізу програм. Його метою є отримання характеристик, функцій та структури коду застосунку. Отримана інформація служить вхідними даними для подальшого аналізу.

У даній роботі для цих задач було використано спеціалізований програмний інструмент для виконання поширених завдань статичного аналізу застосунків Android — Androguard. Androguard — це бібліотека Python з відкритим вихідним кодом, яка надає інструменти для аналізу, декомпіляції та розбору програм для ОС Android. Вона широко використовується для статичного

аналізу та завдань зворотного проектування, оскільки дозволяє отримувати корисну інформацію з файлів, що містяться в APK пакеті застосунку (таких як «AndroidManifest.xml», «classes.dex» та інших ресурсів).

В результаті декомпіляції переданого на вхід ПЗ APK файлу формуються наступні програмні об’єкти:

- екземпляр класу APK, який представляє завантажений APK файл та містить набір полів і методів, котрі дозволяють отримати необхідні групи даних про застосунок. Даний об’єкт використовується для доступу до різних метаданих і ресурсів застосунку, таких як інформація про пакет, «AndroidManifest.xml», дозволи та компоненти.
- екземпляр класу DalvikVMFormat, який представляє проаналізований виконуваний файл Dalvik (DEX). Файли DEX містять скомпільований байт-код Java застосунку. Клас DalvikVMFormat містить методи й атрибути для доступу та керування розібраним байт-кодом, який надалі використовується в аналізі застосунку.
- екземпляр класу Analysis, який представляє дані статичного аналізу виконуваних файлів Dalvik. Даний клас містить методи й атрибути для доступу до різноманітної інформації, такої як графіки викликів, графіки потоків керування та інших даних, пов’язаних з кодом застосунку. Даний клас полегшує розуміння структури застосунку та сприяє виявленню потенційних вразливостей або виявлення шкідливого коду.
- словник декомпільованих класів, який зіставляє назви класів із відповідним декомпільованим вихідним кодом Java. Цей словник використовується для доступу до декомпільованого коду Java для подальшого аналізу, пошуку конкретних шаблонів та розбору логіки застосунку.

Наведені програмні об’єкти надають змогу взаємодіяти зі структурою та кодом потенційно зловмисного застосунку в межах середовища виконання розробленого ПЗ та допомагають у виявленні небезпечних шаблонів поведінки. Інформація, котра отримується з цих об’єктів складається з дозволів, викликів

системних API, намірів, графіків потоку керування, а також даних про пакети та класи застосунку.

4.3. Завантаження набору правил пошуку вразливостей

Ключовим аспектом запропонованого способу виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку є гнучкість, котра проявляється у здатності власноруч налаштовувати правила пошуку шкідливого програмного забезпечення згідно з власними потребами та використовувати різні набори правил залежно від ситуації.

В даній роботі пропонується наступна структура правил визначення небезпечних секції коду, котра являється JSON об'єктом, що складається з таких частин (рисунок 15):

- Назва правила. Текстове поле, що слугує місцем для змістовного найменування правила з метою відображення його суті. Дане поле разом з додатковою інформацією буде показано користувачеві у випадку коли вразливість, описану в правилі, буде знайдено.
- Ступінь критичності. Числове поле, що слугує для відображення ступеню критичності вразливості, котра описана в правилі. Дане значення може бути будь-яким та встановлюється користувачем виходячи з використовуваної ним метрики. В даній роботі використовуються значення від 1 до 5, де правило з критичністю 1 вважається легким порушенням, а 5 – критичною загрозою безпеці.
- Дозволи. Список, що включає в себе перелік дозволів застосунку, наявність яких притаманна поведінці зловмисної секції коду, що описується у правилі.
- Методи. Список, що включає в себе перелік методів, наявність та послідовний виклик яких притаманні поведінці програмного забезпечення, котре несе загрозу безпеці користувача. Об'єктом списку являється JSON, котрий містить у собі опис обраного методу відповідно до синтаксису сирцевих

бібліотек. Опис включає в себе сирцеве ім’я класу методу, що викликається, назву самого методу та його сигнатуру.

Дана структура надає можливість створювати опис поведінки різноманітних вразливостей та зловмисних шаблонів коду, що є важливим крок на шляху їх виявлення.

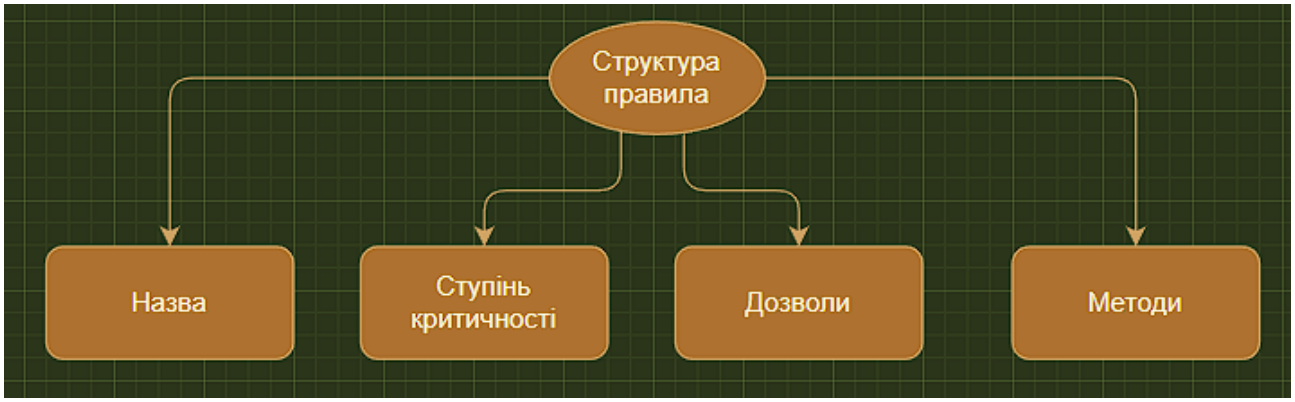


Рисунок 15 – Структура правила пошуку вразливостей

Набір правила пошуку вразливостей зберігається у файлі типу JSON, котрий містить список описаних правил. На практиці, правила пошуку вразливостей мають наступний вигляд (рисунок 16):

```
{
    "rule": "Spy",

    "severity": 4,

    "permissions": [
        "android.permission.ACCESS_COARSE_LOCATION",
        "android.permission.ACCESS_FINE_LOCATION"
    ],

    "api": [
    {
        "class": "Landroid/telephony/TelephonyManager",
        "method": "getCellLocation",
        "descriptor": "()Landroid/telephony/CellLocation;"
    }
    ]
}
```

Рисунок 16 – Приклад правила пошуку вразливостей

На рисунку 16 можна побачити приклад опису правила пошуку потенційного відслідковування місцезнаходження користувача. Поле «rule» містить назву правила, поле «severity» відповідає його ступеню критичності, а поля «permissions» та «api» являються переліками дозволів та викликів методів відповідно, котрі відповідають за доступ до інформації про місцезнаходження пристрою.

Запропонована структура правил надає можливість вибору для користувача – використовувати підготовлений та визначений розробником набір для визначення шкідливого програмного забезпечення, або підготувати свій перелік правил, виходячи з власних потреб.

4.4. Пошук та оцінка наявних порушень

Процес визначення шкідливого програмного забезпечення у застосунку базується на графі викликів застосунку та переданому наборі правил для пошуку зловмисних секцій коду. Для визначення факту присутності вразливостей у застосунку, в даній роботі пропонується обробити кожне правило шляхом послідовного проведення перевірок на:

- Наявність вказаного дозволу.
- Наявність викликів вказаних у правилі методів.
- Відповідність сигнатури викликаних методів.
- Послідовність вказаних викликів методів.

На першому етапі аналізується маніфест додатку на наявність вказаних у правилі дозволів. Якщо результат даної перевірки є успішним, тобто всі вказані у правилі дозволи наявні в застосунку, то процес перевірки переходить на наступний етап. В протилежному випадку перевірка даного правила зупиняється, так як без належного дозволу застосунок не зможе виконати вказані в ньому системні виклики.

На другому етапі відбувається пошук викликів методів, котрі вказані у правилі. Оскільки ім'я методу може використовуватися не одним системним

класом, але й класом, що визначений користувачем, то на даному етапі використовується як назва класу, так і назва методу. При успішному знаходженні викликів, вказаних в правилі, перевірка переходить на наступний етап.

На третьому етапі перевіряється відповідність семантики визначених викликів до наявної в правилах. У разі відсутності повного співпадіння правило не виконується і процес обробки даного правила зупиняється, в іншому випадку відбувається перехід до наступної перевірки.

На четвертому етапі перевіряються послідовності наявних викликів вказаних методів. Якщо послідовність, вказану в правилі пошуку вразливості, буде знайдено в рамках однієї секції коду, то вважається, що в додатку було знайдено вразливість.

Залежно від етапу, на якому зупинилась перевірка, визначається ступінь впевненості у наявності описаної у правилі вразливості. Це означає, що якщо процес обробки поточного правила зупинився на першому етапі, то впевненість у наявності вразливості є мінімальною та оцінюється як 25%. Кожен наступний етап у разі успішності збільшує ступінь впевненості. У разі успішного виконання всіх етапів вважається, що ступінь впевненості в наявності описаної в правилі вразливості становить 100%.

В свою чергу, залежно від номера етапу, на якому зупинилась обробка правила, залежить інтерпретація ваги, котру дане правило має у фінальному звіті. Вага правила визначається за наступною формулою:

$\frac{2^{n-1} * severity}{2^3}$, де n = [1;4] – це останній пройдений етап перевірки правила

У випадку, якщо не було пройдено жодного етапу перевірки, то вага правила вважається рівною нулю.

У результуючому звіті аналізу виводяться впевненість та фінальна вага кожного правила. Якщо хоча б одне правило було знайдено зі 100% впевненістю, то застосунок вважається небезпечним (рівень небезпеки визначається

максимальним значенням небезпеки серед знайдених правил). Якщо жодне з правил не було знайдено повністю (впевненість від 25% до 75%), то сума кінцевих ваг правил у відношенні до суми їх початкових ваг визначає ступінь небезпеки застосунку.

Висновки до четвертого розділу

В даному розділі було описано запропонований спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунків в ОС Android. В основу нього покладено прагнення до створення гнучкого інструменту для дослідження та оцінювання безпеки програм, який дозволив би легко налаштовувати конфігурацію аналізу відповідно до актуальних потреб та надавав би детальні відомості про аналізований застосунок.

Важливим етапом розбору застосунку є процес декомпіляції та видобування інформації з APK файлу. Було проведено знайомство з певними його особливостями, котрі дозволяють перетворити бінарний код додатка у вихідний код, сприяючи аналізу його структури та функціональності. Даний розділ ще раз підкреслив важливість даного етапу, котрий дозволяє виявити потенційні зловживання та безпекові недоліки, а також підтримує постійне поліпшення якості та безпеки програмного забезпечення.

Крім цього, було представлено та описано запропоновану структуру правил визначення небезпечних секції коду. Вона дозволяє описати різні шаблони зловмисного програмного забезпечення та легко піддається розширенню й редагуванню. Це дозволяє конфігурувати аналіз застосунку залежно від обраної цілі та напряму впливати на його перебіг.

На основі сформованого набору правил відбувається безпосередня перевірка застосунку на наявність секцій коду, котрі несуть небезпеку, що дозволяє визначити їх місцезнаходження, а також на основі комплексної оцінки результатів визначити рішення по кожному конкретному випадку та безпечності застосунку в цілому.

5. ТЕСТУВАННЯ РЕАЛІЗОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1. Тестування програмного забезпечення

Для перевірки роботи розробленого програмного забезпечення, котре реалізовує запропонований спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунків в ОС Android було створено тестовий застосунок та наповнено його поширеними варіантами небезпечних секцій коду. Наприклад, на рисунку 17 зображено простий приклад зловмисного програмного забезпечення, котре слідкує за переміщеннями користувача:

```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

public class MainActivity extends AppCompatActivity {

    private LocationManager locationManager;
    private LocationListener locationListener;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        locationListener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                tracker.send(location.getLatitude(), location.getLongitude());
            }
```

Рисунок 17 – Секція коду тестового застосунку, котра містить відслідковуванням місцезнаходження користувача

Даний тестовий застосунок було передано на вхід розробленого ПЗ з ціллю перевірки ефективності даного інструмент. Головна мета полягала в тому, щоб визначити, чи зможе розроблений інструмент точно виявити зловмисну

поведінку у тестовому застосунку та призначити відповідні оцінки серйозності та рівні достовірності кожного зі знайдених випадків. Результат роботи ПЗ зображено на рисунку 18:

```
kravc@DESKTOP-7QCRSFM MINGW64 ~/diploma/apkAnalysis
$ python apkAnalyser.py

Total security risk from possible threats: 38.67 %
Security risk: HIGH (5)

+----------------------+-------------+------------+--------------+----------------+
| Malicious Case       | JSON File   |   Severity | Confidence   |   Actual Score |
+======================+=============+============+==============+================+
| Tracking Geolocation | rule_1.json |          3 | 50%          |           0.75 |
+----------------------+-------------+------------+--------------+----------------+
| Sending SMS          | rule_2.json |          5 | 100%         |           5    |
+----------------------+-------------+------------+--------------+----------------+
| Accessing Contacts   | rule_3.json |          4 | 75%          |           2    |
+----------------------+-------------+------------+--------------+----------------+
| Stealing Credentials | rule_4.json |          2 | 50%          |           0.5  |
+----------------------+-------------+------------+--------------+----------------+
| Accessing Camera     | rule_5.json |          3 | 75%          |           1.5  |
+----------------------+-------------+------------+--------------+----------------+
| Monitoring Clipboard | rule_6.json |          1 | 25%          |           0.12 |
+----------------------+-------------+------------+--------------+----------------+
| Keylogging           | rule_7.json |          4 | 100%         |           4    |
+----------------------+-------------+------------+--------------+----------------+
| Accessing Microphone | rule_8.json |          3 | 50%          |           0.75 |
+----------------------+-------------+------------+--------------+----------------+
| Infecting Other Apps | rule_9.json |          5 | 75%          |           2.5  |
+----------------------+-------------+------------+--------------+----------------+
```

Рисунок 18 – Результат роботи програми

Розроблений інструмент виявлення зловмисних застосунків на основі статичного аналізу виводить результат аналізу у консоль у вигляді таблиці, яка відображає результати аналізу для кожного виявленого випадку зловмисного програмного забезпечення та загального висновку по аналізу.

Узагальнений висновок складається з декількох частин. Загальна оцінка ступеню небезпеки додатку вираховується як максимальне значення небезпеки серед правил, котрі було знайдено з впевненістю 100%. Ступінь небезпеки потенційних загроз (тобто правил, котрі були знайдені з впевненістю від 25 до 75%) – це відношення суми фактичних оцінок для цих правил до суми максимально можливих значень їх небезпеки (тих, що вказані в кожному правилі окремо). Обчислення співвідношення цих двох сум використовується для

визначення рівня потенційної загрози, котру становлять неповністю знайдені небезпечні сценарії.

В той же час, таблиця результатів по кожному з правил складається з таких стовпчиків:

- Назва порушення: у цьому стовпці наведено назву виявленої зловмисної поведінки, такої як відстеження геолокації, надсилання SMS, запис дзвінків тощо. Ці дії представляють потенційні ризики безпеці та порушення конфіденційності.
- Назва файлу з правилом: у цьому стовпці відображається ім’я файлів JSON, котрі містять правила та шаблони, які використовуються розробленим ПЗ для виявлення конкретної зловмисної поведінки в цільовому APK файлі.
- Серйозність: у цьому стовпці вказується оцінка серйозності виявленого випадку зловмисного програмного забезпечення в діапазоні від 1 (найнижча) до 5 (найвища). Оцінка серйозності представляє рівень потенційної шкоди, яку зловмисна поведінка може завдати користувачеві або його пристрою.
- Достовірність: у цьому стовпці відображається рівень достовірності виявлення, виражений у відсотках (25%, 50%, 75% або 100%). Рівень достовірності вказує на впевненість інструменту в точному визначенні конкретної зловмисної поведінки.
- Фактична оцінка: у цьому стовпці відображається розрахована фактична оцінка для кожного виявленого випадку шкідливого програмного забезпечення. Фактичний бал обчислюється за раніше наведеною формулою та допомагає оцінити загальний ризик, пов’язаний з кожною виявленою шкідливою поведінкою, беручи до уваги як її серйозність, так і рівень її достовірності.

Крім наведеного загального результату роботи програми, існує додаткова опція виводу більш детальної інформації по кожному знайденому правилу. На рисунку 19 можна побачити відображення результатів аналізу виявленого випадку відстеження місцезнаходження користувача:

	Information
Permissions	android.permission.ACCESS_FINE_LOCATION
Bytecode Methods Usage	Landroid/location/LocationManager;
	Landroid/location/LocationListener;
Bytecode Methods Sequence	1. Landroid/location/LocationManager;->getSystemService(Ljava/lang/String;)Ljava/lang/Object;
	2. Landroid/location/LocationListener;->onLocationChanged(Landroid/location/Location;)V
	3. Landroid/location/LocationManager;->requestLocationUpdates(Ljava/lang/String;JFLandroid/location/LocationListener;)V

Рисунок 19 – Детальний опис знайденої небезпечної секції коду

Результати представлені у форматі таблиці, що містить три основні розділи:

- Надані дозволи: у цьому розділі перераховано дозволи Android, котрі було використано зловмисним програмним забезпеченням для реалізації неправомірних дій. У цьому випадку шкідливому програмному забезпеченню потрібен дозвіл «android.permission.ACCESS_FINE_LOCATION» для доступу до даних про точне місцезнаходження користувача.
- Використання вказаних методів: у цій секції показано методи, які використовує зловмисне програмне забезпечення, а саме «LocationManager (Landroid/location/LocationManager;)» та «LocationListener» для відстеження місцезнаходження користувача.
- Послідовність методів: у цій секції виводяться послідовності викликів знайдених методів, які зловмисне програмне забезпечення використовує для відстеження місцезнаходження користувача. У таблиці подано послідовність викликів методів, у якій вони виконуються. У цьому випадку методи «getSystemService», «requestLocationUpdates» та «onLocationChanged» викликаються в певному порядку для доступу та обробки даних про місцезнаходження користувача.

Аналізуючи отримані результати, можна отримати більш детальне уявлення про поведінку виявленого шкідливого програмного забезпечення. Ця інформація може бути використана для оцінки потенційних ризиків для безпеки користувача, пов’язаних з проаналізованим застосунком Android та вжиття відповідних заходів для захисту даних та конфіденційності.

5.2. Аналіз окремих тестових випадків

Крім наведеного вище загального тестування роботи розробленого програмного забезпечення, котре реалізовує запропонований спосіб виявлення зловмисного програмного забезпечення, існує потреба розібрати окремі випадки, в яких крім використання описаного правила пошуку небезпечної секції коду необхідно також провести додаткові маніпуляції над нею для отримання більш точного результату.

Першим тестовим випадком є виявлення небезпечного програмного забезпечення, котре використовує вразливість типу «ін’єкція коду». Дана вразливість виникає у ситуаціях, коли програма належним чином не контролює виконуваний код, що призводить до випадків, в яких зловмисник може вплинути на керуючі дані, які використовуються для генерації синтаксису чи структури коду.

В свою чергу, це може призвести до ситуації, в якій зловмисник може створювати вхідні дані, які маніпулюють процесом генерації коду, що призводить до виконання довільного, потенційно шкідливого коду. Код, наданий зловмисником, може бути виконаний за дозволу скомпрометованого процесу, що потенційно може призвести до різноманітних негативних наслідків, таких як несанкціонований доступ, пошкодження даних, відмова в обслуговуванні та інші.

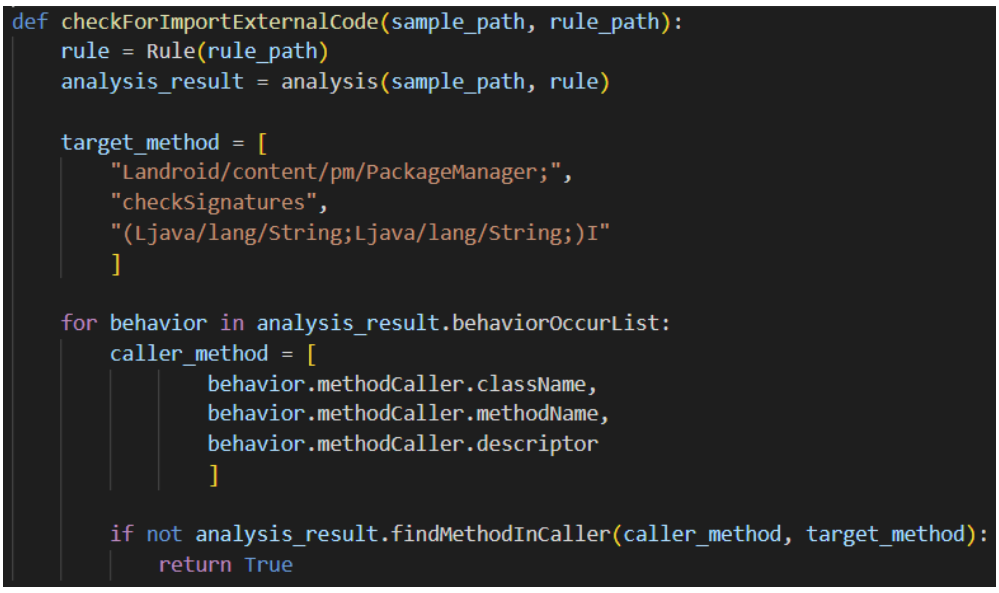
В ОС Android дана ситуація виникає якщо застосунок Android отримує та запускає код з іншого неперевіреного застосунку. Для реалізації даної ситуації, код зловмисного додатку повинен містити використання методу «createPackageContext» з класу «Context», котрий створив об’єкт зв’язку з іншим застосунком та завантаження його код за допомогою методу «loadClass». При цьому перевірка («PackageManager.checkSignatures») застосунку, код якого використовується, має бути відсутня.

Виходячи з наведених вище даних було створено наступне правило (рисунок 20):

```
{
    "rule": "Import and execute code from a different APK.",

    "severity": 4,

    "permission": [],

    "api": [
        {
            "class": "",
            "method": "createPackageContext",
            "descriptor": "(Ljava/lang/String;I)Landroid/content/Context;"
        },
        {
            "class": "Ljava/lang/ClassLoader;",
            "method": "loadClass",
            "descriptor": "(Ljava/lang/String;)Ljava/lang/Class;"
        }
    ]
}
```

Рисунок 20 – Правило виявлення ін’єкції коду в застосунку ОС Android

Однак, так як частиною логіки пошуку є відсутність використання перевірки застосунку, код якого використовується, то пошук даного небезпечного сценарію відбувається наступним чином (рисунок 21):



```
def checkForImportExternalCode(sample_path, rule_path):
    rule = Rule(rule_path)
    analysis_result = analysis(sample_path, rule)

    target_method = [
        "Landroid/content/pm/PackageManager;",
        "checkSignatures",
        "(Ljava/lang/String;Ljava/lang/String;)I"
        ]

    for behavior in analysis_result.behaviorOccurList:
        caller_method = [
                behavior.methodCaller.className,
                behavior.methodCaller.methodName,
                behavior.methodCaller.descriptor
                ]

        if not analysis_result.findMethodInCaller(caller_method, target_method):
            return True
```

Рисунок 21 – Виявлення ін’єкції коду в застосунку ОС Android

У даному випадку, для виявлення отримання та використання неперевіреного стороннього коду іншого додатку, код застосунку, що аналізується, перевіряється на наявність описаної в правилі поведінки. Якщо її було знайдено, то метод, в якому вона знаходиться, аналізується на наявність

перевірки зовнішнього додатку методом «PackageManager.checkSignatures». Відсутність використання даного методу підтверджує наявність описаної в даному прикладі зловмисної секції коду.

Другим тестовим випадком є виявлення наявності небезпеки SQL ін'єкцій у застосунку в ОС Android. За допомогою SQL-ін'єкції зловмисник може маніпулювати SQL-запитами, вводячи зловмисний код SQL за допомогою інструментів введення користувача. Це може призвести до несанкціонованого доступу, маніпуляції, втрати даних та інших серйозних наслідків.

Припустимо, що застосунок в ОС Android має поле для вводу певної інформації, котра потім зберігається програмою у базі даних. Якщо дані зберігають баз їх попередньої обробки, то даний сценарій несе пряму загрозу безпеці даних користувача, котрий має дає застосунок. Для маніпуляцій з БД в ОС Android використовується метод «SQLiteDatabase.rawQuery», котрий приймає строку з запитом. Запит будується шляхом поєднання синтаксису SQL з вже введеними даними. Виходячи з даного сценарію можна побудувати відповідне правило поведінки небезпечної секції коду (рисунок 22):

```
{
    "rule": "SQL Injection",

    "severity": 3,

    "permission": [],

    "api": [
        {
            "class": "Landroid/database/sqlite/SQLiteDatabase;",
            "method": "rawQuery",
            "descriptor": "(Ljava/lang/String; [Ljava/lang/String;)Landroid/database/Cursor;"
        }
    ]
}
```

Рисунок 22 – Правило поведінки SQL ін'єкції в застосунку ОС Android

Однак, для виявлення використання незмінного тексту вводу одного правила не достатньо. Необхідно провести додаткову маніпуляцію, а саме відслідкувати, чи використовує знайдений в сценарії метод виконання SQL запиту текст з компоненту вводу. Для цього потрібно перевірити аргументи, з

якими викликається метод «SQLiteDatabase.rawQuery» та знайти серед них використання введеного користувачем тексту. Це можна зробити за допомогою виявлення методу «EditText.getText», котрий повертає введений в застосунок текст.

Результуючий приклад використання розробленого інструменту на основі запропонованого в даній роботі способу можна побачити на рисунку 23:

```
def checkForSQLInjection(sample_path, rule_path):
    rule = Rule(rule_path)
    analysis_result = analysis(sample_path, rule)

    target_method = [
        "Landroid/widget/EditText;",
        "getText",
        "()Landroid/text/Editable;",
    ]

    for rawSqlCall in analysis_result.behaviorOccurList:
        if rawSqlCall.isArgFromMethod(
            target_method
        ):
            return True
```

Рисунок 23 - Виявлення небезпеки наявності SQL ін’єкції в застосунку ОС Android

Ще одним окремим випадком є приклад виявлення надання доступу до методів застосунку іншому сторонньому коду, що надає зловмиснику можливість завдання шкоди системі та призвести до небажаних наслідків. Це може надати зловмиснику ширшу зону атаки та потенційні можливості для використання.

У контексті ОС Android даний сценарій може статися, наприклад, якщо застосунок надає доступ до певного свого методу за допомогою JavaScript, який працює у компоненті веб-переглядачі WebView. Таким чином застосунок

дозволяє коду JavaScript викликати наданий метод, що становить значний ризик безпеки користувачу та системі.

Правило виявлення використання JavaScript у компоненті WebView Виглядає наступним чином (рисунок 24):

```
{
    "rule": "JavaScript usage by WebView",

    "severity": 3,

    "permission": [],

    "api": [
        {
            "class": "Landroid/webkit/WebView;",
            "method": "getSettings",
            "descriptor": "()Landroid/webkit/WebSettings;"
        },
        {
            "class": "Landroid/webkit/WebSettings;",
            "method": "setJavaScriptEnabled",
            "descriptor": "(Z)V"
        }
    ]
}
```


Рисунок 24 - Виявлення використання JavaScript у компоненті WebView застосунку

Дане правило описує поведінку секції коду, в якій компонент WebView налаштовується таким чином, що JavaScript під час його роботи є дозволеним (метод «setJavaScriptEnabled» під час виклику налаштувань «getSettings»). Однак, сам факт дозволу використання JavaScript в застосунку не є достатньою підставою, щоб назвати його зловмисним. Для цього, за допомогою додаткових маніпуляцій з представленням коду додатку, необхідно знайти виклик методу «addJavascriptInterface», котрий дозволяє використовувати код застосунку іншій стороні. Виявлення даного сценарію за допомогою розробленого інструменту на основі запропонованого в даній роботі способу можна побачити на рисунку 25:

```
def checkForJSExploit(sample_path, rule_path):
    rule = Rule(rule_path)
    analysis_result = analysis(sample_path, rule)

    targetMethod = [
        "Landroid/webkit/WebView;",
        "addJavascriptInterface",
        "(Ljava/lang/Object; Ljava/lang/String;)V"
    ]

    for eachJsExecution in analysis_result.behaviorOccurList:
        methodCaller = eachJsExecution.methodCaller
        secondaryAPI = eachJsExecution.API[1]

        jsEnabled = secondaryAPI.getArguments()[1]
        apiExposed = analysis_result.findMethodInCaller(methodCaller, targetMethod)

        if jsEnabled and apiExposed:
            return True
```

Рисунок 25 - Виявлення надання сторонньому коду доступу до методів застосунку

Таким чином, було розглянуто окремі тестові випадки, в яких крім використання запропонованого у даній роботі опису правил пошуку небезпечних секції коду проводились додаткові маніпуляції з метою отримання більш точного результату.

Висновки до п’ятого розділу

У підсумку, розроблений інструмент на основі статичного аналізу зміг виявити всі протестовані випадки шкідливого програмного забезпечення у підготовленому тестовому застосунку. Результат були коректними та надали повну інформацію про знайдені випадки зловмисного програмного забезпечення. Розроблений інструмент призначив відповідні оцінки серйозності та рівні достовірності для кожного виявленого випадку, що вказує на його ефективність у виявленні різних типів шкідливого програмного забезпечення.

Підсумовуючи, розроблений інструмент на основі статичного аналізу виявився ефективним у виявленні випадків зловмисного програмного забезпечення у застосунках ОС Android. Процес тестування продемонстрував потенціал інструменту як цінного активу в боротьбі із загрозами зловмисного програмного забезпечення Android.

Як і з будь-яким автоматизованим інструментом, важливо пам’ятати, що статичний аналіз має свої обмеження, і він може виявити не всі можливі випадки зловмисного програмного забезпечення або заплутаний шкідливий код. Тому вкрай важливо використовувати комбінацію методів статичного інших аналізів, а також інші заходи безпеки, щоб забезпечити повний захист мобільних пристроїв і програм.

ВИСНОВКИ

В даній магістерській роботі було поставлено задачу розробити спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу у застосунках ОС Android, котрий покликаний покращити гнучкість існуючих варіацій, а також отримувати більш детальну інформацію про застосунок, що аналізується, шляхом надання більш розгорнутого результату аналізу.

В першому розділі було наведено загальні відомості та проведено аналіз предметної області даної роботи. Як наслідок, було досліджено наявні програмні рішення та інструменти для аналізу безпеки мобільних додатків, а також визначено основні вимоги та виявлено певні закономірності у роботах, котрі базуються на статичному аналізі ПЗ.

Другий розділ розглядає архітектурні особливості операційної системи Android та її модель безпеки, з метою кращого розуміння існуючих механізмів та способу їх використання у виявленні зловмисного програмно забезпечення.

В третьому розділі було розглянуто методи та підходи до статичного аналізу програмного забезпечення, їх переваги та недоліки, а також описано наскільки ці методи можуть допомогти у виявленні потенційних проблем з безпекою в мобільних застосунка ОС Android. Крім того було розглянуто особливості аналізу застосунків у цій системі та наведено порядок побудови даного процесу.

Четвертий та п’ятий розділи представляють та перевіряють на тестових сценаріях розроблений в даній роботі спосіб виявлення зловмисного програмного забезпечення на основі статичного аналізу застосунку в ОС Android. Запропонований спосіб пропонує підхід до виявлення та оцінку ступеня загрози зловмисного програмного забезпечення на основі підготовленого набору файлів з описом поведінки зловмисного коду. Кожен з файлів має запропоновану в даній роботі структуру, котра дозволяє описати окремі сценарії небезпечних секцій коду, а також надати оцінку їх ступеня небезпеки.

Використання окремих файлів для зберігання моделей поведінки забезпечує гнучкий і масштабований спосіб керування та оновлення правил поведінки зловмисного програмного забезпечення. Це означає, що система може постійно розвиватися та адаптуватися до нових загроз з мінімальними затратами в плані зміни логіки роботи програмного забезпечення.

Не зважаючи на те, що запропонований спосіб насамперед розроблений для ОС Android, загальний підхід (тобто використання окремих файлів із запропонованою в роботі структурою для опису поведінки зловмисного програмного забезпечення) може бути розширений та потенційно застосований і для інших операційних систем.

Ефективне виявлення зловмисного програмного забезпечення за допомогою запропонованого методу може значно підвищити безпеку пристроїв Android. Це допоможе уникнути порушень конфіденційності даних користувачів та інших інцидентів з безпекою.

В цілому, запропонований в даній роботі спосіб представляє практичний підхід до виявлення зловмисного програмного забезпечення в застосунках ОС Android та має переваги з точки зору точності, ефективності та масштабованості.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. StatCounter Global Stats: Operating System Market Share. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення 02.02.2023).
2. WEI WANG, ZHENZHEN GAO, MEICHEN ZHAO, YIDONG LI, JIQIANG LIU, XIANGLIANG ZHANG.: DroidEnsemble: Detecting Android Malicious Applications With Ensemble of String and Structural Static Features. 2018. URL: https://www.researchgate.net/publication/325095237_DroidEnsemble_Detecting_Android_Malicious_Applications_With_Ensemble_of_String_and_Structural_Static_Features (дата звернення 07.02.2023).
3. S. Sathiya Keerthi, Chih-Jen Lin, and C.-J. Lin.: "Support Vector Machines: A Review". 2006. URL: https://ieeexplore.ieee.org/document/1046920 (дата звернення 07.02.2023).
4. Alpaydin, E.: "The k-Nearest Neighbors Algorithm". 2010. URL: https://link.springer.com/article/10.1007/s12065-009-0023-7 (дата звернення 07.02.2023).
5. Leo Breiman: "Random Forests". 2001. URL: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf (дата звернення 07.02.2023).
6. H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck: “Structural detection of android malware using embedded call graphs”, 2013, pp. 45–54.
7. Babu Rajesh V, Phaninder Reddy, Himanshu P and Mahesh U Patil. DROIDSWAN: DETECTING MALICIOUS ANDROID APPLICATIONS BASED ON STATIC FEATURE ANALYSIS. URL: https://www.researchgate.net/publication/307797420_DROIDSWAN_DETECTING_MALICIOUS_ANDROID_APPLICATIONS_BASED_ON_STATIC_FEATURE_ANALYSIS (дата звернення 10.02.2023).

8. Yu Feng, Saswat Anand, Isil Dillig, Alex Aiken: “Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis”. URL: https://www.cs.utexas.edu/~isil/fse14.pdf (дата звернення 10.02.2023).
9. Miller, R. J., & Riedewald, M. (Eds.). “Datalog in Academia and Industry”. Morgan & Claypool Publishers, 2011.
10. L. O. Andersen: “Program analysis and specialization for the C programming language”. PhD thesis, University of Cophenhagen, 1994.
11. Nikolay Elenkov Foreword by Jon Sawyer: “Android Security Internals Android Security Internals An In-Depth Guide to Android’s Security Architecture”. 2015.
12. Bahman Rashidi and Carol Fung: “A Survey of Android Security Threats and Defenses”. Virginia Commonwealth University, Richmond, Virginia, USA, 2015.
13. APK File Structure. URL: https://sec-art.net/2022/01/21/android-pentesting-series-part-2-apk-file-structure/ (дата звернення 15.02.2023).
14. System and kernel security. URL: https://source.android.com/security/overview/kernel-security (дата звернення 15.02.2023).
15. Android, “Google android documents, android application sandboxing mechanism”. URL: https://source.android.com/security/app-sandbox (дата звернення 15.02.2023).
16. Security-Enhanced Linux in Android. URL: https://source.android.com/security/selinux (дата звернення 17.02.2023).
17. Verified Boot. URL: https://source.android.com/security/verifiedboot (дата звернення 19.02.2023).
18. Encryption. URL: https://source.android.com/security/encryption (дата звернення 22.02.2023).
19. Application security. URL: https://source.android.com/security/overview/app-security (дата звернення 25.02.2023).

20. RENÉ MAYRHOFER, JEFFREY VANDER STOEP, CHAD BRUBAKER, NICK KRALEVICH. “The Android Platform Security Model”. URL: https://dl.acm.org/doi/10.1145/3448609 (дата звернення 01.03.2023).
21. Application Signing. URL: https://source.android.com/security/apksigning (дата звернення 01.03.2023).
22. Amr Amin, Amgad Eldessouki, Menna Tullah Magdy, Nouran Abdeen, Hanan Hindy, Islam Hegazy AndroShield: “Automated Android Applications Vulnerability Detection, a Hybrid Static and Dynamic Analysis Approach”. “Information” (Switzerland), 2019, pp. 2-3.
23. Andersen, L.O.: “Program Analysis and Specialization for the C Programming Language.” Ph.D. Thesis,University of Cophenhagen, Copenhagen, Denmark, 1994.
24. “Steensgaard, B. Points-to analysis in almost linear time. In Proceedings of the 23rd ACM SIGPLAN-SIGACT”. Symposium on Principles of Programming Languages, St. Petersburg Beach, FL, USA, 21–24 January 1996;pp. 32–41.
25. Li Lia, Tegawende F. Bissyand, Mike Papadakisa, Siegfried Rasthoferb, Alexandre Bartela, Damien Octeauc, Jacques Kleina, Yves Le Traon, “Static Analysis of Android Apps: A Systematic Literature”. “Information and Software Technology”, 2017, 88, pp. 67-95.