Національний університет «Чернігівська політехніка»
Міністерство освіти і науки України
Національний університет «Чернігівська політехніка»
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

КАРПАЧЕВ ІГОР ІГОРОВИЧ

УДК 004.942; 004.021

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ БЕЗПЕКИ МОБІЛЬНИХ ПРИСТРОЇВ

05.13.06 – інформаційні технології

технічні науки

галузь знань

Подається на здобуття наукового ступеня кандидата технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Карпачев Ігор Ігорович

підпис

Науковий керівник (консультант) Казимир Володимир Вікторович,
доктор технічних наук, професор

Чернігів – 2021

АНОТАЦІЯ

Карпачев Ігор Ігорович. Інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.06 «Інформаційні технології». – Національний університет «Чернігівська політехніка», Чернігів, 2021.

Захист роботи відбудеться в Чернігівському національному університеті «Чернігівська політехніка», МОН України, м. Чернігів.

У роботі вирішено актуальне *наукове завдання* із подальшого розвитку інформаційної технології забезпечення функціональної безпеки мобільних пристроїв за рахунок удосконалення існуючої моделі безпеки ОС Android шляхом впровадження методу динамічного виявлення потенційно небезпечних додатків з метою подальшого їх блокування для запобігання небажаних впливів.

Розроблена інформаційна технологія враховує результати попередніх досліджень та забезпечує вибір оптимальної стратегії протидії виявленому шкідливому програмному забезпеченню при прийнятті рішень з урахуванням ризику.

Мета дисертаційної роботи полягає в покращенні функціональної безпеки мобільних пристроїв за рахунок удосконалення моделі безпеки ОС Android з можливістю врахування ризику використання шкідливих програмних додатків.

Досягнення поставленої мети передбачає вирішення таких завдань:

1. Визначення потенційних загроз функціонуванню мобільних пристроїв.
2. Аналіз існуючої системи безпеки ОС Android щодо загроз безпечній роботі програмних додатків.
3. Розробка моделі прав доступу при взаємодії ОС Android із програмними додатками.

4. Розробка методів динамічного виявлення та блокування потенційно небезпечних додатків.

5. Розробка програмних засобів реалізації запропонованих методів забезпечення функціональної безпеки.

6. Оцінка ефективності розроблених моделей, методів та інформаційної технології шляхом проведення експериментів із реальними приладами та додатками.

Об'єкт дослідження – система функціональної безпеки ОС Android та процес її функціонування в умовах зовнішніх загроз.

Предмет дослідження – моделі та методи забезпечення функціональної безпеки на рівні операційної системи мобільних пристроїв.

Основні результати дослідження та наукова новизна роботи полягають в тому, що було розроблено інформаційну технологію виявлення потенційно небезпечних програмних додатків задля забезпечення функціональної безпеки мобільних пристроїв, що функціонують на базі операційних систем Android.

Проаналізовано множину напрацювань у сфері функціональної безпеки, з яких формується концепція безпеки мобільного пристрою. Встановлено, що питання функціональної безпеки мобільних пристроїв, які базуються на ОС Android, в умовах динамічного росту кількості додатків, їх популярності, а отже, і кількості користувачів, залишається до кінця не вирішеним. Зважаючи на це, зроблено висновок про те, що актуальним є наукове завдання з розробки інформаційної технології забезпечення функціональної безпеки мобільних пристроїв за рахунок удосконалення існуючої моделі безпеки ОС Android шляхом впровадження методу динамічного виявлення потенційно небезпечних додатків з метою подальшого їх блокування для запобігання небажаних впливів.

У розрізі наукового завдання розроблена модель сигнатури функціонального ланцюжка додатку, яка використовується для знаходження міри ідентичності та подібності між аналізуємим додатком та додатками з бази

даних шкідливих додатків. Сформульовані особливості побудови та аналізу сигнатури функціонального ланцюжку додатку під час його виконання, визначено антиемуляційні заходи, складність за наявності у вихідному коді циклів, анонімних класів, або декількох потоків, які виконуються паралельно. Проаналізовано методи вирівнювання послідовностей із біоіформатики, які можуть бути використані для знаходження коефіцієнтів ідентичності та подібності між аналізуємим додатком та додатками з бази даних шкідливого програмного забезпечення.

Вперше запропонована класифікація типів небезпечних додатків, яка, на відміну від існуючих, базується на групуванні API функцій додатків та дає можливість оцінити їх по ступеню потенційних впливів при прийнятті рішень на використання додатків.

Вперше розроблена модель прав доступу при взаємодії ОС Android із застосуваннями, яка, на відміну від існуючих, встановлює відношення між групами, дозволами та функціями API та дає можливість ввести кодування функцій для їх ідентифікації.

Удосконалено метод динамічного виявлення потенційно небезпечних додатків, який, на відміну від існуючих, використовує ланцюжки API-функцій, що будуються під час виконання додатків, та дозволяє оцінити ризик при ідентифікації шкідливих програмних додатків.

Набула подальшого розвитку інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв, яка на відміну від існуючих, дозволяє здійснити динамічне управління процесами використання додатків ОС Android за рахунок аналізу дозволів на використання ресурсів.

Практичне значення отриманих результатів полягає в тому, що розроблена інформаційна технологія має практичне втілення у вигляді програмного комплексу, до складу якого входять два програмних засоби, які працюють з використанням хмарного середовища та реалізовані на мовах C#, Kotlin і JavaScript:

- програмний засіб AMalDetector, який призначений для моніторингу процесу виконання мобільних додатків та підтримки прийняття рішення щодо шкідливості мобільного ПЗ шляхом комунікації із серверним аналізуючим ядром CMMD та прийняття остаточного рішення базуючись на результатах аналізу СФЛД;

- програмний засіб CMMD, який призначений для обчислення схожості та подібності аналізованої та шаблонної СФЛД, шляхом сиквенційної агрегації фрагментованих API функцій та використання методів глобального та локального порівняння послідовностей.

Розроблена інформаційна технологія та програмний комплекс перевірені в умовах практичного проведення дослідження на реальних мобільних додатках. Проведені тести підтвердили здатність запропонованої інформаційної технології підвищувати ефективність системи функціональної безпеки мобільних пристроїв на базі ОС Android за істинним позитивним показником (TPR) на рівні 99% при низькому хибно позитивному показнику (FPR) на рівні 0,2% з одночасним обчислюванням коефіцієнтів ідентичності та подібності для порівнюваних ланцюжків викликів API функцій.

Результати дисертаційних досліджень впроваджені:

- при виконанні міжнародного проєкту Tempus 530319-TEMPUS-1-2012-1-DE-TEMPUS-JPHES «Інноваційна гібридна стратегія IT-аутсорсингового партнерства з підприємствами (IHSITOP)»;

- при виконанні НДР «Розробка програмно-апаратного комплексу захищеної системи електронного голосування «Mobile-RADA» за договорами НУ «Чернігівська політехніка» з Чернігівською обласною радою № 480/18, 492/19, 508/20 та з Корюківською міською радою № 79/482/19;

- при створенні мобільного додатка «Datatouch (Datascope Ltd.)» для будівельних компаній та системи управління доставками «DMS (Datascope Ltd.);

- у навчальному процесі на кафедрі інформаційних та комп’ютерних систем НУ «Чернігівська політехніка» при проведенні лекцій та лабораторних

робіт із дисципліни «Програмування мобільних пристроїв» у процесі навчання бакалаврів спеціальності 123 – комп’ютерна інженерія та з дисципліни «Статистичні методи обробки інформації» в процесі навчання аспірантів спеціальності 122 – комп’ютерні науки.

Ключові слова: мобільний пристрій, функціональна безпека, ОС Android, модель прав доступу, API функція, сигнатура функціонального ланцюжка додатку.

ABSTRACT

Karpachev Ihor Ihorovich. Information technology to ensure the functional safety of mobile devices. – Qualifying scientific work on the rights of the manuscript.

The dissertation for the degree of the candidate of technical sciences (philosophy doctor) in specialty 05.13.06 "Information technologies". – National University "Chernihiv Polytechnic", Chernihiv, 2021.

The defending of the work will take place at National University "Chernihiv Polytechnic", Ministry of Education and Science of Ukraine, Chernihiv.

The dissertation solves the current scientific problem of further development of information technology to ensure the functional security of mobile devices by improving the existing security model of Android OS by introducing a method of dynamic detection of potentially dangerous applications in order to further block them to prevent unwanted effects.

The developed information technology takes into account the results of previous research and provides the choice of the optimal strategy for counteracting the detected malware when making risk-based decisions.

The purpose of the dissertation is to improve the functional security of mobile devices by improving the security model of the Android OS with the ability to take into account the risk of using malicious software applications.

Achieving this goal involves solving the following tasks:

1. Identification of potential threats to the operation of mobile devices.

2. Analysis of the existing security system of the Android OS for threats to the safe operation of software applications.

3. Development of a model of access rights for Android OS interaction with software applications.

4. Development of methods for dynamic detection and blocking of potentially dangerous applications.

5. Development of software for the implementation of the proposed methods of functional safety.

6. Evaluation of the effectiveness of the developed models, methods and information technology by conducting experiments with real devices and applications.

The object of research is the functional security system of the Android OS and the process of its operation in the conditions of external threats.

The subject of research - models and methods of ensuring functional security at the level of the operating system of mobile devices.

The main results of the study and the scientific novelty of the work are that the information technology for detecting potentially dangerous software applications was developed to ensure the functional security of mobile devices running on Android operating systems.

Many developments in the field of functional security, from which the concept of mobile device security is formed, are analyzed. It is established that the issue of functional security of mobile devices based on Android OS, in the conditions of dynamic growth of the number of applications, their popularity and, consequently, the number of users, remains unresolved. In view of this, it is concluded that the scientific task of developing information technology to ensure the functional security of mobile devices by improving the existing security model of Android by introducing a method of dynamic detection of potentially dangerous applications to further block them to prevent unwanted effects.

In the context of the scientific task, a signature model of the functional chain of the application is developed, which is used to find the degree of identity and similarity between the analyzed application and applications from the database of malicious applications. The peculiarities of construction and analysis of the signature of the functional chain of the application during its execution are formulated, antiemulation measures, complexity in the presence of cycles, anonymous classes, or several streams that are executed in parallel are defined. The methods of sequence

alignment in bioinformatics, which can be used to find the coefficients of identity and similarity between the analyzed application and applications from the malware database, are analyzed.

For the first time, a classification of types of dangerous applications is proposed, which, unlike existing ones, is based on the grouping of API functions of applications and makes it possible to assess them by the degree of potential impact on decisions on the use of applications.

For the first time, a model of access rights was developed when Android OS interacts with applications, which, unlike existing ones, establishes relationships between groups, permissions and API functions and allows you to enter the encoding of functions to identify them.

The method of dynamic detection of potentially dangerous applications has been improved, which, unlike existing ones, uses chains of API functions that are built during the execution of applications, and allows to assess the risk in identifying malicious software applications.

The information technology for ensuring the functional security of mobile devices has been further developed, which, in contrast to the existing ones, allows for dynamic control of the processes of using Android applications by analyzing resource permissions.

The practical significance of the obtained results is that the developed information technology has a practical embodiment in the form of a software package, which includes two software tools that work using the cloud environment and are implemented in C #, Kotlin and JavaScript:

- AMalDetector software, which is designed to monitor the process of mobile applications and support the decision on the harmfulness of mobile software by communicating with the server analysis core CMMD and making a final decision based on the results of SFLD analysis;

- CMMD software, which is designed to calculate the similarity and similarity of the analyzed and template SFLD, by sequential aggregation of fragmented API functions and the use of methods of global and local sequence comparison.

The developed information technology and software are tested in terms of practical research on real mobile applications. The tests confirmed the ability of the proposed information technology to increase the efficiency of the functional security system of mobile devices based on Android OS with a true positive indicator (TPR) of 99% with a low false positive index (FPR) of 0.2% while calculating the coefficients of identity and similarity for comparable function API call chains.

The results of dissertation research are implemented:

- in the implementation of the international project Tempus 530319-TEMPUS-1-2012-1-DE-TEMPUS-JPHES "Innovative hybrid strategy of IT outsourcing partnership with enterprises (IHSITOP)";

- in the implementation of research and development of software and hardware complex of secure electronic voting system "Mobile-RADA" under the agreements of NU "Chernihiv Polytechnic" with Chernihiv Regional Council № 480/18, 492/19, 508/20 and with Koryukiv City Council № 79 / 482/19;

- when creating a mobile application "Datatouch (Datascope Ltd.)" for construction companies and delivery management system DMS (Datascope Ltd.);

- in the educational process at the Department of Information and Computer Systems NU "Chernihiv Polytechnic" during lectures and laboratory work on the subject "Programming of mobile devices" in the process of teaching bachelors specialty 123 - computer engineering and the discipline "Statistical methods of information processing" in the process of teaching graduate students majoring in 122 - computer science.

Keywords: mobile device, functional security, Android OS, access rights model, API function, application function chain signature.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Казимир В., Карпачев І. Функціональна безпека архітектури мобільної операційної системи Android. *Технічні науки та технології*: науковий журнал. 2015. № 1 (77). С. 127-134.

2. Kazymyr V., Karpachev I. Improving of existing permission system in Android OS. *International Journal "Information Models and Analyses*". 2015. Vol.4. P. 336-344. ISSN 1314-6416 (printed). ISSN 1314-6432 (Online).

3. Kazymyr V., Karpachev I. Improving time-critical code performance with JNI. *Mathematical machines and systems*. 2016. Vol. 2.P.72-77.

4. Казимир В., Карпачев І. Забезпечення контролю доступу застосувань до ресурсів ОС Android методом системної безпеки. *Технічні науки та технології*: науковий журнал. 2016. № 4 (6). С. 131-138.

5. Казимир В., Карпачев І., Усик А. Моделі систем безпеки ОС Android. *Технічні науки та технології*: науковий журнал. 2018. № 2 (12). С. 116-126.

6. Казимир В., Карпачев І., Сіпаков В. Динамічний аналіз послідовностей API-викликів OS Android. *Технічні науки та технології*: науковий журнал. 2019. № 4 (18). С. 85-91.

7. Карпачев І., Казимир В. Виявлення шкідливих додатків ОС Android по сигнатурі функціонального ланцюжка додатку. *Технічні науки та технології*: науковий журнал. 2021. № 1 (23). С. 109-117.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Карпачев И.И. Системо-центрическая безопасность в Android. Математичне та імітаційне моделювання систем «МОДС 2015» : X Міжнародна науково-практична конференція (Чернігів, 22-26 Червня 2015 р.). Чернігів, 2015. С. 422-424.

2. Karpachev I. Bounded context in strategic domain-driven design // *Математичне та імітаційне моделювання систем «МОДС 2016»*: XI Міжнародна науково-практична конференція (м. Чернігів-с. Жукін, 27 червня – 1 липня 2016 р.). Чернігів, 2016. С. 398-400.

3. Казимир В.В., Карпачев І.І., Литвин С.В., Усік А.М. Архітектура моделей системи безпеки мережі інтернету речей на базі ОС Android. *Математичне та імітаційне моделювання систем «МОДС 2018»* : XIII Міжнародна науково-практична конференція (Київ-Чернігів-Жукін, 25-29 червня 2018 р.). Чернігів, 2018. С. 335-336.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СФЛД	сигнатура функціонального ланцюжку додатку;
ПЗ	програмне забезпечення;
API	application programming interface;
APK	application package;
AMD	android malware detector;
CMMD	cloud mobile malware detector;
DL	device layer;
FTS	full text search;
KNN	K-nearest neighbour;
MLP	multilayer perceptron;
P2M	people to machine;
SDK	software development kit;
SL	service layer;

ВСТУП

Обґрунтування вибору теми. У наш час мобільні пристрої дедалі більше стають характерною рисою і необхідним елементом як технологічного розвитку, так і життєзабезпечення. Поступово в умовах всеосяжної діджіталізації вони проникають у всі сфери життя людини. Завдяки мобільним пристроям здійснюється управління складними кіберфізичними системами, у тому числі через мережі Інтернету речей, стає доступним широкий спектр фінансових та соціальних послуг, організується обмін інформацією та моніторинг життєвоважливих функцій. Фактично, такого роду пристрої переходять із розряду іграшкових приладів у категорію критичних застосунків.

Питання забезпечення функціональної безпеки критичних застосунків достатньо глибоко висвітлено в наукових працях відомих закордонних та вітчизняних науковців, таких як A. Avizienis, G. Johnson, J.C. Laprie, B. Randell, Є. В. Брежнєв, О. М. Одарущенко, В. В. Скляр, В. С. Харченко. Виходячи із сформованої вказаними вченими загальної таксономії понять загальна проблема забезпечення функціональної безпеки належить до систем моніторингу та управління. На сучасному етапі функціональна безпека повинна спиратися на інформаційну безпеку та кібербезпеку, що доповнюють одна одну. У той час як мета інформаційної безпеки полягає в забезпеченні доступності, цілісності й конфіденційності даних системи, функціональна безпека забезпечує здатність системи виконувати передбачені функції при існуванні ризику виникнення небезпечних подій, в тому числі під впливом зовнішніх загроз.

Що стосується безпосередньо мобільних пристроїв, то тут особливо гостро питання функціональної безпеки постає перед найбільш поширеними реалізаціями на базі ОС Android, дослідженню функціональної безпеки яких присвячена значна кількість публікацій сучасних науковців та фахівців наукових лабораторій IT-корпорацій: J. Warton, P. Roche, S. Felker, A. Саммерс, K. Gopal, A. Davies, M. Zhan, R. Mayer, A. Dao, D. Smith, T. Norby, M. Alison,

С. В. Зайцев, М. С. Дорош, І. В. Стеценко та ін. Отримані в межах даних досліджень результати вказують на те, що найбільший рівень загрози слід пов'язувати не з апаратними ресурсами та операційною системою (ОС), а саме з мобільними додатками, що містять у собі такі вразливості, як недосконалості у визначенні привілеїв користувачів, витік інформації, низький рівень захисту функціональних компонентів, уразливості міжкомпонентних комунікацій та інші. Більшість існуючих заходів безпеки, що надаються ОС Android, базуються на попереджувальних діях та системних обмеженнях для забезпечення безпеки платформи загалом.

Слід зазначити, що підходи на основі машинного навчання, статичного аналізу та штучних нейромережевих алгоритмів, що дозволяють визначити рівень загрози на основі базових параметрів програмного коду, наприклад, при аналізі викликів API (Application Programming Interface), також не забезпечують належний рівень функціональної безпеки. Враховуючи значну затримку у вирішенні питань безпеки та потенційний ризик втрати приватних даних або навіть порушення функціональної безпеки системи, існує потреба в додатковому блоці безпеки, який допоможе повідомити користувача про потенційно шкідливе програмне забезпечення під час виконання. Тому актуальним є наукове завдання із подальшого розвитку інформаційної технології забезпечення функціональної безпеки мобільних пристроїв за рахунок удосконалення існуючої моделі безпеки ОС Android шляхом впровадження методу динамічного виявлення потенційно небезпечних додатків з метою подальшого їх блокування для запобігання небажаних впливів

Зв'язок роботи з науковими програмами, планами, темами. Результати дисертаційної роботи використані при розробці мобільного додатку в межах європейського проєкту за програмою Tempus-IV «Інноваційна гібридна стратегія IT-аутсорсингового партнерства (IHSITOP)», у комерційних науково-дослідних роботах Національного університету «Чернігівська політехніка» при створенні захищеної системи електронного голосування «Mobile-RADA»,

мобільному додатку «Datatouch (Datascope Ltd.)» для будівельних компаній та розповсюдженій системі доставок «DMS (Datascope Ltd.)».

Об'єкт дослідження – система функціональної безпеки ОС Android та процес її функціонування в умовах зовнішніх загроз.

Предмет дослідження – моделі та методи забезпечення функціональної безпеки на рівні операційної системи мобільних пристроїв.

Мета і завдання дослідження. Мета дисертаційної роботи полягає в покращенні функціональної безпеки мобільних пристроїв за рахунок удосконалення моделі безпеки ОС Android з можливістю врахування ризику використання шкідливих програмних додатків.

Досягнення поставленої мети передбачає вирішення таких завдань:

- визначення потенційних загроз функціонуванню мобільних пристроїв.
- аналіз існуючої системи безпеки ОС Android щодо загроз безпечній роботі програмних додатків.
- розробка моделі прав доступу при взаємодії ОС Android із програмними додатками.
- розробка методів динамічного виявлення та блокування потенційно небезпечних додатків.
- розробка програмних засобів реалізації запропонованих методів забезпечення функціональної безпеки.
- оцінка ефективності розроблених моделей, методів та інформаційної технології шляхом проведення експериментів із реальними пристроями та додатками.

Методи дослідження. При розв'язанні поставлених завдань були використані: методи системного аналізу та методи теорії множин – при аналізі системи безпеки ОС Android та розробці моделі прав доступу, методи біоінформатики – у процесі розробці методу динамічного виявлення потенційно небезпечних додатків, теорії ймовірностей та математичної статистики – для планування та статистичного аналізу результатів

експериментів із реальними додатками, об’єктно-орієнтованого аналізу та графічні нотації UML – при проєктуванні та розробці програмних засобів, які реалізують запропоновану інформаційну технологію забезпечення функціональної безпеки мобільних пристроїв.

Наукова новизна одержаних результатів.

1) **Вперше** запропонована класифікація типів небезпечних додатків, яка, на відміну від існуючих, базується на групуванні дозволів на використання API функцій за ступенем потенційних впливів, що дає можливість оцінити додатки за рівнем небезпеки для користувача при прийнятті рішень на їх використання.

2) **Вперше р**озроблена модель прав доступу при взаємодії ОС Android з програмними додатками, яка, на відміну від існуючих, встановлює відношення між групами, дозволами та API функціями, що дає можливість використовувати псевдосимволи функцій для прискорення ідентифікації додатків.

3) **Удосконалено** метод динамічного виявлення потенційно небезпечних додатків за рахунок використання сигнатур функціональних ланцюжків додатків, що будуються та порівнюються під час виконання додатків, що дозволяє оцінити ризик при їх ідентифікації.

4) **Набула подальшого розвитку** інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв, яка, на відміну від існуючих, дозволяє здійснити динамічне управління процесами використання додатків ОС Android за рахунок аналізу виявленого вектору атаки.

Практичне значення одержаних результатів.

Розроблена інформаційна технологія забезпечення функціональної безпеки мобільних додатків має практичне втілення у вигляді програмного комплексу, до складу якого входять два розроблені програмні засоби:

- програмний засіб AMalDetector, який призначений для моніторингу процесу виконання мобільних додатків та підтримки прийняття рішення щодо шкідливості мобільного програмного забезпечення шляхом комунікації з серверним аналізованим ядром CMMD;

- програмний засіб CMMD, який призначений для виявлення схожості та ідентичності аналізуємої та шаблонної сигнатур функціональних ланцюжків додатку (СФЛД) шляхом сиквенційної агрегації фрагментованих викликів API функцій із використанням методів глобального та локального вирівнювання послідовностей.

Розроблена інформаційна технологія та програмний комплекс перевірені в умовах практичного проведення дослідження на реальних мобільних додатках. Проведені тести підтвердили здатність запропонованої інформаційної технології підвищувати ефективність системи функціональної безпеки мобільних пристроїв на базі ОС Android за істинним позитивним показником (TPR) на рівні 99% при низькому хибно позитивному показнику (FPR) на рівні 0,02% з одночасним обчислюванням коефіцієнтів ідентичності та подібності для порівнюваних ланцюжків викликів API функцій.

Результати дисертаційних досліджень впроваджені:

- при виконанні міжнародного проєкту Tempus 530319-TEMPUS-1-2012-1-DE-TEMPUS-JPHES «Інноваційна гібридна стратегія IT-аутсорсингового партнерства з підприємствами (IHSITOP)»;

- при виконанні НДР «Розробка програмно-апаратного комплексу захищеної системи електронного голосування «Mobile-RADA» за договорами НУ «Чернігівська політехніка» з Чернігівською обласною радою № 480/18, 492/19, 508/20 та з Корюківською міською радою № 79/482/19;

- при створенні мобільного додатка «Datatouch (Datascope Ltd.)» для будівельних компаній та системи управління доставками «DMS (Datascope Ltd.);

- у навчальному процесі на кафедрі інформаційних та комп'ютерних систем НУ «Чернігівська політехніка» при проведенні лекцій та лабораторних робіт із дисципліни «Програмування мобільних пристроїв» у процесі навчання бакалаврів спеціальності 123 – комп'ютерна інженерія та з дисципліни «Статистичні методи обробки інформації» в процесі навчання аспірантів спеціальності 122 – комп'ютерні науки.

Особистий внесок здобувача. Усі результати, які виносяться до захисту, отримані автором особисто. У роботах [58, 89] здобувачу належать усі теоретичні та практичні результати; також проведено аналіз та обґрунтовано недосконаліть існуючої система привілеїв ОС Android у площині функціональної безпеки; запропоновано методи покращення захисту ОС від шкідливого програмного забезпечення. У роботах [82, 114] здобувачем обґрунтовано використання Java Native Interface (JNI) у критичних з погляду продуктивності частинах програмної системи для підвищення швидкості виконання мобільного програмного додатка, описані розроблені здобувачем моделі та практичні результати експериментів з ними. У роботах [134, 44, 98] здобувачеві належить розробка та реалізація інформаційної технології забезпечення функціональної безпеки ресурсами ОС Android, методів побудови СФЛД на базі динамічного функціонального трасування API викликів, способів обробки послідовностей за допомогою локального та глобального методів порівняння СФЛД, математична модель прав доступу.

Апробація результатів роботи. Основні наукові та практичні результати дисертаційної роботи доповідались та обговорювались на наукових конференціях:

- X Міжнародна науково-практична конференція «Математичне та імітаційне моделювання систем», MODS-2015 (Чернігів-Київ, 22-26 червня, 2015);

- XI Міжнародна науково-практична конференція «Математичне та імітаційне моделювання систем», MODS-2016 (Чернігів, 27 червня – 1 липня, 2016);

- XIII Міжнародна науково-практична конференція «Математичне та імітаційне моделювання систем», MODS-2018 (Київ – Чернігів – Жукин, 25-29 червня, 2018).

Публікації. За матеріалами дисертаційної роботи опубліковано 10 наукових праць, серед яких 1 стаття у періодичному науковому виданні Європейського Союзу з наукового напряму дисертації, 6 статей у наукових

фахових виданнях України та 3 публікації у збірниках матеріалів міжнародних наукових конференцій.

Структура та обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та 5 додатків. Повний обсяг дисертації становить 141 сторінку, у тому числі: 110 сторінок основного тексту, 52 рисунків, 9 таблиць, список використаних джерел із 142 найменувань та 5 додатків.

РОЗДІЛ 1
ЗАДАЧІ ТА МЕХАНІЗМИ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ БЕЗПЕКИ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Визначення та перспективи використання мобільних пристроїв

Мобільні пристрої – це портативні комп'ютери, які поєднують в собі обчислювальні можливості звичайних стаціонарних комп'ютерів, оснащених дісплеями і клавіатурою, з доступом до Інтернету через бездротові мережі або за допомогою стільникового зв'язку. До класу мобільних пристроїв зазвичай відносять планшети, електронні зчитувачі, нетбуки, телефони та смартфони. Завдяки зручності використання та достатньо високій швидкодії мобільні пристрої стають все більш популярними серед користувачів. За даними Radicati Group [**Error! Reference source not found.**] загальна кількість користувачів мобільних пристроїв зросте з 6,8 мільярдів у 2019 році до більше ніж 7,3 млрд до кінця 2023 року, а загальна кількість мобільних пристроїв, включаючи телефони та планшети, у 2023 році перевищить 16,8 млрд (рис. 1.1).

Рік	Користувачів мобільних пристроїв у світі (мільйонів)	Всього мобільних пристроїв (мільйонів)	Мобільних пристроїв на одного бізнес-користувача
2019	6802	13092	1.92
2020	6951	14017	2.02
2021	7101	14913	2.10
2022	7255	15961	2.20
2023	7332	16804	2.29

Рисунок 1.1 – Світовий тренд мобільних пристроїв, 2019–2023

На фоні загальної статистики особливо вражаючим є зростання числа смартфонів (рис. 1.2) [2].

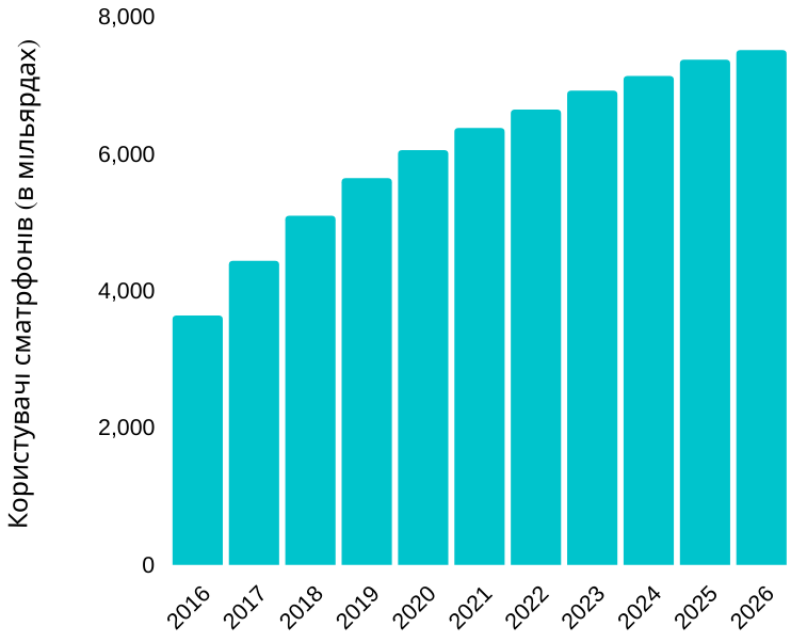


Рисунок 1.2 – Світовий тренд смартфонів, 2016–2026

Смартфони стають також найпопулярнішим типом мобільних телефонів. Остання статистика користувачів смартфонів показує, що з усіх мобільних телефонів, які використовуються сьогодні, більше 75 відсотків - це смартфони. Кількість глобальних користувачів смартфонів буде продовжувати зростати і досягне 4,3 млрд до 2023 року. Враховуючи очікуване вісім мільярдів населення, рівень проникнення смартфонів у 2023 році становитиме 53,8 відсотка. Тож, більше половини населення планети буде оснащено смартфоном. Це призводить до того, що смартфони стають постіним атрибутом життя людини, супроводжуючи їх та надаючі життєво необхідні можливості.

Для України питання, пов’язані із використанням мобільних пристроїв, та смартфонів закрема, є особливо важливим в руслі схваленної Кабінетом Міністрів «Концепції розвитку цифрової економіки та суспільства України на 2018-2020» [3]. Застосунок «Дія» [4] поступово стає засобом життєзабезпечення, надаючи можливості використання смартфонів у всіх критичних ситуаціях. Все це висуває особливо високі вимоги до функціональної безпеки мобільних пристроїв та їх програмного забезпечення.

1.2 Функціональна безпека мобільних пристроїв

З точки зору забезпечення ефективної та безпечної роботи мобільних пристроїв інформаційна безпека повинна розглядатися як складова функціональної безпеки. У той час як мета інформаційної безпеки полягає в забезпеченні доступності, цілісності й конфіденційності даних системи, блок функціональної безпеки забезпечує коректне виконання функцій системи управління і переведення об’єктів управління в безпечний стан у разі виникнення системних відмов [5, 6, 7, 8, 9].

Важливим етапом побудови стратегії функціональної безпеки мобільних застосувань є розуміння процесу сертифікації обладнання та ліцензування програмного забезпечення [9, 10]. Програмні та апаратні системи управління включають у себе функції безпеки й характеризуються надійністю резервування даних, відмовостійкістю, якістю самодіагностики та стійкістю до зовнішніх факторів, що мають бути визначені при розробці мобільних додатків. Системний аналіз роботи мобільних застосувань обов’язково включає у себе блок забезпечення функціональної безпеки, що здійснює моніторинг потенційно небезпечних умов та ідентифікує відповідні події, що можуть призвести до втрати даних, доступу до конфіденційних даних сторонніх осіб або блокування сторонніми особами належного доступу.

1.2.1 Поняття функціональної безпеки

Поняття функціональної безпеки є складовою частиною поняття загальної безпеки, що обмежується аналізом надійності роботи системи керування та обладнання, що пов’язане з цією системою, а також ймовірних помилок оператора й потенційних змін у середовищі, що можуть вплинути на ефективність функціонування загального комплексу [9-11]. Таким чином, метою функціональної безпеки є розробка методології, що включає у себе попереджувальні заходи для уникнення ситуацій, пов’язаних із ризиком для

функціонування системи керування і її складових [12], що включає у себе розгляд стратегій зловмисників та визначення показників допустимого рівня ризику. При забезпеченні функціональної безпеки ризик визначається як ймовірність виникнення певної події відповідно до ступеня її потенційної небезпеки, що дозволяє побудувати математичну модель та отримати оптимальний підхід через вирішення задачі пошуку оптимального шляху по графу. Проведення заходів із забезпечення функціональної безпеки можна розглядати як кінцевий етап виконання проєкту, на якому розглядаються функціональні властивості компонентів комплексу як окремих підсистем та складових загальної функції роботи всієї системи. Функціональні стандарти безпеки при цьому розповсюджуються на контроль та моніторинг електронних та програмних вузлів комплексу, а стратегія забезпечення загальної безпеки реалізується лише за умови визначення та виконання кожної окремої функції безпеки на відповідному рівні. Переважно це включає у себе виконання таких кроків [10, 13]:

- визначення потенційних ризиків та необхідних функцій безпеки;
- оцінка ефективності стратегій для зниження рівня ризику;
- проведення заходів із забезпечення функціональної безпеки на етапі проєктування системи та визначення її життєвого циклу (ЖЦ);
- тестування відповідності роботи системи встановленим стандартам безпеки через визначення ймовірності та критичності відмов функціональних вузлів системи;проведення аудитів безпеки з метою вивчення відповідності алгоритму виконання ЖЦ системи параметрам безпеки.

Методологія забезпечення функціональної безпеки системи, так само як і методологія забезпечення загальної безпеки, не може бути визначена без розгляду системи загалом та середовища, у межах якого ця система функціонує та з елементами якого вона взаємодіє. Отже, поняття функціональної безпеки на рівні визначення архітектури системи слід віднести до систем моніторингу та систем управління [9, 13]. У зв’язку з тим, що моніторинг включає у себе

збір даних, виявлення критичної відмови, а також елементи контролю та керування. Цей блок також може бути віднесений до системи управління. При забезпеченні безпеки функціонування мобільного застосування враховуються два типи відмов: випадкові і систематичні [14, 15, 16, 17, 18, 19, 20, 21, 22]. Випадкові відмови викликаються виходом із ладу апаратних компонентів, у цьому випадку застосовуються методики резервування, системи діагностики, а також поділ та дублювання функціональних компонентів. Систематичні відмови переважно пов’язують з помилками проектування системи та програмного забезпечення [23]. Зменшення ризику появи систематичних відмов забезпечується через вдосконалення процесів проектування, розробки й тестування програмних і апаратних засобів, визначення ЖЦ системи управління тощо. Слід зауважити, що класичне резервування не дає можливості уникнути систематичних відмов, тому в цьому випадку застосовується різнонаправлене резервування, при якому для резервних каналів використовуються різні засоби програмного й апаратного забезпечення.

Ключовим фактором, що вказує на ефективність експлуатації мобільного додатку, є поняття функціональної придатності як сукупності опису ознак, що визначають призначення, основні, необхідні й достатні функції ПЗ та специфікаціями вимог потенційного користувача. При проектуванні програмного комплексу ознаки функціональної придатності повинні конкретизуватися в специфікаціях як на систему загалом, так і відповідно до окремих складових. До таких ознак можна віднести функціональну повноту вирішення завдань, ступінь покриття функціональних вимог специфікацій та стабільність функціонування при вдосконаленні та оновленні ПЗ, число реалізованих вимог замовника та ін. Крім цього, функціональну придатність відображають спеціалізовані критерії, які тісно пов’язані з конкретними завданнями і самою сферою застосування програмного комплексу. Функціональна придатність значною мірою модифікується у межах ЖЦ ПЗ, і відповідно змінюється конкретний зміст функцій та процедур ПЗ. На різних

етапах ЖЦ функції елементів ПЗ повинні визначатися на відповідність опису у ТЗ та специфікаціях [24], що дозволяє поетапно формалізувати параметри і визначати рівень функціональної придатності продукту. З цієї позиції функціональна придатність визначається якістю взаємозв'язку й узгодженості послідовних формулювань змісту і реалізації основних елементів стандартизованих вимог ПЗ:

- мета ПЗ згідно з яким формується перелік завдань, на основі якого формується докладний опис його призначення і здійснюється аналіз середовища застосування;
- призначення ПЗ деталізовано у вимогах до функцій комплексу програм і його компонентів;
- функції ПЗ, реалізація вимог до яких забезпечується попереднім аналізом властивостей середовища виконання ПЗ;
- вихідна інформація, що отримується внаслідок функціонування ПЗ, та результати, що отримує користувач при експлуатації ПЗ.

Таким чином, рівень функціональної безпеки і якості роботи ПЗ може бути передбачуваним і керованим. Він безпосередньо залежить від ресурсів, що виділяються на його досягнення, а головне, від системи якісних та ефективних технологій, що використовуються на всіх етапах ЖЦ ПЗ.

1.2.2 Показники якості функціональної безпеки

Для аналізу якості функціональної безпеки необхідно визначити узагальнені параметри, що її характеризують відповідно до міжнародних та вітчизняних стандартів, передусім це ISO 61508 [25] та його вітчизняний аналог ДСТУ EN 62061:2014 [26]. Зазвичай для визначення показників якості функціональної безпеки програмних і апаратних засобів виділяють три групи параметрів:

- внутрішні параметри;
- параметри середовища;

- параметри даних.

Для кожної групи можна визначити перелік класів функціональної безпеки програмних засобів та відповідних характеристик згідно зі специфікою системи, що підлягає розгляду. Так, наприклад, для стандартної моделі апаратно-програмного комплексу [27] виділяють такі класи (рис. 1.3):

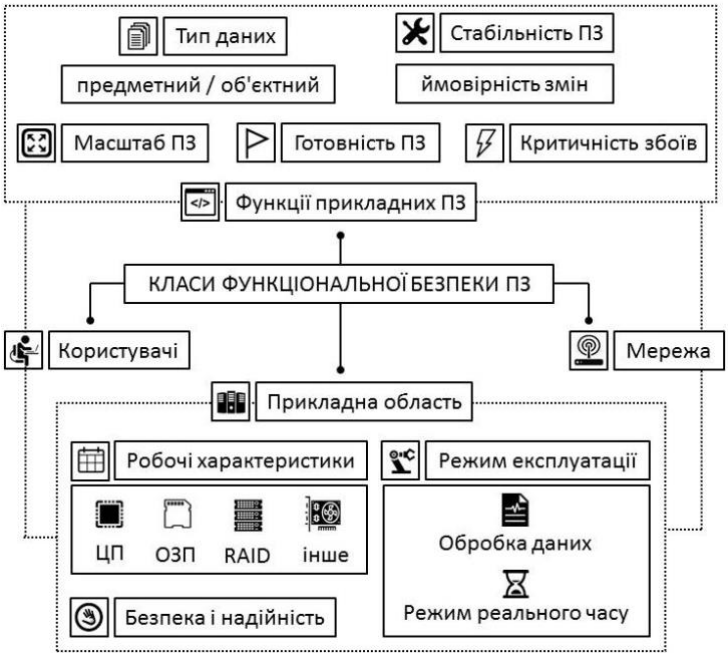


Рисунок 1.3 – Класи функціональної безпеки програмних засобів

- функції прикладних програмних засобів (ПЗ), що включають у себе системи управління об’єктами або процесами;

- масштаб програмного комплексу та ПЗ;

- тип представлення даних, що може бути предметним або об’єктним;

- критичність для загальної системи збоїв у роботі ПЗ;

- стабільність ПЗ, що визначає ймовірність внесення змін;

- готовність програмного продукту відповідно до конкретного переліку запитів загальної системи;

- режим експлуатації, що включає у себе процес обробки даних у режимі реального часу;

- прикладна область системи, що включає у себе обладнання, на основі якого здійснюється керування процесами й об’єктами;

- мінімальні робочі характеристики апаратного комплексу та обчислювальних ресурсів;
- вимоги безпеки і надійності функціонування загальної системи;
- середовище локальної та глобальної мережі;
- рівень користувачів, необхідний для забезпечення роботи системи.

Якісні показники ПЗ можна розділити на базові рівні, при цьому відповідність показника рівню може бути визначена при виборі способу обчислення показника [28, 29]. Під час аналізу здебільшого використовують такі рівні (рис. 1.4):

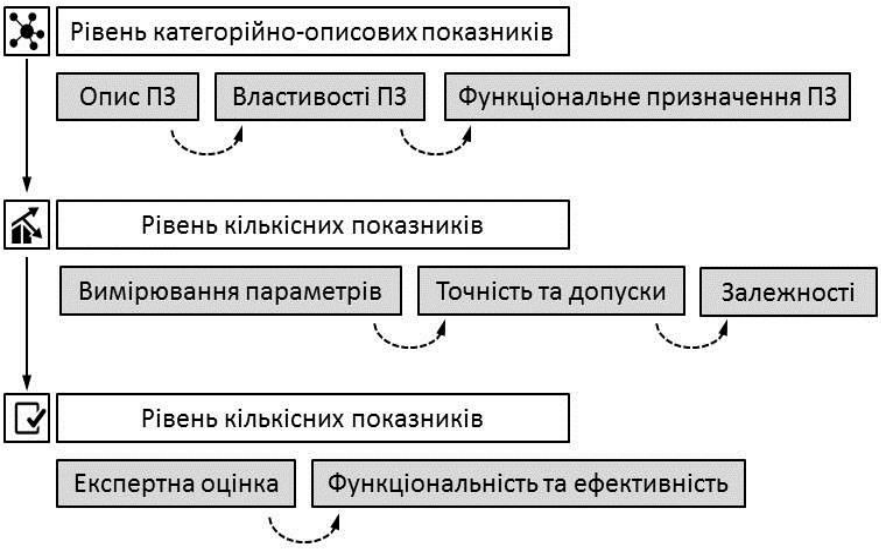


Рисунок 1.4 – Базові рівні оцінки показників програмних засобів

- рівень категорійно-описових показників, на якому визначаються набір властивостей і загальні характеристики досліджуваного ПЗ;
- рівень кількісних показників, на якому можна отримати табличні дані, що відображають залежності, які характеризують роботу ПЗ;
- якісний рівень показників, на якому визначається експертна оцінка показників роботи системи.

До рівня категорійно-описових показників належать властивості програмного забезпечення та опис його даних, що охоплює всі функції досліджуваного ПЗ. Для проведення коректного порівняння цього класу показників необхідно сформувати вибірку ПЗ одного типу. Ключовим

категорійно-описовим показником є показник функціональної придатності, для визначення якої необхідно провести детальний і коректний опис властивостей системи. Встановлення показника функціоЇнальної придатності є першим етапом визначення параметрів функціональної безпеки ПЗ та інших стандартизованих показників якості комплексу [10, 27].

До рівня кількісних показників належать конструктивні характеристики ПЗ, що можуть бути достовірно визначені із зазначеним рівнем точності. Ці параметри найбільшою мірою впливають на функціональну придатність ПЗ, тому їх вибір необхідно проводити на етапі проектування ПЗ. До переліку параметрів відносять функціональну безпеку, коректність роботи, надійність і ефективність експлуатації. Зазначені величини документуються у ТЗ та специфікаціях вимог разом із відповідною методикою їх чисельних вимірювань на етапі випробувань ПЗ.

Якісний рівень показників, у свою чергу, не може бути визначеним лише на основі конструктивних характеристик. Такі показники визначаються відповідно до функціонального призначення ПЗ за погодженням з експертною оцінкою ЖЦ ПЗ і можуть бути виражені у вигляді логічних змінних, бальних або процентних значень.

При розробці проекту для отримання кількісних показників якості ПЗ першочергово слід визначити оптимальні значення та допуски для кожного з параметрів. Надалі ці значення мають бути зафіксовані у специфікаціях вимог до проекту властивостей і ознак кожного параметра якості, що враховують реальні обмеження ресурсів, доступних для їх досягнення в життєвому циклі ПЗ. Таким чином, кінцевою фазою цього етапу є отримання для кожного показника діапазонів допустимих значень, конструктивних характеристик якості або меж кількісних та якісних шкал. Вирішення вказаних завдань спрямоване на забезпечення високої функціональної придатності ПЗ шляхом підвищення безпеки та ефективності роботи системи в умовах обмежених ресурсів на ЖЦ. Тому параметри і вимоги до них слід обирати саме з урахуванням їхнього впливу

на функціональну придатність. Так, наприклад, високі вимоги до параметрів, що здійснюють незначний вплив на основні функціональні характеристики ПЗ і при цьому потребують значних апаратних та програмних ресурсів, доцільно обмежити.

Визначення меж кількісних та якісних шкал функціональних параметрів систем при забезпеченні функціональної безпеки здійснюється на основі таких принципів [10, 11, 30, 31]:

- граничні значення характеристик обмежуються допустимим або раціональним використанням програмно-апаратних ресурсів;
- допустимі витрати ресурсів (включаючи час виконання процедури) для реалізації функціональної безпеки і належної ефективності експлуатації системи повинні забезпечувати функціональну придатність ЖЦ системи на відповідному рівні;
- допустимі мінімальні значення показників функціональної безпеки і ефективності роботи системи якості можуть відповідати значенням, при яких починає знижуватися функціональна придатність застосування системи і ПЗ;
- обмежені значення окремих конструктивних характеристик якості не повинні негативно впливати на значення інших пріоритетних характеристик.

Згідно зі стандартами [25, 26] кожен із параметрів функціональної безпеки має бути визначено з точністю, зазначеною в технічному завданні та специфікаціях. Проведені вимірювання при цьому мають бути об’єктивними, відтворюваними й відповідним чином документованими, норми допустимих похибок вимірювання визначаються заздалегідь.

1.3 Сучасний стан функціональної безпеки мобільних пристроїв

Аналіз сучасних публікацій свідчить, що найбільший рівень загрози пов’язують не з апаратними ресурсами та операційною системою (ОС)

мобільного пристрою [32, 33, 34, 35, 36], а саме з мобільними додатками, що містять у собі такі класи вразливостей, як (рис. 1.5):

- некоректність визначення привілеїв користувачів та персоналу;
- втрати та пошкодження інформаційних блоків;
- низький рівень захисту функціональних компонентів;
- вразливості міжкомпонентних комунікацій.

Рисунок 1.5 – Стратегія захисту від типових уразливостей мобільних додатків

Дані вразливості значною мірою виявляються за допомогою автоматичного статичного аналізу [37], що гарантує можливість розробити масштабоване програмне забезпечення з достатнім рівнем захисту. Одним із прикладів статичного аналізу є реалізація з використанням LSI (Latent Semantic Index) яка базується на сингулярній векторній декомпозиції SVD (від англ. Singular Vector Decomposition) [38 - 41] дозволів файлу маніфесту “AndroidManifext.xml”, що дозволяє не встановлюючи програмний додаток проаналізувати карту дозволів та математично порівняти її з шкідливим шаблоном дозволів. LSI та реалізації які базуються на даному алгоритмі призначені для полегшення створення

природної мови та були успішно використовували у низці проектів, як академічних, так і комерційних (в тому числі в аналізу зловмисного ПЗ) [42]. Недоліком статичного аналізу є високий рівень помилкових спрацьовувань. Це відповідає помилці другого роду, яка має бути відслідкована додатковим блоком або користувачем у ручному режимі, що суттєво ускладнює систему захисту і збільшує рівень взаємодії на рівні P2M [43] (машина-людина), чого бажано уникати. Також стандартним є підхід, при якому елемент програми, що є потенційно небезпечним, відсилається розробнику програмного забезпечення, що надалі корегує програмний код через оновлення. Але при цьому підході виявлені уразливості можуть не розглядатися протягом довгого часу, залишаючи можливість для їх застосування зловмисником.

Треба зазначити, що відмінності між очікуваними й отриманими результатами функціонування програм можуть з’являтися також унаслідок помилок у програмному коді й масивах даних, а також помилок проектування системи. На практиці в процесі розробки ПЗ вихідні вимоги уточняються і деталізуються за погодженням між замовником і розробником. Основою для таких уточнень є знання та неформалізовані уявлення фахівців, результати проміжних етапів ЖЦ системи управління. Однак унаслідок відсутності еталонних формалізованих даних, помилки у вихідних даних буває доволі важко виявити. Проблеми функціонування ПЗ, що не пов’язані з кібератаками (КА) та зловмисним втручанням у роботу апаратного комплексу, проявляються як випадкові похибки і розрізняються за своїм змістом і кінцевим результатом. Повне усунення недоліків, що впливають на безпечність та якість функціонування складних ПЗ, є принципово неможливим. Рішення полягає у виявленні чинників, від яких залежать зазначені, у створенні методів та засобів зменшення їхнього впливу на функціональну придатність ПЗ, а також в ефективному розподілі обмежених ресурсів для забезпечення необхідної якості функціонування комплексу програм. Комплексне застосування цих методів і засобів у процесі створення, розвитку та застосування ПЗ дозволяє усунути негативні чинники або значно послабити їхній вплив [44].

1.4 Складові системи функціональної безпеки

Методи по забезпеченню безпеки систем керування повинні забезпечувати зниження ризиків відповідно до заданих рівнів. Складові системи функціональної безпеки при цьому можна поділити на організаційні та технічні методи.

Узагальнена схема експлуатації апаратного чи програмного комплексу з погляду забезпечення функціональної безпеки складається з блоків контролера, пов'язаного з виконавчою ланкою і сенсором, що взаємодіють із середовищем виконання; при цьому доступ до контролера можна отримати через інтерфейс користувача, а результатом роботи контролеру є дані, отримані в результаті моніторингу (рис. 1.6).

Така схема є типовою для вбудованих систем промислової автоматизації, побутових пристроїв та комунікаційних мереж. На її основі може бути побудована більш складна й розгалужена архітектура, що включає об'єднання в мережі сенсорів, контролері, виконавчих механізмів, а також інформаційних сховищ.

Комплексний підхід включає також розділення системи на рівні виконання, що є особливо характерним для хмарних сервісів, центрів обробки даних (ЦОД) та Інтернету речей (IoT: Internet of Things).

Рисунок 1.6 – Узагальнена схема роботи комплексу з погляду забезпечення функціональної безпеки

При такому підході виконання задачі поділяється на рівень застосувань (AL: Application Layer), сервісний рівень (SL: Service Layer), мережевий рівень (NL: Network Layer) та апаратний рівень (DL: Device Layer), на якому реалізується керуюча система. З погляду інформаційної безпеки критичними є інтерфейси DL-NL і DL-AL [45], що надають доступ саме до апаратного рівня (рис. 1.7).

Рисунок 1.7 –Багаторівнева схема роботи комплексу

Група організаційних методів включає у себе управління проектами, організацію ЖЦ систем інформаційної та функціональної безпеки, визначення стандартів побудови програмного коду, використання трансляторів, компіляторів та бібліотек, контроль апаратних компонентів, а також розробку відповідних специфікацій. З іншого боку, група технічних методів складається з комплексу процедур резервування, забезпечення різноманітності процесів захисту від внутрішніх і зовнішніх загроз, розробку інтерфейсу і засобів самодіагностики системи [46].

1.4.1 Безпека сучасних операційних систем

Нині найбільш детальний аналіз забезпечення функціональної безпеки операційних систем (ОС), їх оточення та застосувань було проведено для ОС Microsoft Windows, що характеризуються значними вразливостями [47]. Детальний розгляд цього сімейства ОС в подальшому дозволить порівняти його

з UNIX-подібною системою Android та вказати переваги останньої щодо забезпечення безпеки експлуатації.

Ключовим етапом розробки стратегії із забезпечення функціональної безпеки ОС є впровадження політики моніторингу та ведення журналу безпеки (SL&M: Security Logging and Monitoring) з метою забезпечення конфіденційності, цілісності та доступності інформації, що використовується системою. У межах цієї політики мають бути визначені мінімальні вимоги до ведення журналів безпеки та моніторингу систем компанії, а також стандартизовані процедури моніторингу та реєстрації даних, що стосуються потенційно небезпечних процесів ОС.

Журнали безпеки включають у себе інформацію про системні події, їх функціонал та зв’язок з іншими подіями в середовищі виконання системних алгоритмів, а також вказують на системні порушення та некоректне використання ресурсів. При належному налаштуванні та правильному використанні журнали безпеки можуть стати важливим засобом для отримання форм звітності кожної події, що відбувається в межах ОС. Вони дають відповіді на такі критичні питання, що стосуються функціональної та інформаційної безпеки, зокрема:

- участь користувачів та програмних блоків у досліджуваному процесі (питання «хто?»);
- час події (питання «коли?»);
- залучені елементи ОС (питання «де?»);
- особливості процесу (питання «як?»).

Незалежно від типу ОС розробник системи SL&M має забезпечити гарантії того, що повний набір інформації, пов’язаний із функціонуванням системи, зокрема дії користувачів, належним чином реєструються і підлягають оперативному перегляду.

Ведення журналу безпеки включає у себе реєстрацію таких подій, що виникають при експлуатації ОС:

- усі спроби входу в систему, включаючи успішні та невдалі;

- факти додавання, видалення та модифікації облікових записів;
- перелік користувачів, які вносять зміни в облікові записи;
- внесення змін у журнал безпеки в ручному режимі;
- внесення змін у налаштування системи;
- доступ до важливих даних з обмеженням прав доступу;
- запуск сеансу роботи, вимкнення та перезавантаження ОС;
- системні помилки;
- створення нових сервісів;
- завершення роботи з додатком та його перезавантаження;
- помилки, що виникають при роботі з додатками;
- створення та припинення процесів;
- створення критичних процесів;
- внесення змін у системний реєстр;
- модифікації політики безпеки;
- використання прав адміністратора;
- доступ до системних файлів;
- системний збій ОС.

Як можна побачити, вказані події, що підлягають реєстрації, можна поділити на чотири основні групи: взаємодія «користувач-ОС», взаємодія «користувач-програмні додатки», взаємодія «ОС-програмні додатки» і системні помилки (рис. 1.8).

Відповідно до політики ведення журналу безпеки можна визначити ключові аспекти політики моніторингу, зокрема: перевірка сесії при запуску системи, що включає забезпечення реєстрації всіх подій, пов’язаних із сеансами облікового запису та доступу до відповідних процесів; забезпечення функціонування журналу безпеки після перезавантаження або тимчасової відмови роботи системи; забезпечення альтернативного інструментарію ведення журналу безпеки в разі зупинки роботи основного блоку; передача

даних журналу безпеки до координаційного центру з розмежуванням привілеїв та забезпеченням належного рівню доступу [48].

Рисунок 1.8 – Структура процесів, що реєструються в журналі безпеки

Отже, журнал безпеки має бути постійно активним і захищеним від несанкціонованого доступу, модифікації даних та внесення змін у його структуру.

1.4.2 Безпека оточення операційних систем

Оточення операційних систем є за замовчанням змінним, тому в цьому випадку важливо визначити привілеї доступу до нього як з боку адміністратора, так і з боку користувачів. Слід зазначити, що ані користувачі, ані адміністратор не може мати змогу відключити алгоритм заповнення журналу безпеки або внесення змін у його дані. Конфігурація цього блоку може бути змінена адміністратором, але лише за умови попереднього дозволу від внутрішньої служби безпеки інформації. При цьому нові дані заносяться автоматично через фіксацію перевірених подій, що підлягають відстеженню.

На апаратному рівні масив даних журналів безпеки повинен зберігатись таким чином, щоб уникнути стороннього доступу та порушення структури

журналу, інакше цей елемент не може бути включеним у систему безпеки. Доступ до перегляду журналів безпеки має обмежуватися правами адміністратора, відповідального за аналіз даних журналу та забезпечення цілісності його структури; ця вимога стосується як локальних журналів, так і централізованого інформаційного сховища. Апаратно-програмний комплекс інформаційного сховища має характеризуватися достатньою інформаційною ємністю, відповідними обчислювальними ресурсами та програмними механізмами автоматичної обробки даних журналів безпеки, що включає в себе аналіз потенційних загроз [49].

Таким чином, централізована система збору та аналізу даних журналів безпеки повинна відповідати таким вимогам.

- забезпечення надійності та ефективності функціонування інфраструктури керування журналами безпеки;
- впровадження контрольних засобів моніторингу з метою забезпечення постійного доступу для стратегічно важливої з погляду функціональної безпеки інформації та формування сповіщення для групи експлуатаційної безпеки системи;
- забезпечення надійного зберігання та передачі даних журналів безпеки через використання захищених протоколів та технологій шифрування.

Отже, сервери, на яких зберігаються журнали безпеки, мають розглядатися як критичні активи та бути захищеними згідно зі стандартами конфіденційної інформації. Оточення ОС, що не має вбудованих ланок об’єднання з централізованою системою журналів, мають бути налаштовані та доповнені елементами контролю і доступу. На рівні системної аналітики при цьому має відбуватися регулярний перегляд та аналіз журналів безпеки відповідно до вимог, що забезпечує релевантність зібраної інформації. При цьому блоки даних, що утримують надлишкову чи помилкову інформацію, мають своєчасно видалятися для підтримки належного рівня продуктивності системи та зменшення кількості помилкових спрацьовувань.

1.4.3 Безпека програмних додатків

Ключовим аспектом безпеки програмних додатків є коректність визначення мети ЖЦ, що відповідає системній ефективності ПЗ та потребує адекватної експертної оцінки. Призначення програмних додатків на цьому етапі проектування формалізується згідно з вимогами до функцій компонентів і всього програмного комплексу. Повнота виконання повного набору функцій, що відповідають призначенню програмного додатку є характеристикою, яка визначає потенційну можливість реалізації його функціональної придатності загалом. Відстеження деталізації поставленого завдання і охоплення вимог, починаючи від системних завдань, а також конкретизація і корегування завдань, повинні забезпечувати належний рівень функціональної придатності комплексу.

В ОС Android існує набір захищеного API, за допомогою якого програми можуть мати доступ до даних користувачів (інформаційна безпека). На рис. 1.9 представлені типи сховищ даних, які вимагають прямої згоди користувача.

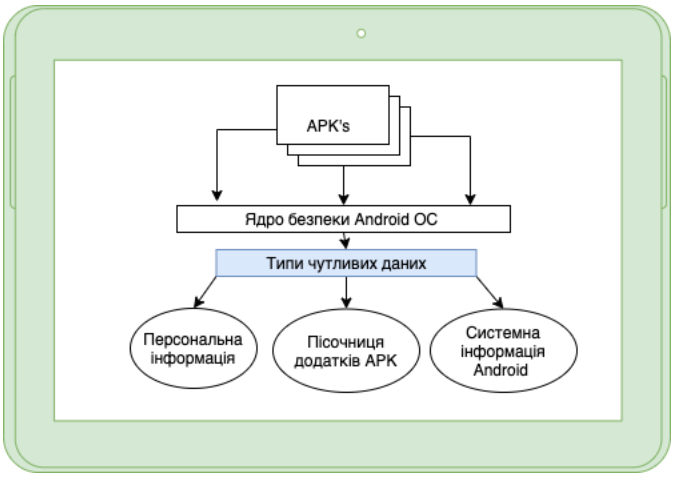


Рисунок 1.9 – Типи сховищ даних, які вимагають прямої згоди користувача

Функції додатків реалізуються в апаратному, операційному та зовнішньому програмному середовищі системи, характеристики якої також істотно впливають на функціональну придатність. Для виконання необхідних функцій комплексу програм необхідна адекватна вихідна інформація від об’єктів середовища виконання алгоритму, зміст якої має повністю забезпечувати реалізацію

декларованих функцій. Повнота формалізації структури і якості вхідної інформації для виконання необхідних функцій є однією з важливих складових при визначенні функціональної придатності програмного додатку. Змістовна частина цієї інформації визначається конкретними завданнями системи, їхніми основними технічними показниками функціональності.

Відстеження і оцінювання адекватності та повноти складу вихідної інформації, а також її відповідність призначенню додатку є останнім етапом визначення параметрів функціональної придатності системи, незалежно від типу програмного додатку.

При проектуванні у складі поняття функціональної придатності можна виділити дві групи базових параметрів, що визначають функціональні та структурні вимоги, а також особливості експлуатації додатків. При формалізації і виборі функціональних вимог необхідно максимально чітко сформулювати в рамках ТЗ такі пункти [50]:

- специфікацію ЖЦ системи, додатків і їхні складові;
- системну ефективність та технічні показники додатків як складові загальної системи;
- призначення застосунків;
- середовище виконання алгоритмів;
- умови ефективної і безпечної експлуатації додатків;
- функціональні завдання додатків і їхніх складових;
- необхідний і достатній рівень забезпечення функціональної та інформаційної безпеки додатків;
- характеристики якості і регламент вирішення функціональних завдань;
- відповідність додатків та його складових стандартам.

Відповідно до призначення додатків забезпечення функціональної безпеки може стати одним з основних завдань або навіть повністю визначати функціональну придатність програмного комплексу. На жаль, цю характеристику у багатьох випадках важко звести до кількісних показників і

часто її доводиться оцінювати відповідно до типових процедур використання додатків та за величиною ресурсоємності, що очевидно суттєво впливають на функціональну придатність системи. При цьому коректність роботи додатку визначається через отримання прийнятних за рівнем якості результатів його експлуатації. Вибір вимог при проектуванні здійснюється через визначені та взаємопов’язані характеристики функцій комплексу та окремих модулів програм, а також правила їхньої структурної побудови й організація інтерфейсів взаємодії [51]. Таким чином, поняття коректності роботи додатку охоплює забезпечення очікуваних результатів із необхідним ступенем точності відповідно до вимог ТЗ та специфікацій. У процесі проектування та розробки модулів і груп програм застосовуються локальні структурні параметри коректності, кожен із яких може характеризуватися окремим інструментарієм визначення якості й отримується на основі аналізу коректності функціонування (через детерміновані чи стохастичні проміжки часу або безпосередньо в режимі реального часу). Вимоги до характеристики коректності можуть подаватися у вигляді опису двох основних вимог, яким повинні відповідати всі програмні компоненти та додатки загалом: відстежуваність і верифікація щодо реалізації вимог при послідовній деталізації описів програмних компонентів та вибір ступеня охоплення тестовими алгоритмами функцій програмних компонентів, що буде достатнім для функціонування ПЗ з необхідною якістю і точністю за умов обмеження ресурсів. Для визначення цієї величини при розробці інформаційної тхнології необхідна організація систематичної реєстрації, визначення змісту функцій і контроль відсотка процесів, що не пройшли тестування. Отже, за показник коректності можна поставити відносне число протестованих функцій щодо відношенню загального числа виконуваних [52]. Такий показник відображає трудомісткість і тривалість тестування, що безпосередньо впливає на функціональну придатність програмного додатку.

1.5 Аналіз системи функціональної безпеки ОС Android

Нині ОС Android, розроблена на основі ядра Linux та віртуальної машини Java, є лідером на ринку смартфонів і найпопулярнішою системою для інших мобільних пристроїв, таких як планшети, електронні книги, наручні годинники, фітнес-браслети та ін. Бурхливий розвиток мобільних додатків, розроблених для ОС Android, вказав на вразливі місця цієї системи та викликав необхідність вдосконалення інструментів, що здатні забезпечити належний рівень функціональної безпеки на основі контекстного і систематичного підходів. Аналіз сучасних публікацій [15, 19, 21, 34, 32, 37, 53, 54] вказує на експоненціальне зростання кіберзагроз для мобільних додатків системи Android, при цьому автоматичні способи виявлення та класифікації зловмисних алгоритмів не дають можливості уникнути цього типу загроз [55, 56]. Слід зазначити, що підходи на основі машинного навчання штучних нейромережевих алгоритмів, що дозволяють визначити рівень загрози на основі базових параметрів програмного коду [57], наприклад, при аналізі викликів API функцій, також не забезпечують належного рівня ефективності відстеження кібератаки, через чутливість до синтаксису коду. Компоненти вбудованої системи функціональної безпеки ОС Android представлені на рис. 1.10 [58, 59].

Контроль доступу за замовчуванням
Аутентифікація
Шифрування даних за замовчуванням
Шифрування на рівні файлової системи
Ізоляція додатків
Оновлення ОС
Оновлення безпеки
Система привілеїв додатків
Запит привілеїв у реальному часі

Рисунок 1.10 – Компоненти вбудованої система функціональної безпеки ОС Android

Система Android надає користувачам найбільш широкі можливості для організації десктопу мобільного пристрою через вибір і встановлення відповідного програмного забезпечення і додатків. Тому дуже важливо, щоб користувачі усвідомлювали особливості поведінки мобільних додатків для прийняття відповідних рішень. З цією метою користувачам для кожного мобільного додатку надається інформація щодо дозволів, яких вимагає програмне забезпечення, та анотація, надана розробником. Такий рівень інформованості не може вважатися ефективним при розробці стратегії інформаційного та функціонального захисту, тому що ця інформація не є повною та не забезпечує користувача усвідомленням щодо рівня потенційних загроз, які може спричинити мобільний додаток. Через відсутність контекстної підказки, користувач не може сприймати особливості сервісів за допомогою простого переліку дозволів. Статистичний аналіз текстових описів програмного забезпечення свідчить про те, що формат опису часто значною мірою відхиляється від стандарту, тому він також не може вважатися надійним засобом інформування користувача [60].

Окремим важливим завданням є забезпечення захисту сучасних систем типу «Інтернет речей» (IoT: Internet of Things). IoT являє собою глобальну мережу комп’ютерів, датчиків виконавчих пристроїв [61, 62, 63, 64], що зв’язуються між собою через інтернет-протокол (IP: Internet Protocol). Очевидно, що для мережі IoT недосконала система захисту може надати збитків не лише на рівні викрадення важливої інформації, а й на безпосередньо фізичному рівні [65]. У такому випадку ми працюємо із системою з великою кількістю складових, кожну з яких можна вважати окремою апаратно-програмною платформою. Усі елементи мають бути з’єднанні через безпечний протокол, що зумовлює встановлення єдиного стандарту оцінки журналів реєстрації, захисту і шифрування даних, впровадження безпечних протоколів комунікації, контролю за виконанням потенційно небезпечних ПЗ, а також аналізу мобільних додатків і веб сервісів [66-68].

1.6 Критичні місця застосувань ОС Android

Програмні додатки для ОС Android зазвичай будуються на основі мови програмування Java за підтримки «Android Software Development Kit» (SDK) [69]. При цьому програмний код будується в межах класів Java і надалі компілюється DEX-файл, що реалізується на віртуальній машині Java Dalvik. DEX-файл та додаткові файли (макет, зображення та ін.) збираються у APK-пакет (APK: Android Package – формат архівних файлів-додатків), що розміщується на ринку програмних додатків, таких як «Google Play Market» з описом, представленим переважно в текстовому форматі. Після того як APK-пакет завантажується та встановлюється на мобільний пристрій, DEX-файл виконується вже на віртуальній машині (DVM: Dalvik Virtual Machine) [70] цього пристрою. Фактично, DEX-файл виступає в ролі плагіна до базового коду, тому що основна частина виконання програмного коду відбувається в рамках базових процедур, вбудованих у ОС Android. Файл DEX взаємодіє з платформою Android за допомогою прикладного програмного інтерфейсу додатків (API: Application Programming Interface). Інфраструктура системи Android також надає спеціальний набір API, що можуть розглядатися як функціональні елементи мобільних додатків, зокрема «Activity», «Service», «Broadcast Receiver» і «Content Provider». Клас «Activity» відповідає за роботу з графічним інтерфейсом GUI і безпосередньо взаємодіє з користувачем. «Service» виконує у фоновому режимі внутрішні операції системи, приймаючи запити на обслуговування від інших додатків або їхніх компонентів. «Broadcast Receiver» є компонентом, який обробляє загальносистемні повідомлення, а «Content Provider» інкапсулює вміст даних і передає їх через уніфікований інтерфейс. Взаємодія компонентів здійснюється через блок «Intents», хоча розробник програмного забезпечення при цьому може створювати власну структуру дозволів, щоб захистити компоненти мобільного додатку від зовнішнього впливу, але, очевидно, що такий механізм

використання спеціальних дозволів може бути використаний і зловмисником [71].

Очевидно, що API є найбільш чутливим компонентом ОС Android, що захищається через систему дозволів на доступ (Android Permission), які визначають привілеї користувачів і розробників. Для запуску критичних функцій програмного додатка для розробника необхідно отримати доступ через файл маніфесту застосувань AndroidManifest.xml з подальшим погодженням користувача, який надає застосуванню відповідні привілеї. Після того як дозволи надані й додаток встановлено на мобільний пристрій скасувати їх під час виконання додатка може бути достатньо складно й система не забезпечує обмеження додатку від зловживання своїми привілеями.

Як було вказано, способи автоматичного виявлення зловмисного програмного коду системою Android поділяються на дві загальні категорії:

- виявлення зловмисного коду на основі підписів;
- виявлення зловмисного коду на основі машинного навчання.

Підходи, що базуються на основі підпису, ґрунтуються на пошуку конкретних шаблонів у байт-кодах та викликах API функцій, що легко обійти через трансформацію програмного коду на рівні байт-коду, машинно-незалежного коду низького рівня, що генерується транслятором і виконуваний інтерпретатором [72]. Підходи на основі машинного навчання вилучають шаблони поведінки програми (зазвичай, запити на дозвіл та критичні виклики API функцій) та застосовують стандартні алгоритми машинного навчання для виконання класифікації зловмисного коду. Однак оскільки вилучені функції пов’язані із синтаксисом програми, такий тип детекторів також не є надійним інструментом захисту від кіберзагроз. На сьогодні ефективність програмних засобів, розроблених на основі зазначених підходів, більшою мірою залежить від якості розробленої схеми виявлення, характерної для певної поведінки програмного додатку, тому їхню спеціалізацію можна вважати досить вузькою.

Іншим завданням є класифікації зловмисного програмного коду, що може виконуватися в межах ОС Android. Як було вказано, базовим підходом є використання методу машинного навчання. Найбільш поширені алгоритми класифікації при цьому базуються на імовірнісних генеративних моделях оцінки ризиків, хешуванні функцій по послідовності коду операцій для виявлення повторного використання шкідливого коду та гібридних підходах. Високу ефективність показує застосування широкого статичного аналізу для вилучення наборів функцій із файлів маніфесту та програм байт-коду, при якому всі набори функцій вбудовуються у спільний векторний простір. Таким чином, функції, що сприяють виявленню шкідливих програм, можна проаналізувати на основі геометричних методів. Однак ці підходи обмежені аналізом зовнішніх особливостей коду та не в змозі здійснювати точну та повну інтерпретацію поведінки додатків.

Типовим методом боротьби з вразливостями ОС Android є система автоматичного створення патчів [73, 74]. За умов того, що додатків Android не розроблено інструментарію автоматичного та ефективного виправлення вразливостей, було проведено дослідження для побудови алгоритмів автоматичної генерації патчів для стандартних програмних додатків, що виконуються згідно з парадигмою клієнт-сервер. Такі алгоритми аналізують програмний код, визначають вразливості та їх причину, що призвела до неналежного використання додатку програмних ресурсів ОС Android та апаратних ресурсів мобільного пристрою, після чого створює патч вихідного коду, що може бути використано для тимчасового виправлення вразливості без втручання у процес оператора. Згідно з альтернативними підходами можуть бути використані класична теорія типів та структури аналізу даних, перетворення вихідного коду для застосування автоматично створених патчів до уразливих сегментів на основі опису широкого класу припущень.

Компанія Google, як основний розробник ОС Android, вибрала політику окремих патчів для системи безпеки та операційної системи (рис. 1.11). Такий

підхід дозволяє паралельно розробляти нові функції ОС та швидко реагувати на нові загрози з боку шкідливого ПЗ.

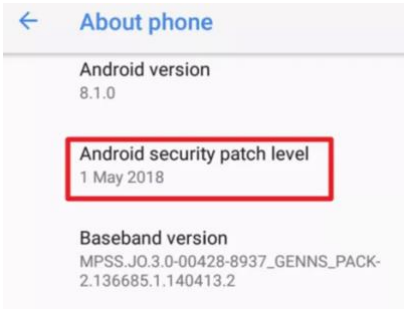


Рисунок 1.11 – Окремий патч системи безпеки ОС Android

У межах цього процесу перезаписується DEX-файл для встановлення всіх викликів API та забезпечення виконання відповідних правил безпеки. Також може бути використано тимчасове блокування додатків Android з метою контролю потенційного порушення безпеки та конфіденційності. Крім того, останнім часом було розроблено алгоритми, що дозволяють розміщувати підказки в програмі байт-кодів і таким чином захищати доступ до ресурсів. З іншого боку, для вирішення проблем, пов’язаних із суттєвими вразливостями програмних додатків, необхідно розробити нову техніку, яка дозволила б ефективно контролювати конфіденційність інформаційного потоку, що здійснюється додатками ОС.

Характерною особливістю програмних додатків Android є стрімке зростання програмного коду, що відбувається внаслідок оновлень та патчів. Водночас через обмежені ресурси на мобільних пристроях існує значне обмеження розміру додатків, і таким чином, процес перезапису байт-коду потребує фази оптимізації. Щоб знайти шаблони помилок і надмірного коду у вихідному коді Java, було розроблено оптимізований динамічний інструментарій на базі виконання між процедурного аналізу вказівника (static pointer alias analysis [75]).

Іншою проблемою є втрата конфіденційних даних, що здійснюється через приховані канали системи за допомогою універсальних функцій API для Android. З метою вирішення даної проблеми були розроблені статичні та динамічні

алгоритми для виявлення та аналізу потенційних інформаційних втрат [76]. Базова система була побудована на основі алгоритмів, що використовуються для забезпечення конфіденційності при роботі з програмними додатками та операційною системою. Відповідно, було розроблено інструментарій статичного та динамічного аналізу для виявлення втрат конфіденційних даних. При цьому найбільше поширення отримали ресурсоємні інструменти динамічного аналізу, що модифікує DVM та обробляє байт-код DEX-файлу для виконання динамічного аналізу обфускації [77], зокрема алгоритм TaintDroid [78]. Статичний аналіз додатків ОС Android, з іншого боку, включає процедуру перетворення коду Java та впровадження інструмент його аналізу для виявлення прихованих потоків даних. Недоліки зазначених методів пов'язані з фундаментальною причиною помилкових спрацьовувань, оскільки в межах зазначених підходів легітимне використання конфіденційних даних не відрізняється від процесів, пов'язаних з інформаційними втратами, що суттєво знижує ефективність побудованої на їх основі системи захисту. Контроль передачі даних чи інформаційного потоку (IFC: Information Flow Control) у системах Android максимально активно використовується для вирішення проблеми втрати конфіденційних даних. При розробці відповідного інструментарію аналізується структура інформаційного потоку для віртуальної машини (ВМ) Java, яка включає у себе статичний аналіз для захоплення неявних потоків. У різних підходах також пропонується розширення мови програмування Java, додавання статично перевірених анотацій, розробка системи керування виконанням компонентного рівня для додатків, виділення захищених доменів у межах одного процесу та ін.

У галузі IoT нині існує ОС «Android Things 1.0», що отримала префікс стабільної, але потребує подальшого дослідження ефективності забезпечення безпеки виконання додатків. У межах цієї роботи пропонується провести аналіз та розробити систему захисту IoT відповідно до стандартних підходів, характерних для ОС Android, використовуючи моделі прав доступу, привілеїв, роботи додатків та викликів API функцій.

Проведений аналіз на основі [14, 15, 19, 21, 24, 32, 34, 50, 54, 79] дозволяє визначити основні типи загроз, що можуть бути пов’язані з використанням програмних додатків ОС Android (рис. 1.12):

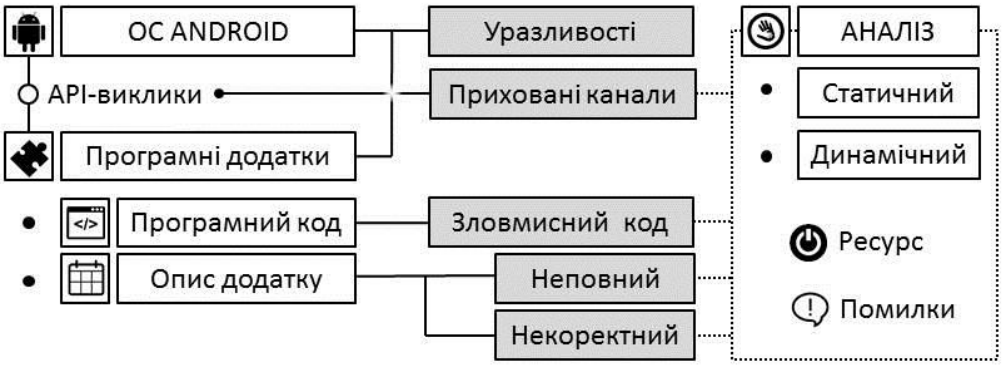


Рисунок 1.12 – Основні типи кіберзагроз програмних додатків ОС Android

- наявність зловмисного програмного коду (malware) у середовищі виконання програмного додатку або в ньому самому;

- наявність вразливостей у програмних додатків та ОС;

- перехоплення зловмисниками конфіденційних даних (privacy leakage);

- некоректний опис програмного додатку.

ОС Android як UNIX-подібна система від початку характеризувалася високим рівнем захисту, особливо в порівнянні з платформами сімейства Windows, але, як показують статистичні дослідження, кількість кібератак на мобільні додатки, що працюють на базі даної ОС, з кожним роком зростає експоненційно а шкідливі додатки випускаються кожні 10 секунд [80]. При цьому наявні алгоритми виявлення та класифікації зловмисного програмного коду показують низьку ефективність та надмірно високу ресурсоємність щодо більшості мобільних пристроїв.

Відкрита система, побудована на базі Linux, дозволяє зловмисникам використати наявні вразливості та влаштувати за допомогою розробленого програмного продукту такі неочевидні для захисних систем протиправні дії, як ескалація привілеїв додатку (privilege escalation), організація прихованих каналів передачі даних, повторне делегування дозволів (permission re-

delegation), викрадення блоків програмного коду та даних та використання вразливостей міжкомпонентних комунікацій.

На сьогодні зазначені вразливості частково компенсуються інструментарієм на основі алгоритмів автоматичного статичного аналізу, що показують достатню гнучкість при масштабуванні системи та задовільний рівень моніторингу програмного коду. Проте статичний аналіз часто призводить до помилкових спрацьовувань, тому має бути поєднаним з блоком виділення помилково позитивних результатів.

Для відстеження прихованих каналів, по яких передаються конфіденційні дані, здебільшого поєднують статичний і динамічний аналіз [81] інформаційного потоку. При цьому слід зауважити, що наявні алгоритми характеризуються надмірно високою ресурсоємністю та не здатні охопити всі процеси, що відбуваються в рамках експлуатації системи та її додатків. Крім того, наявний підхід дозволяє лише виявити наявність приватної передачі даних, але не здатний здійснити аналіз інструментарію, за допомогою якого здійснюється ця передача, що, часто призводить до помилкових спрацьовувань системи захисту.

1.7 Постановка задач дослідження

Дані статистики та проведений аналіз вказують на необхідність побудови цілісної методології із забезпечення функціональної безпеки мобільних додатків. Практика показує ефективність використання окремих алгоритмів на основі статичного та динамічного аналізу, а також гібридного підходу. Однак нині немає універсального підходу, що здатен забезпечити належний рівень точності визначення зловмисного коду при мінімальному об’ємі апаратних ресурсів, що часто є обмеженими у випадку мобільної платформи.

Забезпечення функціональної безпеки є кінцевим етапом проектування програмного комплексу, що базується на аналізі функціональних властивостей компонентів додатку та системи, у межах якої він виконується. Функціональні

стандарти безпеки лежать в основі моніторингу електронних та програмних елементів програмного комплексу, що потребує виконання кожної окремої функції безпеки на відповідному рівні.

Отже, **заданням дослідження** є побудова моделі комплексної системи забезпечення функціонального захисту додатків ОС Android, що використовуються в межах даних ОС, та важливої інформації, що утримується ресурсами ОС від зовнішніх кіберзагроз [82]. З цією метою в рамках роботи було запропоновано:

- розробити модель прав доступу при взаємодії ОС Android із програмними додатками;
- розробити методи динамічного виявлення та блокування потенційно небезпечних додатків;
- розробити програмні засоби реалізації запропонованих методів забезпечення функціональної безпеки;
- оцінити ефективність розроблених моделей, методів та інформаційної технології шляхом проведення експериментів із реальними пристроями та додатками.

1.8 Висновки по розділу 1

1. Проведено аналіз поняття функціональної безпеки та існуючих систем забезпечення функціональної безпеки мобільних пристроїв як комплексу систем моніторингу та управління. Побудовано узагальнену схему роботи програмного комплексу з погляду забезпечення функціональної безпеки та уточнено особливості його застосування.

2. Показано, що аналіз ефективності функціональної безпеки, має бути проведено на рівні категорійно-описових показників, де визначаються набір властивостей і загальні характеристики досліджуваного додатку. Зазначено роль журналів безпеки як важливого засобу для отримання форм звітності подій, що відбувається в межах операційної системи.

3. Виділено основні типи загроз, пов’язаних із використанням програмних додатків, таких як наявність зловмисного програмного коду, наявність вразливостей у програмних застосуваннях, перехоплення зловмисниками конфіденційних даних, некоректний опис програмного додатку.

4. Визначено поняття функціональної безпеки по відношенню до програмних додатків ОС Android. Показано, що оцінка стану функціональної безпеки має проводитись на рівні категорій показників через визначення набору властивостей програмного коду додатку, які характеризують процес його функціонування, а також ідентифікації ймовірності шкідливості додатку.

Результати досліджень, приведених в розділі, опубліковані в роботах [44, 58, 59, 65, 82].

РОЗДІЛ 2
МОДЕЛІ ТА МЕТОДИ ДИНАМІЧНОГО ВИЯВЛЕННЯ ПОТЕНЦІЙНО НЕБЕЗПЕЧНИХ ДОДАТКІВ

2.1 Аналіз карти дозволів та категоризація викликів АПІ функцій

Попри те, що Android використовує ядро Linux (з деякими відмінностями, в тому числі, в ліцензуванні) – система дозволів значно відрізняється від звичайного дистрибутиву “GNU Linux”. Наприклад, під час запуску звичайного додатка GIMP в ОС Linux користувач не надає дозволів для доступу до ресурсів операційної системи основним чином тому, що виконання додатка проходить на тому ж рівні, де виконуються і сама ОС. Коли додаткам необхідні підвищенні привілеї (root), для виконання користувачу необхідно буде ввести пароль, але звичайним користувацьким програмам це здебільшого не потрібно. Справедливо зазначити, що системи дозволів Android та будь-якого дистрибутиву Linux має певні точки перетину, так виконання External Storage або Network дозволено, якщо користувач або додаток перебуває у відповідній групі – RBAC (Role Based Access Control) [83, 84], якій було дозволено використання цього ресурсу ОС, у той час як в Android ОС переважна більшість дозволів реалізована на рівні самої операційної системи. Система дозволів Android OS працює інакше, виконання додатку проходить у зарезервованому ОС для цього місці під назвою пісочниця (sandbox) та не має прав доступу за замовченням поза цього середовища виконання. Саме тому програмні додатки повинні кожного разу запитувати дозвіл на виконання будь-яких викликів функцій доступу до ресурсів операційної системи. Такий підхід додає опцію користувачу, який уже надав усі дозволи переглянути список і якщо вони очевидно не співпадають с передбачуваним сценарієм виконання функціоналу – заборонити використовувати цей дозвіл або групу дозволів (наприклад, додаток FM-Radio використовує дозвіл групи android.permission-group.CAMERA).

Система дозволів додатку допомагає дотримуватись приватності користувача, захищаючи доступ до:

1. **Даних з обмеженим доступом**, такі як стан системи та контактна інформація користувача.

2. **Дії з обмеженим доступом,** такі як під'єднання до іншого пристрою по Bluetooth або запис звукових доріжок.

Увесь набір дозволів [85] поділяється на групи, які, у свою чергу, складаються зі списку самих дозволів (табл. 2.1).

Таблиця 2.1 – Групи Android Permission та список відповідних дозволів

Група	Список дозволів
android.permission-group.AFFECTS_BATTERY	android.permission.CHANGE_WIFI_MULTICAST_STATE
	android.permission.FLASHLIGHT
	android.permission.TRANSMIT_IR
	android.permission.VIBRATE
	android.permission.WAKE_LOCK
android.permission-group.APP_INFO	android.permission.GET_TASKS
	android.permission.KILL_BACKGROUND_PROCESSES
	android.permission.MANAGE_ACTIVITY_STACKS
	android.permission.PERSISTENT_ACTIVITY
	android.permission.REAL_GET_TASKS
	android.permission.RECEIVE_BOOT_COMPLETED
	android.permission.REMOVE_TASKS
	android.permission.REORDER_TASKS
	android.permission.RESTART_PACKAGES
android.permission-group.AUDIO_SETTINGS	android.permission.MODIFY_AUDIO_SETTINGS
android.permission-group.BLUETOOTH_NETWORK	android.permission.BLUETOOTH
	android.permission.BLUETOOTH_ADMIN
	android.permission.BLUETOOTH_MAP
	android.permission.BLUETOOTH_PRIVILEGED

Така категоризація дозволів та відповідних ним API функцій дозволяє у подальшому конкретизувати напрямок вектору атаки шкідливого додатка при нотифікації користувача [44].

SDK Android та система доступу дає багаті можливості для роботи зі смартфоном, наприклад, є можливість отримувати фотографії, відстежувати переміщення смартфону, записувати звук з мікрофона, перехоплювати SMS-повідомлення. Перераховані можливості потрапляють під категорію конфіденційні. На жаль, до недавнього часу єдиний спосіб дізнатися, які дозволи буде використовувати додаток, можна було побачити тільки в діалоговому вікні при інсталяції додатку. На практиці цей підхід виявився неефективним, тому що користувач зазвичай не читає і автоматично погоджується з установкою. У результаті в Google Play Store [86] з'явилася багато додатків, які отримували доступ до дозволів, які абсолютно не потрібні для виконання своїх завдань. З виходом Android 6 (API Level 23) ситуація змінилася [87]. З'явилися динамічні дозволи, які необхідно приймати під час виконання додатку. Працює так само, як і в iOS – при запуску програми з'являється системний діалог, який і пропонує користувачеві дозволити або заборонити доступ до функцій SDK. Усі привілеї доступу діляться відповідно до двох рівнів захисту поділені на два типи:

- звичайні (normal);

- небезпечні (dangerous);

Звичайні дозволи – це дозволи, які приймаються користувачем автоматично без його згоди при установці додатку.

Інші (небезпечні) дозволи вимагають спеціального запиту у користувача. Небезпечні дозволи мають доступ до приватних даних користувача, способу їх зберігання та системним компонентам операційної системи. Усі небезпечні дозволи характерні для зловмисного додатка, який може потенціально використати їх для доступу до приватних даних, тобто є першочерговим завданням для інформаційної безпеки (на відміну від цієї дисертаційної роботи, де основним завданням є підтримка функціональної безпеки). Наприклад, якщо

зловмисному додатку надано доступ для читання/надсилання SMS-повідомлень, може бути багато наслідків. По-перше, це може мати наслідки для конфіденційності користувача. Так, користувач може надсилати повідомлення, що містять власні конфіденційні дані, або він може отримувати зловмисні SMS-повідомлення. По-друге, якщо користувач отримує, надсилає або обробляє конфіденційні дані, такі як PHI (Protection Health Information – захищена інформація про здоров'я) або PCI (Payment Card Industry – дані платіжної картки) [88], у разі виявлення цих даних можуть виникнути цивільні наслідки, що може вплинути на користувача або його організацію.

Окрім того, слід враховувати, що основним показником рівня небезпеки з погляду функціональної безпеки є можливість змінювати поточний стан системи, блокуючи прямий доступ до ресурсів операційної системи для користувача. Таким чином, зі списку всіх дозволів необхідно виділити підмножину дозволів, які можуть так чи інакше негативно вплинути на функціональну безпеку.

Оскільки дослідницький проєкт «Android Malgenome Project» не дає ніякої інформації щодо назви або опису впливу, необхідно сформувати додаткові метадані, які визначають вектор атаки шкідливого додатка. До цих метаданих (атрибутів) слід віднести: напрям атаки (група дозволів), характер атаки (дії через певні дозволи) і рівень небезпеки. Базуючись на даних офіційного сайту Google як основного розробника ОС Android, можна виділити множину небезпечних для функціональної безпеки дозволів, що впливають на поточний стан ОС, блокуючи для користувача прямий доступ до критичних ресурсів мобільного пристрою. Така множина може бути представлена у вигляді списку небезпечних дозволів разом із виділеними рівнями небезпеки для користувача [89], фрагмент якого представлено на рис. 2.1.

Визначення рівня небезпеки дозволів відбувається з урахуванням можливості зміни стану апаратних ресурсів ОС, що може бути спричинено зловмисним ПЗ та створювати загрозу функціональній безпеці.

Тобто, якщо функція, яка належить дозволу, має можливість міняти поточний стан аппаратного компоненту, наприклад здійснювати безконтрольний запис у файлову систему або утримання камери тощо, то API функція цього дозволу отримує максимально небезпечний (дуже високий) рівень небезпеки для мобільного пристрою.

WRITE_CALENDAR (середній);
CAMERA (помірно високий);
WRITE_CONTACTS (помірно високий);
ACCESS_FINE_LOCATION (помірно високий);
ACCESS_COARSE_LOCATION (помірно високий);
RECORD_AUDIO (помірно високий);
CALL_PHONE (високий);
RECEIVE_BOOT_COMPLETED (помірно високий);
WRITE_CALL_LOG (дуже високий);
ADD_VOICEMAIL (середній);
USE_SIP (середній);
PROCESS_OUTGOING_CALLS (дуже високий);
BODY_SENSORS (середній);
SEND_SMS (високий)
RECEIVE_SMS (дуже високий);
READ_SMS (дуже високий);
RECEIVEW_APP_PUSH (дуже високий);
RECEIVE_MMS (високий);
WRITE_EXTERNAL_STORAGE (дуже високий).

Рисунок 2.1 – Фрагмент списку небезпечних дозволів з вказаними рівнями небезпеки

Всього пропонується виділити 4 рівня небезпеки, наведені в табл. 2.2.

Таблиця 2.2 – Виділені рівні небезпеки для мобільного пристрою

Рівень небезпеки	Опис
Середній	Широко використовується шкідливим ПЗ для перехоплення приватних даних

Рівень небезпеки	Опис
	але також може впливати на функціональну безпеку
Помірно високий	Використовується доволі складним шкідливим ПЗ яке може мати функцію автостарту та мати доступ до чутливої інформації користувача
Високий	Використовується доволі складним шкідливим ПЗ яке може запускати автономні сервіси, розсилати та обробляти чутливу інформації користувача
Дуже високий	Використовується складним шкідливим ПЗ яке може приймати небажані повідомлення без нотифікації користувача, безконтрольно використовувати апаратні компоненти ОС та виконувати системні функції

2.2 Побудова СФЛД на базі проекту “Malgenome”

Популярність та впровадження смартфонів сильно стимулювали поширення шкідливих програмних додатків для мобільних пристроїв, особливо на таких популярних платформах, як Android. У світлі їх швидкого зростання [90, 91] існує нагальна потреба в розробці ефективних рішень щодо захисту користувача від усіх видів зовнішніх загроз. Однак захисна здатність значною мірою обмежена розумінням цих нових мобільних шкідливих програм та відсутністю своєчасної реакції на відповідні додатки.

На сьогодні існує багато спроб систематично охарактеризувати шкідливі додатки з різних аспектів [92, 93], включаючи способи їх встановлення, механізми активації, а також характеристики та подальшу здатність обійти виявлення з боку наявного мобільного антивірусного програмного забезпечення. Саме остання властивість вимагає необхідності кращої розробки мобільних антивірусних програмних комплексів наступного покоління. У межах цієї роботи як основна характеристика поведінки програмного додатку розглядається набір викликів API функцій операційної системи, який в подальшому називається сігнатурою функціонального ланцюжка додатку (СФЛД).

Як основну інформаційну базу для побудови і дослідження СФЛД був використаний дослідницький проєкт Malgenome [94], який надає набори послідовностей API функцій шкідливих додатків у вигляді «.csv» файлу. Приклад фрагмента шкідливого додатка з проєкту Malgenome наведено на рис. 2.2.

На рисунку 2.2 кожен стовпчик таблиці являє собою набір викликів API функцій та запитів на дозволи від операційної системи Android. Рядки описують наявність виклику для відповідного додатку в бінарному форматі «0» або «1».

1	transact	bindService	onServiceConnected	ServiceConnection	android.os.Binder	READ_SMS	attachInterface	WRITE_SMS
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	1
6	1	1	1	1	1	1	1	1
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	1
9	0	0	0	0	0	0	0	0
10	0	0	0	0	0	1	0	1
11	0	0	0	0	0	1	0	1
12	0	0	0	0	0	1	0	1
13	0	1	1	1	1	0	0	0

Рисунок 2.2 – Фрагмент сигнатури функціонального ланцюжка з проєкту Malgenome

Перший етап побудови СФЛД полягає в обробці всіх ланцюжків з проекту

“Malgenome” шляхом зчитування та подальшої фільтрації виключно функціональних викликів Android SDK. Таким чином, після зчитування, ми отримаємо ланцюжок API викликів, який можна зберегти у реляційній базі даних.

Структуру СФЛД в реляційній базі даних можна представити відношенням багато до багатьох, де з одного боку відношення шкідливий додаток в бінарному форматі (за наявності) разом з описом в текстовому форматі (таблиця 2.3), а з другого - API функції SDK (таблиця 2.4).

Таблиця 2.3 – Структура таблиці шкідливого додатка

ID	APK	Опис шкідливого додатка
1	Null або <бінарні дані>	Додаток може блокувати дисплей пристрою, має функцію автостарту

Таблиця 2.4 – Фрагмент таблиці API функцій SDK

ID	API функція	Псевдосимвол
1	FileInputStream.getChannel	A
2	FileChannel.map	B
3	File.delete	C
4	FileInputStream.close	D
5	Ljava.lang.Class.getField	E
6	File.exists	F
7	AlertDialog.Builder	G
8	alertDialog.setCancelable(false)	H
9	android.os.IBinder.bindService	I

В табл. 2.4 для спрощення застосовані позначення API функцій псевдосимволами, представленими буквами англійського алфавіту.

Необхідно зазначити, що виходячи з швидкості розвитку росту шкідливого програмного забезпечення на ОС Android, важливо постійно оновлювати шаблонний набір даних новими зразками, щоб ефективно продовжувати тестування додатків. Однак шкідливе програмне забезпечення

нульового дня все одно спричинить ускладнення, так як потрібен час для створення та збору трейс-файлів та іншої аналітичної інформації. Таблиця 2.3 відтворює схему зберігання шкідливого додатку разом з текстовим описом, якщо такий присутній, інакше опис формується з сегментів псевдосимволу, детальний процес побудови якого буде розглянутий далі.

Якщо фізичний АПК файл шкідливого додатка буде доступний, то це створює можливості для використання різних способів реверсивного інжинірингу [95] та для вивчення особливостей саме цього представника сімейства шкідливих додатків. Це дозволяє розширити спектр евристичного аналізу для подальшого удосконалення методу виявлення потенціально шкідливого додатка. Небезпека від шкідливого додатку фіксується за допомогою табл. 2.5, яка зв’язує СФЛД з відповідним рівнем небезпеки.

Таблиця 2.5 – СФЛД та рівень небезпеки шкідливого додатку

ID	СФЛД	Рівень небезпеки
1	ABCDEFGHI	високий
2	CPCFAF	помірно високий
3	KPFA	середній

Значення поля «Рівень небезпеки» в табл. 2.5 визначається найбільшим рівнем небезпеки для функцій, присутніх у СФЛД.

Таким чином, використання набору даних Malgenome при певній їх обробці, надає широкий спектр метаданих, формуючих вектор атаки шкідливого програмного забезпечення, які можна отримати, аналізуючи ланцюжкі викликів API функцій та Android привілеїв (дозволів). Використовуючі ці метадані можна виявити зловмисне програмне забезпечення нульового дня, яке може призвести до порушення функціональної безпеки модільного пристрою, але яке ще не позначено як зловмисне програмне забезпечення в Google Play Store.

2.3 Моделювання псевдосимволу СФЛД

Для того, щоб забезпечити ідентифікацію вектора атаки за СФЛД, необхідно дати семантичне тлумачення позначення псевдосимволу у вигляді букв алфавіту.

Назва API функції в Android SDK представляє собою строкову константу, здебільшого від 30 до 100 символів, яка складається з повної назви пакета, назви класу та самої функції SDK Android. Слід зазначити, що хоча така строкова константа і надає інформацію про певні атрибути вектора атаки, її використання напряму є дуже ресурсоємним при проведенні порівняння СФЛД. Нижче пропонується модель псевдосимволу API функції, яка значно спрощує цей процес.

Для того щоб програмний додаток міг використовувати будь-яку функцію ресурсів операційної системи, йому необхідні дозволи (android permissions), які користувач надає під час інсталяції або виконання додатка. У свою чергу, кожен дозвіл належить до своєї групи дозволів операційної системи Android [96, 97]. Отже, кожну функцію можна закодувати у вигляді унікальної трьохсегментної послідовності, яка складається з: номера групи, номера дозволу, який належить цій групі, та номеру функції всередині цього дозволу. Таблиця 2.6 демонструє опис дозволу, клас функцій та текстове повідомлення у форматі, зручному для нотифікації користувача.

Таблиця 2.6 – Метадані для нотифікації користувача

Дозволи	Група	Клас (тип даних)	Опис
WAKE_LOCK	PHONE	android.os.powermanager. wakelock	Підтримує процес в активному режимі та утримує екран від засинання
VIBRATE	PHONE	android.os.vibrator	Доступ до вібровідгуку
USE BIOMETRIC	PHONE	android.hardware.biometrics. BiometricPrompt	Підтримка біометричного модуля, який використовується в пристрої

Дозволи	Група	Клас (тип даних)	Опис
RECEIVE BOOT COMPLETED	PHONE	android.content.Broadcast Reciever	Доставка системного сповіщення, що пристрій завантажився
INTERNET	NETWORK	android.net.*	Відкриття системного мережевого socket
FOREGROUND SERVICE	SERVICE	android.content.intent	Використання сервісу в режимі переднього плану
DISABLE KEYBOARD	PHONE	android.app.KeyManager	Вимкнення функції keyguard якщо це небезпечно
CHANGE WIFI STATE	INTERNET	android.net.Connectivity Manager	Зміна стану бездротового мережевого модуля
CHANGE NETWORK STATE	INTERNET	android.net.Connectivity Manager	Зміна стану мережевого модуля
BLUETOOTH ADMIN	BLUETOOTH	android.bluetooth.Blueto oth Class	Пошук та встановлення з’єднання з іншими пристроями
BLUETOOTH	BLUETOOTH	android.bluetooth.Blueto oth Class	Під’єднання до уже відомих пристроїв
ACCESS WIFI STATE	INTERNET	android.net.WiFi.Info	Доступ до поточного стану Wi-Fi мереж

На рис. 2.3 показана загальна схема бази даних шаблонних СФЛД.

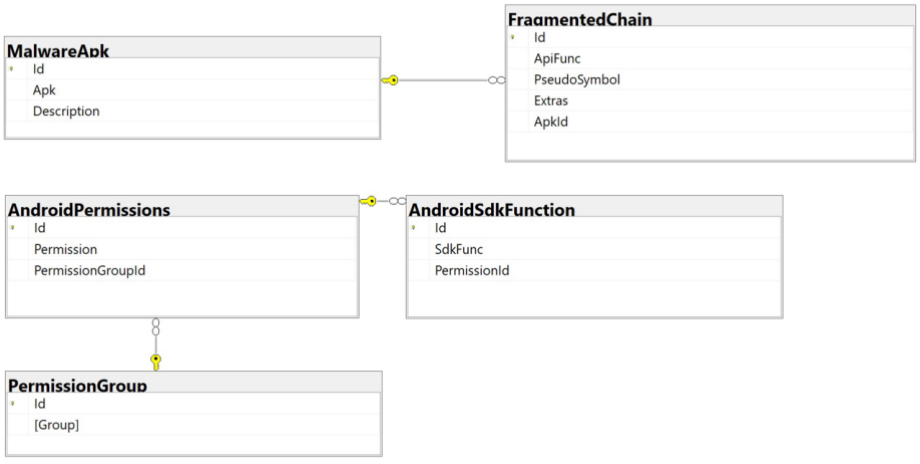


Рисунок 2.3 – Загальна схема бази даних шаблонних СФЛД

Оскільки база даних містить перелік назв API викликів у строковому представленні (API функцій), порівняння послідовностей СФЛД є доволі ресурсномістке завдання, яке можна значно спростити, якщо замінити API функцію на псевдосимвол. При формуванні такого псевдосимволу необхідно враховувати, що для того, щоб програмний додаток міг використовувати будь-яку функцію для доступу до ресурсів операційної системи, йому необхідні дозволи (android permissions), які користувач надає під час інсталяції або виконання додатку. У свою чергу, кожен дозвіл належить до своєї групи дозволів ОС Android. Отже, використовуючи створену базу даних СФЛД, кожну функцію можна закодувати псевдосимволом, який складається з трьох атрибутів: номера SDK-групи, номера дозволу, що належить цій групі, та номера API функції [98].

Математична модель псевдосимволу має вигляд трійки:

$$s = (g, r, f), \tag{2.1}$$

де $s \in S$ – множині псевдо символів, $g \in G$ – множині груп дозволів, r $\in R$ – множині дозволів, f $\in F$ – множині API функцій.

Між множинами G та R, як і між множинами R та F, встановлюється відповідність $1 \to \infty$ (1 до багатьох), а множини S та F знаходяться у взаємно однонозначній відповідності – множина псевдосимволів має потужність, однакову з потужністю множини API функцій. Побудова псевдосимволу починається у зворотному порядку, тобто спочатку знаходиться індекс функції в таблиці функцій SDK, далі знаходиться позиція дозволу зі списку android.permissions [99, 100], в який входить ця функція, та група, до якої входить знайдений дозвіл.

Такий підхід, за рахунок зменшення довжини аналізованої послідовності символів у порівнянні зі строковим представленням API викликів, дозволяє пришвидшити пошук збігів СФЛД та, одночасно, покращити інформативність при нотифікації користувача. Алгоритм побудови символу несе ресурсне навантаження лише на початку побудови бази шаблонних послідовностей, коли

база поповнюється новими зразками.

Детально розглянути трансформацію позначення API функції можна на прикладі функції android.net.ConnectivityManager.setProcessDefaultNetwork (рис. 2.4).

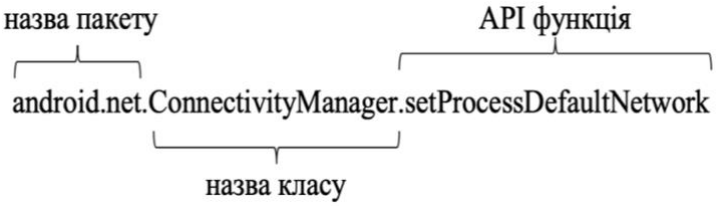


Рисунок 2.4 – Приклад позначення API функції в Android SDK

Після використання запропонованої моделі побудови псевдосимвола API функція буде мати вигляд, наведений на рисунку 2.5.

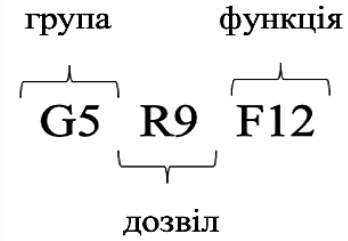


Рисунок 2.5 – Приклад моделювання псевдосимволу API функції

Як видно, для наведеного прикладу початкова строкова репрезентація має 56 символів, а псевдосимвол 7 символів. Саме таке позначення буде використовуватись при пошуку збігів послідовностей.

2.4 Моделювання СФЛД

Відповідно до моделі прав доступу на основі запропонованих псевдосимвольних конструкцій можна побудувати математичну модель послідовності API функцій, за допомогою якої виконується порівняння аналізованої та шаблонної СФЛД. Позначимо всю множину шаблонних СФЛД, що відповідають шкідливим додаткам, як

$$P = \{P_k\},\ k = \overline{1, K}, \qquad (2.2)$$

де K – кількість шаблонних СФЛД у базі даних. У свою чергу, кожний шаблон СФЛД є ланцюжок API функцій

$$P_k = \left(p_j^k\right),\ k = \overline{1, K}, j = \overline{1, n}, \qquad (2.3)$$

де n – довжина k-ї шаблонної послідовності, j – порядковий номер API функції у k-му ланцюжку.

Під час запуску додатку ми отримаємо ланцюжок викликів API функцій, який позначимо як вектор

$$\mathrm{C} = (\mathrm{c_i}),\ \mathrm{i} = \overline{1, \mathrm{m}}, \qquad (2.4)$$

де m – довжина зафіксованого фрагмента послідовності API функції.

СФЛД аналізуємого додатку, як і шаблонні СФЛД, можна представити у вигляді набору псевдосимволів, які значно легше порівнювати при використанні будь-якого алгоритму вирівнювання послідовностей

2.5 Базові методи вирівнювання послідовностей

Вирівнювання послідовностей – це процес порівняння різних послідовностей шляхом пошуку серії окремих символів або шаблонів символів, що мають однакове розташування в обох послідовностях. Існує три основних типи вирівнювання послідовностей: попарне вирівнювання послідовностей, багаторівневе вирівнювання послідовностей та структурне вирівнювання послідовностей. Вирівнювання парних послідовностей одночасно може використовуватися лише між двома послідовностями. Вирівнювання декількох послідовностей – це продовження попарного вирівнювання, що включає більше двох послідовностей одночасно. Структурне вирівнювання послідовностей аналізує всю структуру білкового ланцюга, на відміну від попарного та багаторівневого вирівнювання послідовностей [101]. Вирівнювання послідовностей може бути локальним або глобальним. У той час як локальні вирівнювання знаходять найкращий збіг між двома послідовностями, глобальні

вирівнювання знаходять найкращий збіг за загальною довжиною різних послідовностей, що беруть участь у вирівнюванні. Більшість вирівнювань послідовностей виконуються попарно й засновані на локальних вирівнюваннях (існує три основних методи створення таких попарних вирівнювань: матричних точок, динамічне програмування та методи слів).

У питанні порівняння послідовностей широке практичне застосування набули методи, які використовуються в біоінформатиці. Одна зі сфер цієї галузі лежить у площині застосування машинних алгоритмів для аналізу довгих послідовностей та вирівнювання послідовностей. Оцінка подібності між біологічними послідовностями, мабуть, є основним засобом, за допомогою якого біоінформатика сприяє розумінню біології. Найпоширенішими біоінформативними інструментами для досягнення цієї мети є базовий інструмент локального пошуку вирівнювання (BLAST) та швидке вирівнювання (FASTA), які проводять порівняння між парами послідовностей на основі регіонів локальної подібності [102].

Незважаючи на той факт, що методи, представлені вище, були результатом еволюції різних наук, таких як інформатика, біоінформатика та інших, їх застосування не обмежується лише цими галузями. Нині вони знаходять своє застосування в ідентифікації, оцінці, розумінні та прогнозуванні змін білка, системній біології, обчислювальній еволюційній біології, машинному навчанні та інших [103].

2.5.1 Метод глобального вирівнювання Нідлмана-Вунша

Мета глобального вирівнювання полягає у вирівнюванні двох послідовностей по всій їх довжині [104]. При виконанні глобального вирівнювання ідентичні або подібні елементи послідовностей зміщують у межах рядка, не змінюючи їх порядку, таким чином, щоб вони були один під одним, вирівнюючи всі довжину послідовностей (рис. 2.6).

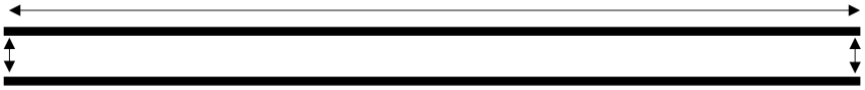


Рисунок 2.6 – Схематичне зображення глобального вирівнювання

Враховуючи те, що зазвичай послідовності вілрізняються, при їх вирівнюванні вставляють розриви (пропуски, англ. gap) для збільшення кількості збігів. Цей прийом використовується при порівнянні послідовностей однакової довжини. Неідентичні, або різні елементи позиціонують як розбіжності та вставляють навпроти них у другій послідовності пропуски.

Основні правила вставки розриву можуть бути пояснені схематично. Так, основна ідея застосування вертикальних та горизонтальних розривів послідовностей може бути проіллюстрована за допомогою рис. 2.7. Якщо в неперервній послідовності виникає розрив (тобто по всьому рядку або колонці немає збігу) по осі X, то пропуск вставляється в ланцюжок, який знаходиться по осі Y (на рисунку 2.7 це послідовність номер два). Це правило працює також і в іншу сторону, тобто якщо розрив виникає по осі Y, то пропуск вставляється в ланцюжок, який знаходиться по осі X (на рисунку 2.7 це послідовність під номером один).

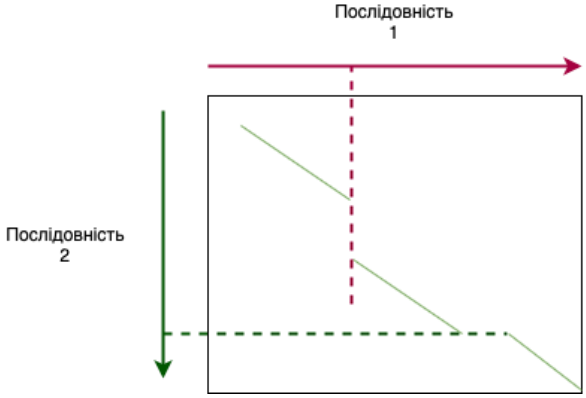


Рисунок 2.7 –Вставка розривів в послідовності

Процедуру вставки розриву розглянемо на прикладі. Припустимо, що необхідно порівняти дві послідовності $S_1 = LPTKCRP$ та $S_2 = LPCRP$, Для цього побудуємо матрицю подібності, представлену на рис. 2.8.

	L	*P*	*T*	*K*	*C*	*R*	*P*
L	**L**						
P		**P**					P
T			**T**				
R						**R**	
P		P					**P**

Рисунок 2.8 – Матриця подібності двох послідовностей

У результаті проходу по матриці з лівого верхнього до правого нижнього краю утворюються два незбіги по осі X, у результаті чого треба вирівняти послідовність S_2, додавши пропуски у відповідні позиції. Результат вирівнювання для даного прикладу наведений на рис. 2.9.

```
LPTKCRP
| | |   | |
LPT--RP
```

Рисунок 2.9 – Результат вирівнювання послідовностей

Щоб мати можливість порівняти потенційні результати вирівнювання послідовностей, потрібно визначати показник (score), який оцінює якість кожного вирівнювання. Формули, що стоять за оцінкою вирівнювання, варіюються від простих сум проходу по матриці до складних значень максимальної ймовірності [105]. У межах цієї роботи буде використаний простий підхід підрахунку сум проходу [106], в якому для оцінки попарного вирівнювання необхідно вказати значення для різних способів порівняння пари елементів:

– числове значення вирівнювання пари, що містять однаковий елемент в обох послідовностях (+1);

– числове значення вирівнювання пари, що містять різні символи в послідовностях (як правило –1);

– числове значення вирівнювання символу в одній послідовності з пропуском в іншій послідовності (як правило 0).

Серед існуючих підходів до глобального вирівнювання найбільш

розповсюдженим є метод Нідлмана-Вунша [107]. В нему лінійна система оцінювання характеризується статичним штрафом за пропуск. Цей підхід проілюстровано на рис. 2.10, на якому показано вирівнювання пари послідовностей CRX та XXCCRXX з використанням оцінки збігу +1, оцінки незбігу -1, та оцінку пропуску -2.

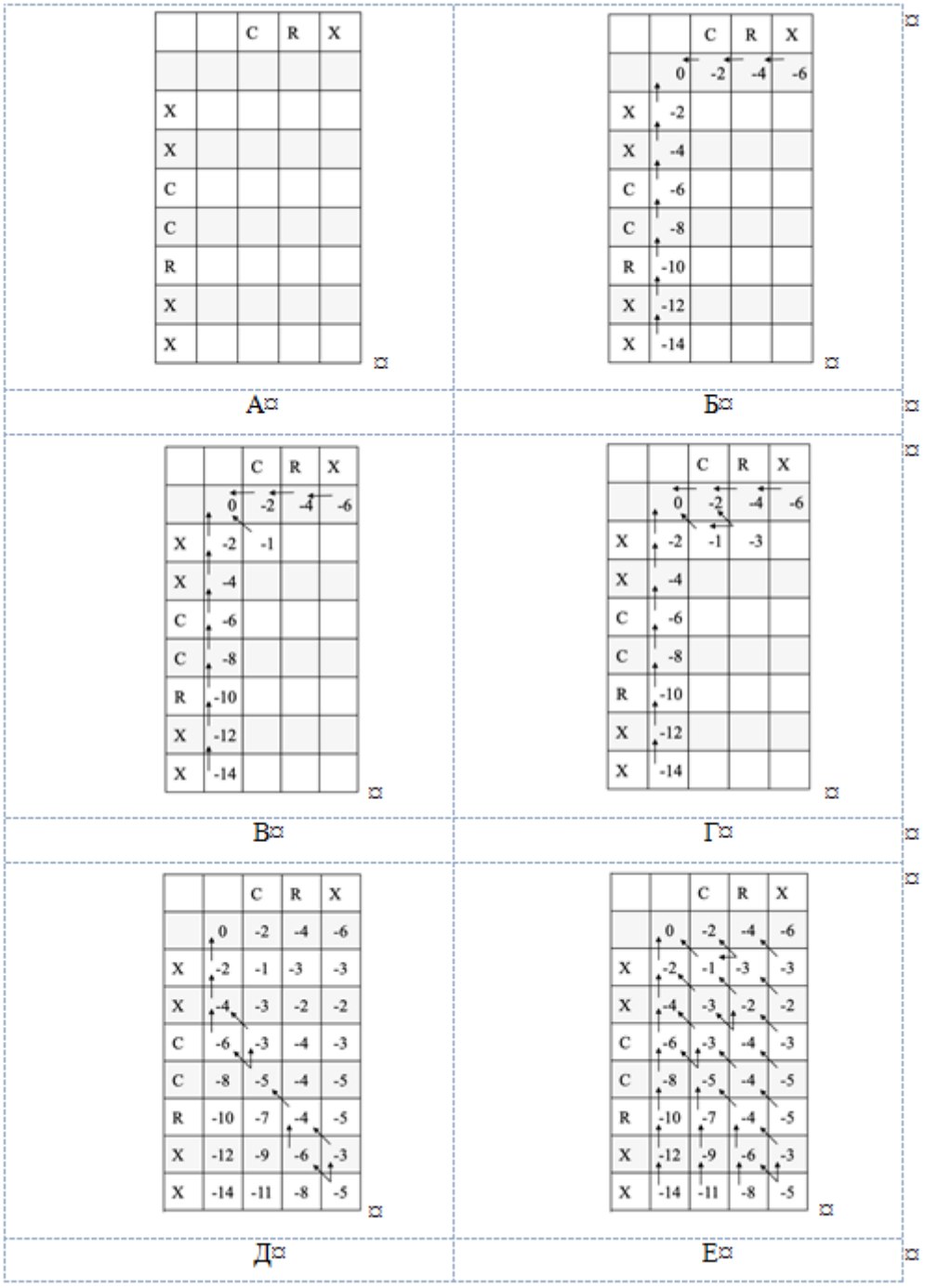


Рисунок 2.10 – Ілюстрація алгоритму глобального вирівнювання Нідлмана–Вунша

Першим кроком є побудова матриці подібності, яка містить кожну послідовність уздовж певної осі, з додатковим порожнім рядком і стовпцем у верхній та лівій сторонах матриці. Кожна позиція в матриці заповнюється максимум трьома можливими значеннями:

1) сума балів клітинки, яка знаходиться по діагоналі її лівого верхнього кута, і оцінка відповідності або невідповідності (залежно від того, чи символи в рядку / стовпці збігаються або ні);

2) сума балів клітинки, яка знаходиться безпосередньо над клітинкою, значення якої обчислюється в даний момент, і оцінка пропуску;

3) сума балів клітинки безпосередньо ліворуч від клітини та оцінка пропуску.

Також існує випадок, коли дозволяється пропустити правило, яке не застосовується. Наприклад, клітинки у верхньому рядку не мають жодних комірок над ними, тому правило 2 не може застосовуватися.

Практичний результат цих правил полягає в тому, що першим кроком є розміщення нуля у верхньому лівому куті матриці, а потім заповнення першого рядка та стовпця зі збільшенням кратних витрат на розрив. Решта комірок матриці заповнюються одна за одною. Важливим моментом є те, що потрібно відстежувати, яке правило було застосовано, тобто яка сусідня комірка (діагональ, вгорі або ліворуч) призвела до заповненого значення. Програмна реалізація вимагає щоб результат проходу зберігався у тимчасовій матриці, яка називається матрицією зворотного зв’язку (на рис. 2.10 це показано для простоти стрілками). Другий важливий момент полягає в тому, що для кількох правил вирівнювання можна створити однакове максимальне значення score, у такому випадку матриця зворотного відстеження повинна належним чином включати всі можливі шляхи до максимального значення.

Процес глобального вирівнювання можна поділити на такі етапи, позначені літерами на рис. 2.10:

А) налаштування матриці;

Б) перший рядок і стовпець заповнюються зростаючими кратно штрафами

за пропуск; у першій комірці буде задано максимум від трьох можливих значень у сміжних клітинках матриці;

В) розраховується значення для першої клітинки, а також вводиться позначення шляху, який привів до цього значення; можливі значення для першої комірки наведені на рис. 2.11.

Діагональ: 0 - 1 = -1
Зверху: -2 - 2 = -4
Знизу: -2 - 2 = -4

Рисунок 2.11 – Підрахунок можливих значень першої комірки

Г) розраховується значення для другої комірки (важлива примітка - записано кілька шляхів, оскільки кілька шляхів призвели до максимального балу); можливі значення другої комірки представлені на рис. 2.12.

Діагональ: -2 - 1 = -3
Зверху: -4 - 2 = -6
Знизу: -1 - 2 = -3

Рисунок 2.12 – Підрахунок можливих значень другої клітинки

Д) заповнена матриця з позначенням діагонального проходу зворотної функції;

Е) завершальна матриця з усіма шляхами; послідовність стрілок від нижнього правого кута до лівого верхнього веде до чотирьох можливих шляхів, отже, чотирьох однаково оптимальних вирівнювань.

У даному прикладі комірка на перетині С та Х може мати три значення: оцінка клітинки у верхньому лівому куті = 0, плюс штраф за невідповідність дорівнює -1; оцінка клітинки над нею, тобто -2, плюс оцінка пропуску -2 у сумі дають -4; оцінка клітинки ліворуч дорівнює -2, плюс оцінка пропуску -2 теж в результаті маємо -4. Максимум від обчислення всіх варіантів виходить -1, який і

заповнюється в клітинку разом зі стрілкою вгорі ліворуч, щоб запам’ятати, що це той шлях, з якого була отримана оцінка. Повторюємо такі дії для наступної клітинки матриці (праворуч) і отримуємо максимальне значення −3, але цього можна досягти за допомогою двох окремих шляхів, тому записується обидва шляхи за допомогою стрілок для зворотного відстеження. Цю процедуру повторюють, поки вся матриця не заповниться значеннями. Значення в останній клітинці, внизу праворуч (–5 у нашому прикладі), являє собою оцінку найкращого вирівнювання. Для цього вирівнювання шлях починається з цієї клітинки і прямує стрілками назад у верхній лівий кут матриці. Діагональна стрілка вказує, що представлені цим рядком і стовпцем матриці значення повинні бути вирівняні. Вертикальна стрілка вказує, що символ у послідовності вздовж вертикальної осі (символ, представлений рядком матриці) повинен бути вирівняний з пропуском у послідовності, представленої горизонтальною віссю. Наступна горизонтальна стрілка вказує на те, що символ у послідовності вздовж горизонтальної осі (символ, представлений стовпцем матриці) повинен бути вирівняний з пропуском у послідовності, що представлена вертикальною віссю.

Якщо існує кілька можливих шляхів повернення до вершини матриці, це вказує на те, що множинні попарні вирівнювання ведуть до однакової оцінки і є однаково оптимальними. У нашому прикладі можливі чотири однаково оптимальні глобальні вирівнювання послідовностей CRX та XXCCRXX, отримані з матриці вирівнювання послідовностей, як це показано на рис. 2.13.

$$
\begin{array}{ccccccc}
X & X & C & C & R & X & X \\
- & - & - & C & R & X & - \\
- & - & - & C & R & - & X \\
- & - & C & - & R & X & - \\
- & - & C & - & R & - & X
\end{array}
$$

Рисунок 2.13 – Набір можливих вирівнювань послідовностей

У цьому конкретному випадку можна стверджувати, що вирівнювання номер 1 є суб’єктивно кращим вирівнюванням, ніж інші, але, виходячи

виключно із зазначеної функції розрахунку пропусків (лінійної), усі ці чотири вирівнювання однакові (кожне з цих вирівнювань містить три збіги та чотири пропуски). Зміна функції підрахунку штрафів за пропуски може змінити результат. У біоінформатиці потужні програмні засоби з великими базами даних, як, наприклад, BLAST [108], використовує евристичні алгоритми для підбору ймовірно найкращого вирівнювання. На практиці дуже мало програм в біоінформатиці знаходять більше ніж одне вирівнювання, навіть якщо існує кілька однаково оптимальних шляхів [109].

2.5.2 Метод локального вирівнювання Смітта-Воттермана

Нині процедура глобального вирівнювання робить припущення, що всі послідовності приблизно однакової довжини. Це припущення може бути неприйнятним з багатьох причин, як в нашому випадку, коли можуть порівнюватися короткі зафіксовані фрагменти СФЛД досліджуємого додатку і довгі послідовності шаблонних СФЛД.

Альтернативним підходом до глобального вирівнювання послідовностей, який обходить вказане вище обмеження, є локальне вирівнювання, в якому частини послідовностей вирівнюються невеликими порціями, або фрагментами (рис. 2.14).

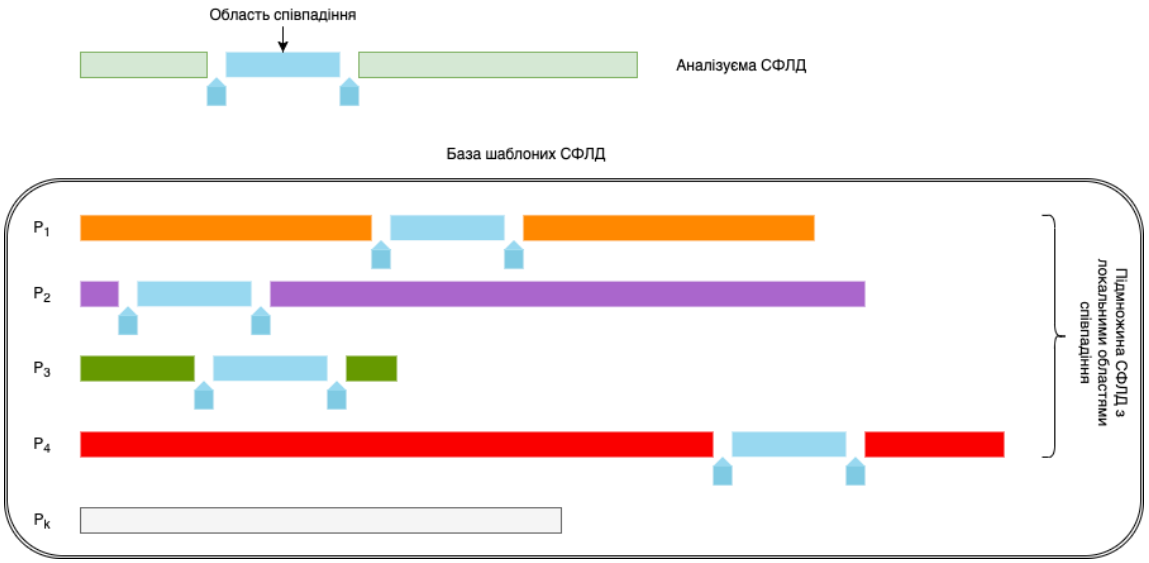


Рисунок 2.14 – Схематичне зображення сегментів локального вирівнювання

Локальне вирівнювання дозволяє вирівнювати області окремо, незалежно від загального порядку в послідовності, дозволяючи при цьому сильно розбіжні області залишати незмінними. Тож, локальне вирівнювання використовується, коли відомо, що є велика кількість шаблонних послідовностей та довжина ланцюжків значно відрізняється як за довжиною, так і, можливо, за порядком у самій послідовності.

Ранні підходи до локального вирівнювання були розроблені Санкоффом та Селлерсом (1979, 1980), але основна процедура, що найбільш широко застосовується, була запропонована Смітом та Ватерманом [110]. Цей алгоритм являє собою модифікацію алгоритму Нідлмена-Вунша до вказаних вище умов. Перша різниця полягає у визначенні значень для клітинки матриці. На додаток до трьох можливих значень, описаних алгоритмом Нідлмана-Вунша, локальний алгоритм вирівнювання допускає четверте можливе значення: нуль. Такий підхід запобігає тому, щоб оцінка вирівнювання ніколи не стала негативною. Якщо це правило виконується, то для клітинки матриці не зберігається стрілка зворотного шляху. Додавання четвертого правила суттєво змінює структуру послідовностей.

Метод локального вирівнювання Сміта-Вотермана має лише три відмінності у порівнянні з методом Нідлмена-Вунша:

1. Початкове значення по краях матриці ініціалізуються нулями.

2. Значення будь-якої комірки матриці не може бути негативним.

3. Відстеження шляху для визначення вирівнювання розпочинається з найвищої оцінки в матриці.

Ці прості правила суттєво змінюють вигляд матриці подібності, що будується в процесі роботи методу вирівнювання, приклад якої наведено на рис. 2.15. Побудова вирівнювання починається з відстеження максимальної оцінки і закінчується, коли на шляху зустрічається нуль. Тож, алгоритм пошуку оптимального шляху відрізняється від глобального вирівнювання тим, що підрахунок може завершитися в будь-який момент, а не в лівому верхньому

куті.

		Z	A	E	T	R	Z	R	N	B	H
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	1	0	0	0	0	0	0	0
T	0	0	0	0	2	1	0	0	0	0	0
I	0	0	0	0	1	1	0	0	0	0	0
Z	0	1	0	0	0	0	2	0	0	0	0
R	0	0	0	0	0	1	0	3	2	1	0
N	0	0	0	0	0	0	0	1	4	3	2

Рисунок 2.15 – Заповнена матриця локального вирівнювання методом Сміта-Ватермана

Зрештою, для наведеного прикладу матриці вирівнювання з оцінкою 4 виглядає так, як показано на рис. 2.16.

E	T	R	Z	R	N
E	T	I	Z	R	N

Рисунок 2.16 – Результат локального вирівнювання методом Сміта-Ватермана

З прикладу можна зробити висновок, що за допомогою алгоритму Сміта-Ватермана можна знайти області збігу аналізуємої та шаблонної СФЛД, при чому зворотній прохід відкидає частини послідовностей, які не відносяться до порівнювальної локальної області.

Метод глобального вирівнювання Нідлмана-Вунша, як і метод локального вирівнювання Сміта-Уотермена, мають однакову часову складність $O(n^2)$.

2.6 Метод динамічного виявлення шкідливих додатків шляхом порівняння СФЛД

Наведені вищі базові методи вирівнювання послідовностей можуть бути використані для побудови удосконаленого методу динамічного виявлення шкідливого додатка [98], який базується на послідовному застосуванні методу та локального вирівнювання Сміта-Уотермена та методу глобального вирівнювання Нідлмана-Вунша.

У термінах математичної моделі СФЛД, описаній у п. 2.4, завдання полягає в тому, щоб знайти шаблонну СФЛД, яка кращим чином відповідає ланцюжку C^q, побудованому в межах певного кванту часу q.

Блок-схема алгоритму запропонованого методу наведена на рис. 2.17.

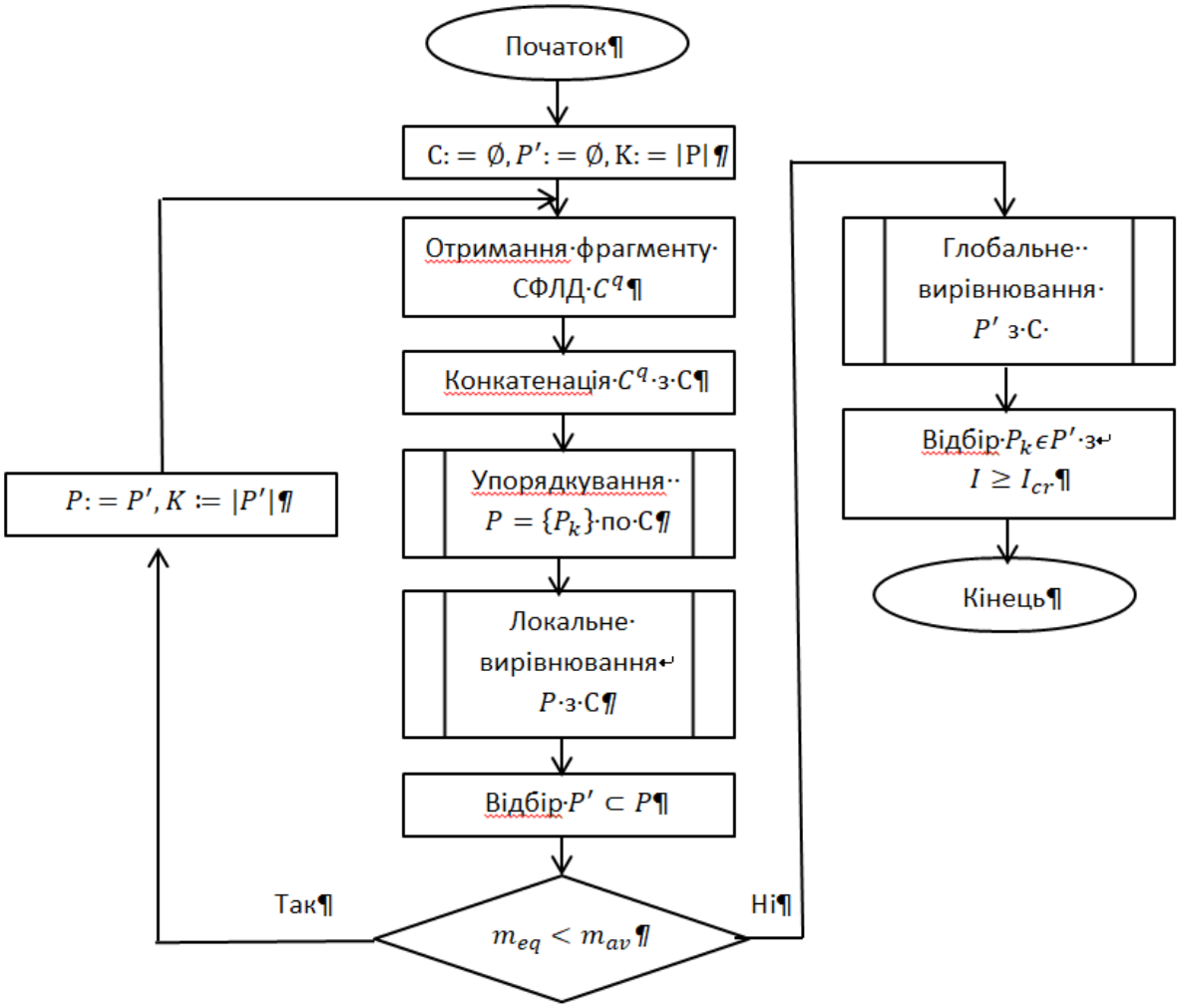


Рисунок 2.17 – Блок-схема алгоритму методу динамічного виявлення шкідливого додатку

У запропонованому методі селекція підмножини СФЛД, яка має локальні області збігів із фрагментованим ланцюжком API-функцій, спочатку відбувається за допомогою методу Сміта-Ватермана. Як вже було відмічено, цей алгоритм зручний при порівнянні двох послідовностей, довжина яких значно відрізняється. У нашому випадку він буде застосований для пошуку повних шаблонів додатку із набору P, подібних фрагментованому ланцюжку C^q. У

цьому випадку послідовності $P_k = (p_j^k)$ та $C^q = (c_i^q)$ є вхідними послідовностями в циклі по k – числу шаблонних СФЛД. Враховуючи, що на початку циклу квантування довжини ланцюжків функцій API можуть бути незначними, ця обставина є вирішальною.

У рамках одного фрагмента (далі індекс q, що індексує фрагмент, для спрощення буде опущено) послідовність API функцій у шаблонній СФЛД, яка являє собою множину функцій, може не відповідати послідовності функцій отриманого ланцюжка $C = (c_i)$. Тому у шаблонній $P_k = (p_j^k)$ необхідно упорядкувати перші m символів відповідно до отриманого ланцюжка $C = (c_i)$. Блок схема алгоритму упорядкування шаблоних СФЛД наведена на рис. 2.18.

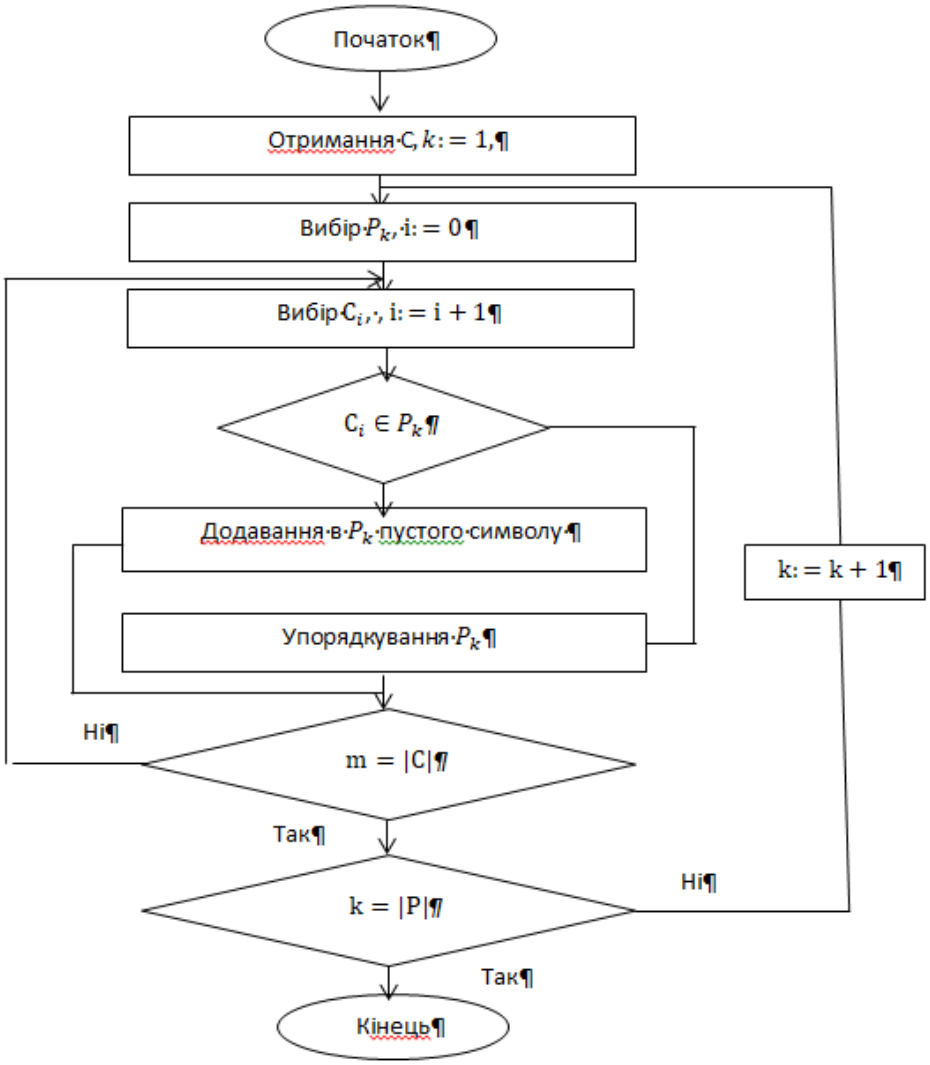


Рисунок 2.18 – Блок схема алгоритму упорядкування шаблоних СФЛД

Якщо для якогось елемента виконуються $c_i \notin P_k$, то p_i^k присвоюється значення неіснуючого символу □. У результаті буде отримана упорядкована СФЛД P_k^0, з якою і буде порівнюватися ланцюжок C.

Далі на кожному кроці циклу по k вирівнювання послідовностей виконується за допомогою матриці подібності $M_{m\times n}$ з елементами $M_{i,j}$, які обчислюються за правилом:

$$M_{i,j} = max\{M_{i-1,j-1} + w;\ M_{i-1,j} + g;\ \ M_{i,j-1} + g; 0\},\ i,j > 1, \qquad (2.5)$$

де w – оцінка за збіг (+1) або невідповідність (–1) символів у рядку i та стовпці j, g – штраф за розрив (g = –1), коли перехід відбувається уздовж рядка або стовпця. Якщо варіанти дають від'ємні значення, результат дорівнює 0.

Матриця заповнюється шляхом обчислення сусідніх значень (діагоналі, верхньої та лівої поточної комірки), починаючи з нульової комірки. Для пошуку оптимального вирівнювання використовується процедура зворотного відстеження шляху, починаючи з комірки з найбільшим значенням і рухаючись до попередніх позицій, поки буде досягнута комірка з оцінкою 0. Серед усіх шаблонів $P = \{P_k\}$ вибираються ті, які мають максимальний бал (без пропусків).

Така процедура повторюється щоразу, коли довжина фрагментованого СФЛД збільшується шляхом додавання знову виявлених функцій API, поки довжина ланцюжка C не стане рівною середній довжині безперервної послідовності API функцій у шаблонних СФЛД, тож поки буде виконуватися умова $m_{eq} < m_{av}$., де m_{eq} – кількість співпадаючих символів ланцюжка C, a m_{av} – середня довжина ланцюжків шаблоних СФЛД. Наприклад, для двох шаблонних послідовностей $P_1 = (D, W, A, B, S, K, C, K, V, B)$, $P_2 = (W, A, B, K, C, W, B)$ та зафіксованої фрагментованої послідовності $C_1 = (A, B, K)$ результати локального вирівнювання можуть мати вигляд, представлений на рис. 2.19.

A□KCKVB

I I

ABK

Рисунок 2.19 – Приклад локального вирівнювання СФЛД

У цьому прикладі, для спрощення, застосовані позначення псевдосимволів СФЛД буквами англійського алфавіту.

Локальний алгоритм вирівнювання послідовностей вирішує завдання пошуку множини потенційно небезпечних СФЛД, які буде використано для подальшого аналізу. Це зменшує кількість СФЛД, які необхідно аналізувати. У результаті буде обрано один або кілька шаблонів, що відносяться до множини $P' \subset P$.

На другому етапі виявлення шкідливого ПЗ використовується алгоритм глобального вирівнювання послідовностей Нідлмана-Вунша. Алгоритм виконує порівняння послідовностей по всій довжині, що необхідно для оцінки збігу аналізованого СФЛД з послідовностями множини P' . На цьому кроці розмірність матриці подібності встановлюється таким чином, щоб забезпечити максимальну узгодженість аналізованих СФЛД по довжині, тож ддорівнює максимальній довжині шаблонних СФЛД. На цьому кроці правило для обчислення значень комірок матриці $M_{m\times n'}$ має таку форму:

$$M_{i,j} = max\{M_{i-1,j-1} + w;\ M_{i-1,j} + g;\ M_{i,j-1} + g;\}, i,j > 1. \quad (2.6)$$

Значення нижньої правої комірки матриці буде найкращим показником вирівнювання з двох послідовностей.

Часова складність алгоритму запропонованого методу з урахуванням складності базових методів вирівнювання та процедури упорядкування шаблонних СФЛД буде складати $O(n^3)$ для кожного кванту часу отримання фрагментованої СФЛД.

На рис. 2.20 наведено приклад глобального вирівнювання для послідовності $\mathrm{C}_2 = (A, B, K, \mathrm{C}, W)$, отриманої в процесі виконання попереднього етапу:

ABKCWB

I I I I I

ABKCWH

Рисунок 2.20 – Приклад глобального вирівнювання СФЛД

Показник порівняння СФЛД (показник ідентичності) повинен бути добре обґрунтованим та зрозумілим, мати одне «підсумкове» число у фіксованому діапазоні, наприклад, від 0% (повністю не збігаються) до 100 % (повністю збігаються). Вторинний показник - подібності буде показувати можливу девіацію фрагментованого ланцюга від шаблонної послідовності. Можливі мінімальні зміни у структурах СФЛД не повинні призводити до значних стрибків у розрахункових значеннях ступеня подібності. Він повинен фіксувати подібність або відмінності між структурами СФЛД на будь-якому заданому рівні точності. В ідеалі ці показники повинні мати інтуїтивну візуальну інтерпретацію, хоча складний характер проблеми перешкоджає загальноприйнятному єдиному рішенню [111].

Пропонується використовувати коефіцієнт ідентичності, який обчислюється за формулою:

$$I = \frac{N}{max\,(L_1,\ L_2)} \times 100, \qquad (2.7)$$

де N – кількість псевдосимволів, які повністю збігаються в обох послідовностях, $L_1,\ L_2$ – довжини порівнюваних СФЛД.

Тоді ризик виявлення шкідливого додатка буде дорівнювати

$$R = 1 - I/100. \qquad (2.8)$$

Для наведеного на рис. 2.21 прикладу коефіцієнт ідентичності дорівнює 83 %, а ризик – 0.17.

Для врахування можливого маскування викликів небезпечних API функцій шляхом мутації певних псевдосимволів, що належать до однієї групи,

пропонується додатково використовувати коефіцієнт подібності СФЛД, який розраховується за формулою:

$$S = \frac{N+E}{max\,(L_1,\ L_2)} \times 100, \qquad (2.9)$$

де E – число псевдосимволів, які не збігаються і належать до однієї групи.

Для прикладу з рис. 2.19, якщо символи B та H належать до однієї групи, коефіцієнт подібності буде дорівнювати 1. Факт виявлення шкідливого додатку встановлюється шляхом порівняння обчисленого значення коефіцієнта ідентичності з його критичним значенням (порогом спрацювання), яке встановлюється за результатами експериментальних досліджень.

2.7 Висновки по розділу 2

1. Проведений аналіз карти дозволів та категоризація викликів АПІ функцій дозволяють сформулювати поняття вектора атаки нкідливого додатку, який включає напрям атаки (групу дозволів), характер атаки (дії через певні дозволи) і рівень небезпеки. Визначення рівня небезпеки дозволів відбувається з урахуванням можливості зміни стану апаратних ресурсів ОС, що може бути спричинено зловмисним ПЗ та створювати загрозу функціональній безпеці.

2. Модель прав доступу на основі запропонованих псевдосимвольних конструкцій, що об’єднують дані про API функції, дозволи та групи дозволів полегшують аналіз послідовностей СФЛД та надають можливість сформувати нотифікацію для користувача про виявлений шкідливий додаток.

3. Як основну характеристику поведінки програмного додатку слід розглядати сігнатуру функціонального ланцюжка додатку, яка представляє собою послідовність викликів API функцій операційної системи під часЮ що утворюється під час роботи додатку.

4. Запропонований метод динамічного виявлення шкідливих додатків, який базується на двух базових методах з біоінформатики, а саме глобальному та локальному вирівнюванню послідовностей, дозволяє побудувати

конструктивний алгоритм не тільки для пошуку шкідливого додатку, а й для оцінки ризику його виявлення щляхом обчислення коефіцієнта ідентичності. Обчислення коефіцієнта подібності допомагає виявити незначні модифікації вектора атаки шкідливого додатка.

Результати досліджень, приведених в розділі, опубліковані в роботах [44,89,98].

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ БЕЗПЕКИ

3.1 Сучасні способи функціонального трасування

Найбільший виклик при функціональному трасуванні API пов'язаний із зворотним інжинірингом (англ. obfuscated) [112] зашифрованих програмних додатків для Android. У той час як програми для Android здебільшого базуються на двох основних мовах Java та Kotlin, розробники вбудовують у власний код шаблони коду «Java Native Interface» (JNI), які написані для системного використання. Цей підхід зазвичай використовується, щоб ускладнити дешифрування скомпільованого коду. На практиці використання JNI може бути зумовлене необхідністю підвищення продуктивності або для підтримки застарілої версії програмного додатка.

Розробники прагнуть ускладнити або зовсім унеможливити зворотний інжиніринг, навмисно розроблюючи двошарові системи, розподіляючи функціональність між Java байт-кодом та системними викликами JNI, що, у свою чергу, робить процес отримання вихідного коду з коду, компільованого APK, складним та ресурсомістким завданням. Як наслідок, потрібно розуміти як байт-код Java, так і ARM асемблер, та мати робочі знання як про середовище Android на базі Java, так і про ОС Linux і ядро, що лежить в основі Android. Такий інтегрований підхід підвищує ймовірність реверсивної декомпіляції [113] будь-якого мобільного програмного комплексу. Крім того, потрібен правильний набір інструментів для роботи як із власним кодом, так і з байт-кодом, що працює в Java віртуальній машині. (JVM)) [114].

Екстракція файлу APK – це процес декомпресії архівного файлу з одного на кілька об'єктів, які можна надалі зчитувати та аналізувати окремо. Після розархівації файлу APK (проста операція розпакування з використанням наявного

на цей момент інструменту стиснення – 7zip, WinRaR та ін.), буде отримано багато придатних для читання об’єктів, наприклад, таких як мультимедійні файли, що використовуються додатком (піктограми, фотографії, відеофайли), хоча вся логіка програми та графічний інтерфейс програми приховані в уже скомпільованих бінарних файлах, таких як classes.dex та AndroidManifest.xml. Щоб ці файли були читабельними, потрібна подальша декомпіляція [115]. Якщо метою було просто прочитати мультимедійні файли, це вже можна зробити, просто проаналізувавши папку ./res/*.

3.2 Особливості аналізу СФЛД на емуляторі OS Android

Дослідники та аналітики зловмисного програмного забезпечення великою мірою покладаються на емулятори або віртуальні пристрої через те, що це середовище аналізу характеризується порівняно низькою ціною. Емулятори також є більш привабливими для автоматизованого масового аналізу, який спільно використовується в машинному навчанні. Багато алгоритмів та механізмів виявлення шкідливого програмного забезпечення з динамічним аналізом покладаються на аналіз API функцій за допомогою інструментів, що працюють у середовищі емулятора. В ідеальному сценарії досліду необхідно використовувати реальні фізичні мобільні пристрої для динамічного аналізу, щоб уникнути проблеми використання антиемуляційних систем, які використовуються зловмисними програмами Android для уникнення виявлення.

Сучасні системи емуляції фізичних мобільних пристроїв здебільшого використовуються для розробки нових мобільних додатків або для різних типів аналізу дослідницькими проєктами в цій сфері [116]. Враховуючи те, що більшість середовищ аналізу, які запускаються в межах системи емуляції, мають багато статичних та динамічних відмінностей від реального пристрою, що використовується користувачем, такі середовища можуть бути легко скомпрометовані шкідливим програмним додатком, яке може мати на меті

заволодіння приватними даними (інформаційна безпека) або тією чи іншою мірою блокувати роботу пристрою (функціональна безпека).

Шкідливі додатки аналізують середовище виконання, використовуючи системні виклики, такі як IMEI код, натискання кнопок, акселерометр, координати GPS тощо, щоб ідентифікувати, чи знаходиться програмний процес в емуляторі чи на реальному пристрої. Зрозуміло, що шкідливий додаток може не проявити себе і таким чином пройти етап динамічного аналізу, залишаючись невизначеним. При побудові СФЛД вразливості такого роду необхідно вирішити на етапі інсталяції будь-якої системи емуляції. Це робиться з подвійною метою: наслідувати штучну поведінку користувачів та проаналізувати справжню поведінку шкідливого програмного додатка. Тобто до моменту побудови СФЛД необхідно створити середовище, яке б було максимально близьке до фізичного пристрою. Такий підхід називається антиемуляцією.

Детальний набір послідовних кроків, які можуть значно ускладнити процес детекції середовища, описаний у посібнику з тестування мобільної безпеки від OWASP [116], де наведено кілька прикладів щодо антиреверсивних методів та різних способів їх аналізу. Існує також Android API, який називається SafetyNet [117], що створює профіль пристрою для перевірки різних варіантів захисту додатка ОС Android.

Перевіряючи різні API виклики, такі як “Build”, “TelephonyManager”, “android.os.SystemProperties”, “ro.product.device”, “ro.kernel.qemu” та інші, можна зробити висновок, працює мобільний додаток на фізичному пристрої чи в системі емуляції операційної системи. За допомогою існуючих інструментів реверсивного інжинірингу, таких як apktool, jadx або cfr, можна отримати вихідний код додатка та перевірити наявність будь-якого модулю антиемуляції, що має на меті приховати реальну поведінку системи [118]. У даний час, багато зусиль направленні на виявлення та видалення артефактів шкідливого

програмного забезпечення але цього недостатньо: надзвичайно важливо зрозуміти, як вони діють, зрозуміти контекст, мотиви та цілі зловмисного ПЗ.

Більшість антиемуляційних програмних підходів використовують стандартні властивості мови програмування, тобто після системного виклику, який отримує будь-яку інформацію про середовище виконання, програмна перевірка використовує набір методів класу String, таких як “equals”, “contains”, “startsWith” or “endsWith” з деякими жорстко закодованими константами, які можна інтерпретувати як такі, що встановлені емулятором.

Незважаючи на те, що компанія Google є прямим і найактивнішим розробником ОС Android, компанія також активно веде розробку антиемуляційних плагінів та систем. Так, в наведеному нижче на рис. 3.1 прикладі коду з кросплатформеного фраймворка Flutter [119] цієї компанії додаток намагається ідентифікувати середовище.

```
fun isEmulator(): Boolean {
    return (Build.BRAND.startsWith( prefix: "generic") && Build.DEVICE.startsWith( prefix: "generic"))
            || Build.FINGERPRINT.startsWith( prefix: "generic")
            || Build.FINGERPRINT.startsWith( prefix: "unknown")
            || Build.HARDWARE.contains( other: "goldfish")
            || Build.HARDWARE.contains( other: "ranchu")
            || Build.MODEL.contains( other: "google_sdk")
            || Build.MODEL.contains( other: "Emulator")
            || Build.MODEL.contains( other: "Android SDK built for x86")
            || Build.MANUFACTURER.contains( other: "Genymotion")
            || Build.PRODUCT.contains( other: "sdk_google")
            || Build.PRODUCT.contains( other: "google_sdk")
            || Build.PRODUCT.contains( other: "sdk")
            || Build.PRODUCT.contains( other: "sdk_x86")
            || Build.PRODUCT.contains( other: "vbox86p")
            || Build.PRODUCT.contains( other: "emulator")
            || Build.PRODUCT.contains( other: "simulator");
}
```

Рисунок 3.1 – Приклад коду ідентифікаціїї середовища виконання

Прикладом антиємуляційної системи з широкими можливостями є також «Cuckoo» - провідна система автоматизованого аналізу шкідливих програм з відкритим кодом [120], безкоштовне програмне забезпечення, яке здатне проводити аналіз будь-якого шкідливого файлу під управлінням ОС Windows,

macOS, Linux або Android ОС. Завдяки відкритому коду «Cuckoo» та широкому модульному дизайну можна налаштувати будь -який аспект середовища аналізу, обробку результатів аналізу та етап звітування. «Cuckoo» має усі налаштування для легкої інтеграції пісочниці у існуючий фреймворк та серверну реалізацію, у потрібному форматі [121].

Інший спосіб перевірити середовище виконання – перевірити властивості системи [122]. Деякі властивості системи на емуляторі відрізняються від властивостей реальних пристроїв, наприклад, марка пристрою, апаратне забезпечення та модель. У таблиці 3.1 наведені значення системних властивостей у віртуальному середовищі, в тому числі в емуляторі.

Таблиця 3.1 – Системні виклики, що компроментують емулятор

Назва властивості	Значення яке ідентифікує емулятор
ro.bootloader	Unknown
ro.bootmode	Unknown
ro.hardware	Goldfish
ro.product.model	Sdk
ro.product.device	Generic
ro.product.name	SDK

Загалом є три способи обійти перевірку емулятора:

– модифікувати програму та видалити перевірку емулятора;

– модифікувати емулятор так, щоб результат системних викликів був такий, як і справжній пристрій;

– модифікувати системні виклики програмного додатка, які направлені на виявлення середовища виконання.

Усі три способи вимагають використання складного інструментарію реверсивного інжинірингу та знання програмного середовища модифікованого додатка. Для декомпіляції програми потрібна велика кількість навичок (наприклад, за допомогою apktool для декомпіляції її в код Smali [122]) з визначення перевірок емулятора та обходу перевірок цілісності програми, які можуть існувати.

Використовуючи модифікації емулятора, дослідники намагаються виправити та перекомпілювати емулятор Android, якщо вихідний код доступний. Найпростішим варіантом є встановлення фреймворку XPosed [123] на свій емулятор, що дозволить змінювати системні виклики, які використовує програма для ідентифікації середовища виконання та маскування виконання на емуляторі. Існує кілька модулів XPosed, що допомагають успішно підміняти системні виклики та імітувати поведінку користувацького пристрою, наприклад, модуль RootCloak [124] допомагає обійти перевірку на наявність користувача підвищених привілеїв (root, su). Але будь-який підхід буде допомагати ідентифікувати середовище виконання лише відомих емуляторів та віртуальних машин емуляції. Для підвищення вірогідності виявлення шкідливих додатків в роботі було використано XPosed framework (Anti Android Emulator Detection Solution - AEDS) [125-127].

3.3 Обробка отриманих ієрархічних послідовностей при багатопотоковому виконанні

Усі алгоритми вирівнювання послідовностей, які використовувались в рамках даної роботи, добре працюють на отриманих СЛФД, що генеруються шкідливим додатком у рамках одного потоку. Однак реальна ситуація дуже відрізняється від ідеальних умов. Послідовність системних API функцій може включати виклики з декількох потоків, що значно ускладнює обробку вхідної СФЛД, створюючи так званий “шум” в послідовності (набір функці, які перехоплюються в результаті багатопотокового виконання або будь-яких програмних циклів). Основний потік процесу може створювати декілька потоків, що, у свою чергу, може створювати більше потоків (concurrency and structured concurrency) [128]. Це призводить до того, що послідовність API функцій може мати складну деревоподібну структуру. Простий спосіб поєднання API функцій із декількох потоків (послідовна агрегація) в одну послідовність СФЛД може призвести до аномального вирівнювання

послідовностей або зовсім низьких коефіцієнтів вирівнювання, не зважаючи на його спосіб (глобальне або локальне вирівнювання).

Проаналізувавші існуючі підходи [129], для вирішення цієї проблеми запропоновано рекурсивний алгоритм для обробки розгалужених послідовностей. Вхідні дані цього алгоритму – це одна послідовність викликів процесу, де системні API виклики з усіх потоків хронологічно об’єднуються, тобто кожна подія в послідовності позначається відповідним ідентифікатором потоку (Thread@addr1, Thread@addr2 і т. д. [130]).

Для того щоб зрозуміти як проходить згортка розгалуження перехопленої СФЛД, необхідно ввести поняття «потоковий метавузол» (далі – метавузол) – це місце в послідовності API викликів, яке передує виклику з ідентифікатором потоку, або сама точка розгалуження, якщо цей виклик є перший в послідовності СФЛД. Приклад розгалуження СФЛД при наявності потоку в вихідному коді аналізуємого додатку зображений на рисунку 3.2

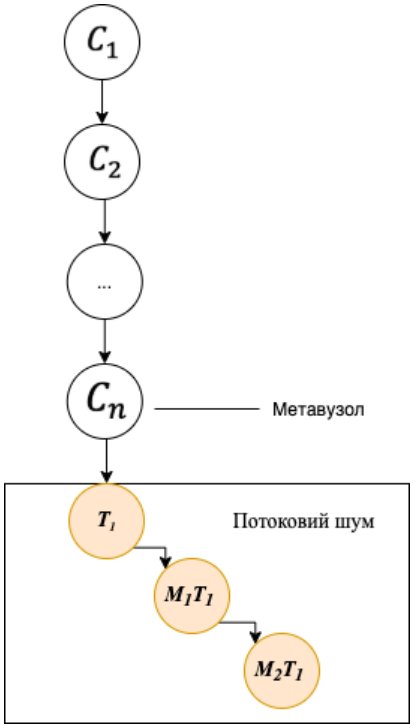


Рисунок 3.2 – Схематичне зображення розгалуження СФЛД при наявності потоків в вихідному коді

Формування структури розгалуженої послідовності розпочинається з перевірки викликів API функцій з початку послідовності. Щоразу, коли зустрічається новий потік, помічається новий метавузол у місці, яке передує виклику, де був створений потік. Це розташування системного виклику (invoke Thread для Java або Thread для Kotlin коду), що відповідає потоку. Таким чином, метавузол являє собою точку, після якої іде розгалуження в основній послідовності. Далі з основної послідовності видаляються всі випадки системних викликів, пов’язаних із новим потоком, які потім додаються до новоствореної послідовності, пов’язаної з метавузлом. Індивідуальне відображення нового виклику потоку та відповідного йому створення нового потоку не завжди можуть бути доступні у профілі виконання, особливо враховуючи лімітовану підтримку suspend функцій у системному відладчику основної IDE для розробки додатків Android Studio на мові Kotlin. Тому призначається нова подія потоку, пов’язана з останнім непризначеним викликом invoke Thread.

Перед проведенням аналізу СФЛД необхідно нормалізувати послідовність викликів при наявності потоків. Для цього всі точки розгалуження з однаковим ідентифікатором потоку рекурсивно згортаються, а їх ідентифікатори видаляються після екстракції самих API функцій цього потоку. API функції після екстракції агрегуються послідовно в метавузол першої події цього потоку, утворюючи неперервну монолітну послідовність СФЛД (рис. 3.3).

В більш складних випадках потокове розгалуження може містити нові розгалуження (потік породжує новий потік). В такому випадку згортання проволдиться від кореневого потоку та рекурсивно повторюється до тих пір, поки не буде отримано монолітну послідовність без розгалужень. Слід зазначити, що згортання проходить після метавузла в першій точці потокового розгалуження. Але такий підхід має ряд недоліків:

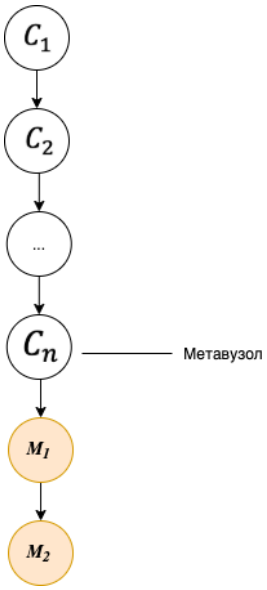


Рисунок 3.3 – Схематичне зображення згорнутої СФЛД після видалення потокового шуму

- якщо зловмисне ПЗ знає роботу алгоритму, то може створити «шумові виклики» на першій потоковій ітерації, що призведе до згортання API функцій, які мають у собі шкідливу складову;
- наявність циклів та умовних операторів всередині потоку може також призвести до втрати корисних API функцій при згортанні розгалужень;
- Suspend функції можуть виконуватись одними й тими самими потоками з загального пулу потоків, що вимагає розробки окремого алгоритму для правильної екстракції API функцій в точці розгалуження;

Можуть існувати й інші інтелектуальні зусилля розробників шкідливого ПЗ направлені на ускладнення аналізу поведінки шкідливого додатку при створенні нових потоків [131].

Особливу складність у неконтрольованому розширені та обробці вхідних послідовностей відіграють цикли (for, foreach, while, do while) або extension functions, які містять цикли. Іноді цикл може створити довгу послідовність повторюваних коротких послідовностей. Таке повторення може містити тисячі системних викликів, що надмірно збільшують вимогу до серверних ресурсів та часу для вирівнювання послідовностей. З цією метою при конкатенації

фрагментованих СФЛД, необхідно визначити послідовно повторювані підпослідовності API функцій та згорнути їх в точках розгалуження. Якщо така послідовність знайдена суміжно та повторюється більше ніж один раз, то необхідно відкинути всі послідовності, які залишилися під час вирівнювання, з метою уникнення тривалої обробки та порівняння СФЛД.

Блок-схема алгоритму рекурсивного видалення потокових та циклічних «шумів» з СФЛД представлена на рис. 3.4.

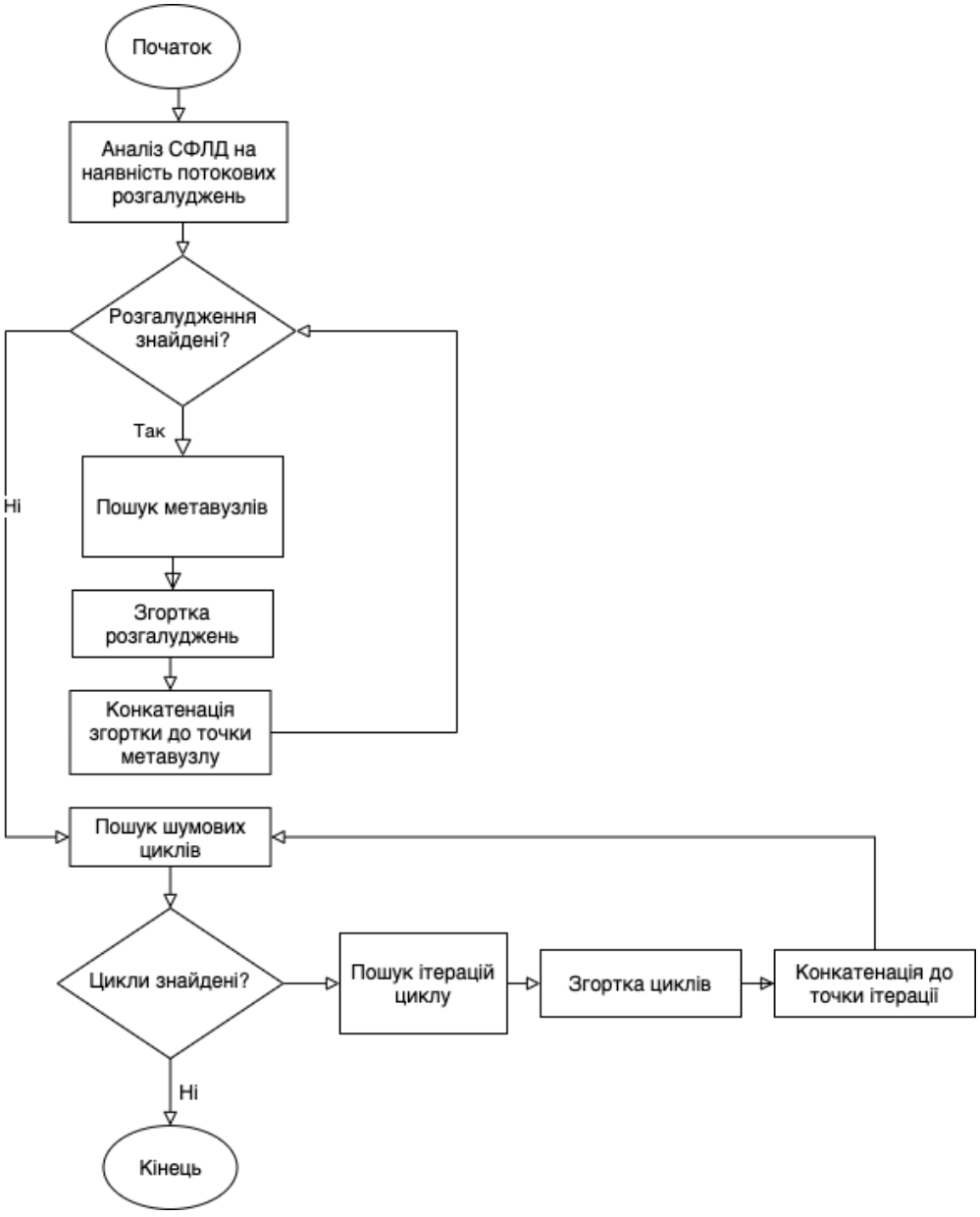


Рисунок 3.4 – Блок-схема алгоритму рекурсивного видалення потокових та циклічних «шумів» з СФЛД

3.4 Побудова СФЛД на базі фреймворка FRIDA

На сьогодні існує велика кількість мобільних пристроїв Android та відповідних магазинів, які містять мільйони різних додатків. Завдяки зростанню популярності цифровізації додатків, які мають доступ до особистих даних, фінансових операцій, приватного спілкування в мережі, дедалі більше зростає необхідність тенденції закритості вихідного коду та способу комунікації додатка з операційною системою. Враховуючи такий вектор розвитку додатків, в будь-якій операційній системі динамічний аналіз програм залишається єдиним інструментом для моніторингу поведінки додатків та захисту користувачів. Сучасні тенденції компаній, які займаються мобільними платформами, спрямовані на розвиток нових та підтримку вже наявних заходів для покращення захисту вихідного коду мобільних додатків, які ускладнюють аналіз та приховують внутрішню роботу програм. Це вторинно ускладнює завдачу динамічного аналізу викликів API функцій. Моніторинг програми під час виконання необхідний, щоб зрозуміти, як вона взаємодіє з ресурсами операційної системи, з ключовими компонентами та її підсистемами.

Операційні системи стандартно забезпечують два режими роботи: низькопривілейований режим користувача для виконання визначених користувачем програм, які не є частиною операційної системи, та режим високопривілейованих програмних підсистем, які працюють на рівні ядра для виконання коду операційної системи (системні драйвери та сервіси) [132]. Оскільки режим ядра має прямий доступ до апаратного забезпечення системи, важливо обмежити доступ до цього режиму. Зазвичай це досягається за допомогою кільця захисту або привілеїв, які забезпечуються обладнанням. Майже всі користувальницькі програми вимагають сервісів, що надаються ядром, таких як користувальницький інтерфейс, операції вводу-виводу, управління файлами, зв’язок з іншими процесами та API виклики. API визначає, як повинні діяти деякі програмні компоненти (підпрограми,

протоколи та інструменти), якщо їх викликають інші компоненти. Відстежуючи та аналізуючи ці взаємодії, можна з'ясувати, як програми поводяться, обробляють конфіденційні дані та взаємодіють між собою. Android пропонує набір функцій API для доступу до захищених ресурсів. Схема виклику API функції зображена на рис. 3.5.

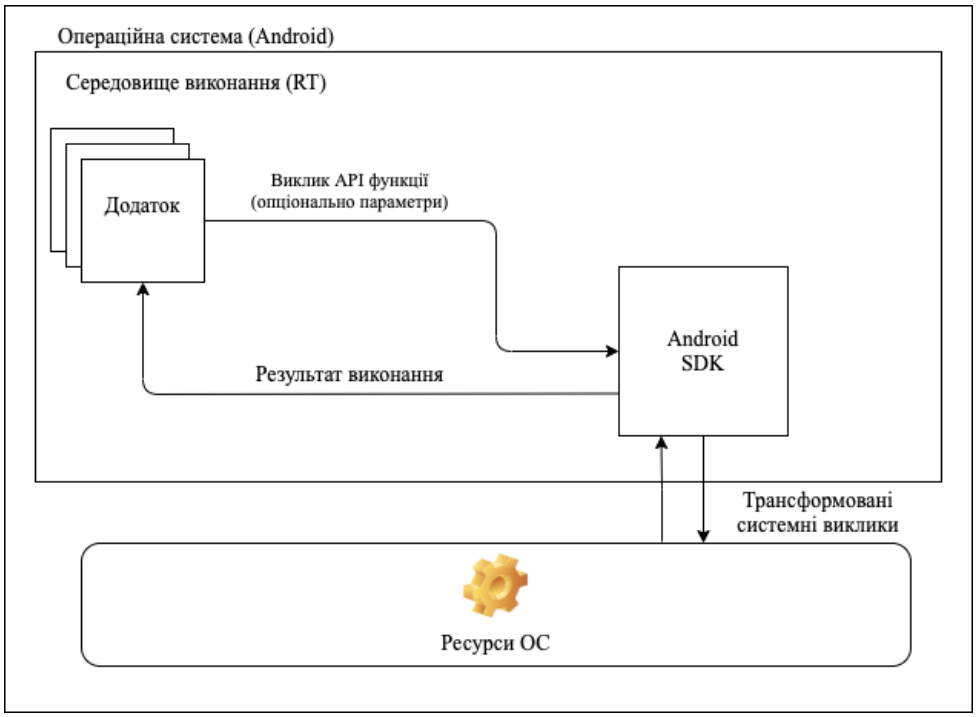


Рисунок 3.5 – Схема комунікації мобільного додатку з ресурсами ОС через виклик API функції

Одним із основних представників динамічного функціонального трейсингу є Frida Framework [133]. І цьому фреймворку динамічний спосіб перехоплення комунікації з ресурсами операційної системи реалізується шляхом вбудовування фрагментів коду, написаних мовою JavaScript, у будь-які програмні додатки, для Windows, Mac, Linux, iOS та Android. Frida також надає набір простих інструментів, побудованих поверх основного ядра, які можуть бути використані як напряму або налаштовані відповідно до потреб.

До складу програмного ядра Frida входять такі компоненти:

- **Frida-CLI** – це циклічний інтерфейс зчитування, аналізу та друку (Read-Evaluate-Print-Loop – REPL), який має на меті наслідувати багато особливостей

програмних мов Python або Cycript, використання яких допомагає швидкому вбудовуванню власного коду, створенню прототипів та легкому налагодженню;

- **Frida-ps** – це інструмент командного рядка, що корисно при взаємодії з віддаленою системою;

- **Frida-trace** – це інструмент для динамічного відстеження викликів API функцій;

- **Frida-Discover** – це інструмент для виявлення внутрішніх функцій програми, які потім можна простежити за допомогою програмного інструменту Frida-trace.

Програмний комплекс Frida написано мовою C з можливістю вбудовувати цільові процеси, де власний код, написаний на JavaScript (JS), виконується з повним доступом до пам'яті, підключенням функцій і навіть викликом власних функцій усередині процесу. Існує двонаправлений канал зв'язку, який використовується для спілкування між додатком та JS, що працює в цільовому процесі. Поверх ядра Frida core є кілька прив'язок, написаних іншими мовами, наприклад, Python, Node.js, .NET, Qml тощо.

Frida складається з двох компонентів: клієнт і сервер, які взаємодіють між собою через два порти TCP 27042 і 27043. Клієнта можна встановити, просто запустивши команду «pip» (фреймворк для мови програмування python) – pip install frida. Для інсталяції потрібні права адміністратора (root) у середовищі Windows або Unix. Ядро Frida пише код безпосередньо в пам'ять процесу. Тобто коли вбудований код під'єднується до запущеної програми у фоновому режимі, система використовує Frida-trace для перехоплення потоку. Завантажувач перехоплює основний потік і запускає новий, підключаючись до сервера (frida-server), який працює на пристрої, і завантажує динамічно створену бібліотеку, що містить агент (frida-agent) разом із вбудованим кодом. Перехоплений потік переходить до початкового стану та відновлюється. Frida декорує системний та функціональний API виклик в обгортку, яка дає можливість виконувати власний

код, підмінити реалізацію функції або модифікувати результат виконання на будь-який інший.

Функціональна схема перехоплення та декорування функціонального API виклику Android SDK з використанням Frida представлена на рис. 3.6.

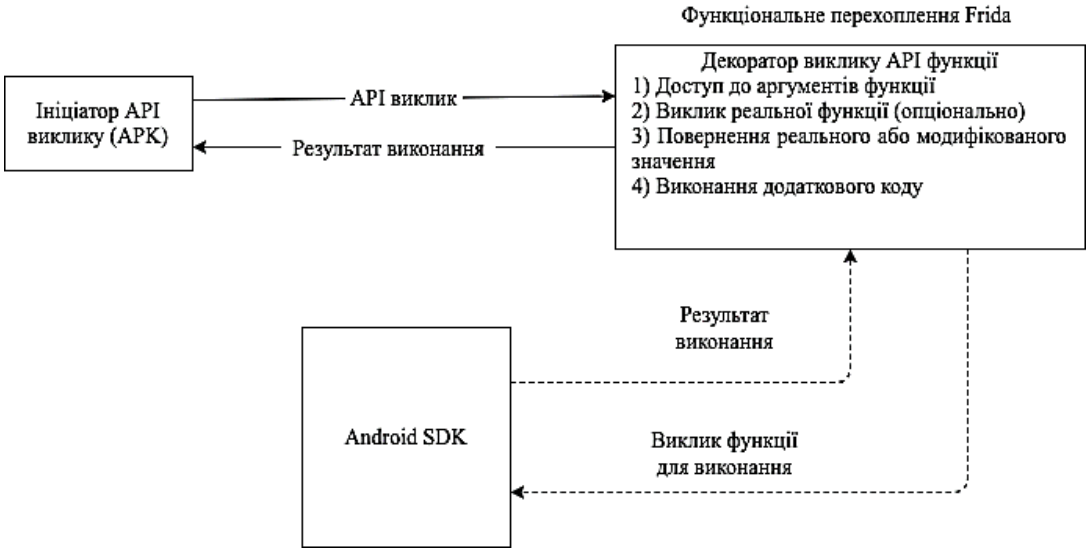


Рисунок 3.6 – Функціональна схема перехоплення API функції

Модуль функціонального перехоплення являє собою вставку коду, написаного мовою JavaScript, який накопичує API виклики на файловій системі в текстовому вигляді та асоціює їх із мобільним додатком [134]. У свою чергу, існують API функції, які надають доступ до обмежених даних або ресурсів мобільного пристрою, які дуже часто трапляються в списку Malware Project та можуть безпосередньо погрожувати функціональній безпеці. Зокрема можна виділити такі варіанти використання API:

- мобільний додаток використовує API, яке необхідне для доступу до конфіденційних даних: getUniqueIMEI, getUSIMnumber, getDeviceId, getSimSerialNumber, getImei або getSubscriberId;
- виклик API, який використовує комунікацію через мережу: setWifiEnabled, prepareHttpClient, execHttpRequest;
- виклик API для надсилання та отримання SMS / MMS-повідомлень, таких як sendTextMessage, subscribeToSmsEvents, SendBroadcast, sendDataMessage.

- API функції, які можуть мати доступ до поточного або попереднього місцезнаходження: getLastKnownLocation, getLatitude, getLongitude, requestLocationUpdates;

- виклики API функцій для виконання зовнішніх або окремих команд, таких як Runtime.exec() та Ljava.lang.runtime.exec().

Для того щоб детально проаналізувати процес побудови СФЛД, необхідно розглянути вихідний код програми під час виконання, а також трейс-файл, який створюється під час динамічного аналізу мобільного додатку. Для прикладу розглянемо код методу ”getPhoneNumber()” всередині MainActivity – основного візуального компенента ОС Android (рис. 3.7).

```
class TestInterceptor {

    private val perm = ""

    @SuppressLint( ...value: "HardwareIds", "MissingPermission")
    fun getPhoneNumber(ctx: Context) : String {
        val tm = ctx.getSystemService(Context.TELEPHONY_SERVICE) as TelephonyManager
        var phoneIdentifiedData = "OpName: ${tm.simOperator}; NetOpName: ${tm.networkOperatorName}"


        if (ActivityCompat.checkSelfPermission(ctx as Activity, perm) != PackageManager.PERMISSION_GRANTED)
            return ""

        phoneIdentifiedData += "Num: ${tm.line1Number}"
        phoneIdentifiedData += "Country: ${tm.simCountryIso}"
        phoneIdentifiedData += "DevId: ${tm.deviceId}"


        return phoneIdentifiedData
    }
```

Рисунок 3.7 – Вихідний код методу getPhoneNumber на мові Kotlin

Наведений вище приклад демонструє процес отримання поточних телефонних метаданих, зокрема, ідентифікатор сім карти, перший рядок телефонного номера (зазвичай мобільний номер користувача), країну за іменованим стандартом ISO та унікальний ідентифікатор мобільного пристрою (насправді багато сучасних пристроїв намагаються приховати унікальний ідентифікатор, і в цьому випадку цей метод просто поверне null).

На рис. 3.8 наведено розроблений скрипт перехоплення API виклику, який складається з двох мов: перша частина написана на Python та використовується

як міст для виконання скрипту inject.js; саме перехоплення написано мовою JavaScript, і ця частина коду вбудовується в цільовий процес.

```
import frida , sys , time
def on_message(message , data): if message['type'] == 'send':
print("[TRACE_1] {0}".format(message['payload'])) else:
print(message) jscode = """
Java.perform(function () {
// Функція перехоплення визначається тут
var MainActivity= Java.use('edu.interceptor.android.MainActivity');
// Після старту кванту Q, коли додаток почне виконання
MainActivity.getPhoneNumber.implementation = function () {
// Показуємо повідомлення для ідентифікації початку трейсингу
send('getPhone');
// Виклик реалізації оригінального методу слухача onClickListener
var output = this.getPhone();
send(output);
// Змінюємо значення яке повертається з методу getPhoneNumber
return output+"INTERCEPTED"; };
}); """

device = frida.get_usb_device()
pid = device.spawn(['edu.interceptor.android.hello_world'])
process = device.attach(pid)
script = process.create_script(jscode)
script.on('message', on_message) print('[Trace_2] Running TEST')
script.load()
device.resume(pid)
sys.stdin.read()
```

Рисунок 3.8 – Розроблений код скрипта Frida.inject для перехоплення API виклику

Сценарій виконання складається з таких кроків:

1. З'єднання з ADB (android device bridge) в рядку "device = frida.get usb device ()".

2. Завантаження процесу для підготовки вставки коду (процес призупиняється та потребує відновлення).

3. Під'єднання коду до процесу, створюється шаблон коду JavaScript та додається у змінну jscode.

4. Реєстр функції зворотного виклику для отримання повідомлень від гостьового користувача (Guest User – використовується термінологія антиемуляторної системи CuckoDroid).

5. Завантажуються скрипт та відновлюється виконання програми.

6. Додається STDIN для зупинки додатка від закінчення виконання.

Код JavaScript робить наступне:

1. Відкриває MainActivity за допомогою Java.use.

2. Підключає метод getPhoneNumber, надсилає його вихідне значення та модифікуємо значення, що повертається з функції.

На рис. 3.9 наведений трейс-файл функції getPhoneNumber, який детально показує послідовність викликів додатку «com.interceptor.android» та додаткові системні виклики, що супроводжують виконання додатку APK в рамках операційної системи Android.

Рисунок 3.9 – Трейс-файл функції getPhoneNumber

Повідомлення надіслані хосту (frida-серверу), включаючи підтвердження, що метод getPhoneNumber під'єднався успішно, посилаючи назад рядок "getPhoneNumber" і повертаючи результат виконання цього методу. Необхідно зазначити, що в рамках даної роботи Frida скрипт модифікує початковий APK файл шляхом декорування API викликів лише на модифікованому пристрої емуляції з повним доступом привілейованого користувача [134]. Існують способи запустити та виконати скрипт в рамках привілеїв звичайного користувача, але це вимагає додаткових зусиль, які лежать поза межами даного дослідження.

Зміст фрагменту файлу включає інформацію про трасування функції getPhoneNumber, яка зберігається в тимчасовій пісочниці файлової системи (рядок «edu.interceptor.android.MainActivity->getPhoneNumber:…INTERCEPTED»).

Також можна помітити системні виклики ОС Android, а саме: android.app.Activity, android.app.Instrumentation, android.app.ActivityThread, android.os.Handler, android.os.Looper. Такі системні виклики, як правило, створюють додатковий шум при трасуванні будь-якого додатку під час динамічного аналізу, що призводить до необхідності додаткової обробки вхідних послідовностей перехоплених СФЛД.

3.5 Архітектура розробленого програмного комплексу

Для реалізації запропонованого методу виявлення шкідливих додатків було розроблено програмний комплекс у складі двох програмних засобів AMalDetector та CMMD. Схема використання програмного комплексу представлена на рис. 3.10. Програмний засіб AMalDetector представляє собою додаток модуля функціональної безпеки, забезпечує комунікацію з серверним веб додатком CMMD на мобільному пристрої. Він являє собою клієнтську частину комплексу функціональної безпеки, яка виконується на мобільному пристрої та базується на декоруванні функціоналу програмного додатка за допомогою Frida.

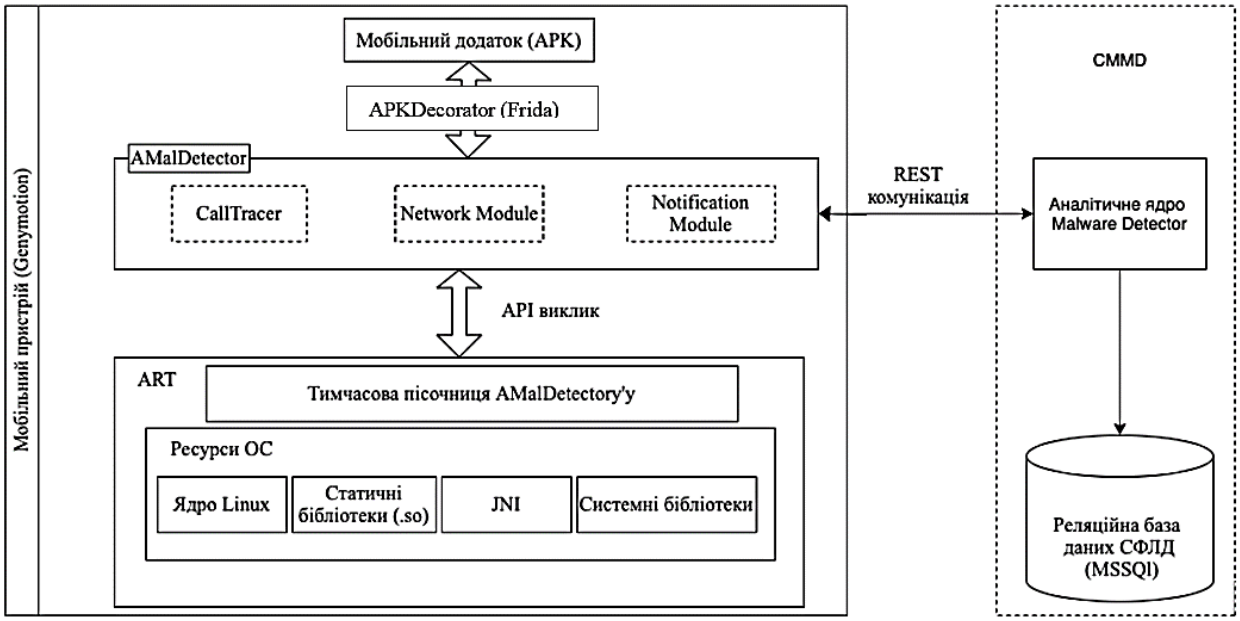


Рисунок 3.10 –Схема використання програмного комплексу

Архітектура програмного засобу AMalDetector складається з трьох модулів: CallTracer, NetworkModule, NotificationModule. APK додатка ініціює API виклики, які перехоплюються Frida і передаються на AMalDetector. APKDecorator є скрипт, який вбудовується в Frida. Він написаний на Python та JavaScript і виступає функцією зворотного зв'язку між потенційно небезпечним додатком та модулем CallTracer. Модуль CallTracer використовує тимчасову пісочницю в середовищі виконання ART (Android RunTime) [135] для серіалізації СФЛД додатка у файл. У поточній реалізації цей модуль також використовується для видалення тимчасових файлів, якщо аналізуємий додаток не є шкідливим.

Модуль AMalDetector в реальному часі сканує додаток та фрагментує API функції. На рис. 3.11 наведено трейс-файл шкідливого додатку.

NetworkModule використовує мережеве програмне забезпечення операційної системи (Network API Android SDK) для побудови та надсилання запиту на сервер CMMD, використовуючи REST сервіси CMMD. NotificationModule формує текстову нотифікацію користувача про потенційну

небезпеку шляхом візуалізації значень атрибутів вектора атаки та отриманих результатів аналізу СФЛД.

```
1  .class public Ledu/test/MonitoringSysCall
2  method internal static log saveFragment(...args)
3  const string p1 "packageName: edu.interceptor.logE(...args)"
4  static invoke {v0}, appendFragmentedCall(..args)
5  Unit return
6  .endMethod
7
8  .method public static sendAttempt(...args)
9  newInstance {v1} Ljava.lang.Thread
10 .newInstance {v2} okHttpClient.invoke(...args)
11 .method okHttpClient.retryPolicy(...args)
12 .method public static localCleanup(..args)
13 Unit return
```

Рисунок 3.11 – Трейс-файл шкідливого додатка

Програмний засіб CMMD реалізований у вигляді веб-додатку як Azure-сервіс, налаштування якого представлено на рис. 3.12.

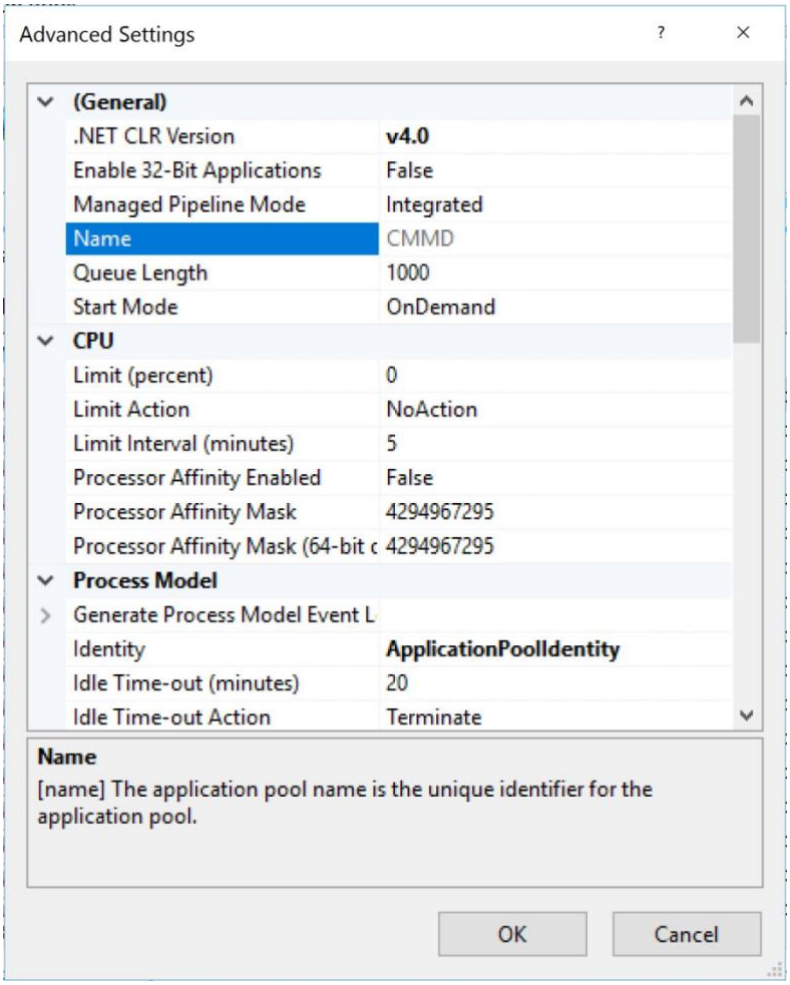


Рисунок 3.12 – Налаштування Azure-сервісу CMMD

CMMD призначений для визначення ідентичності аналізованої та шаблонної СФЛД за допомогою розробленого методу динамічного виявлення потенційно небезпечних додатків, включаючи конкатенацію фрагментованих наборів API функцій, упорядкування шаблонних СФЛД та послідовне використання методів локального та глобального вирівнювання. За результатами порівняння СФЛД сервер CMMD надсилає об’єкт-відповідь із значеннями атрибутів вектора атаки для нотифікації користувача.

Нотифікація користувача про факт виявлення потенційно шкідливого додатку відбувається за допомогою одного із фонових сервісів (WorkManager) модулю NotificationModule, який аналізує об’єкт-відповідь серверного засобу CMMD. Приклад нотифікації користувача про ідентифікацію шкідливого додатка наведено на рис. 3.13.

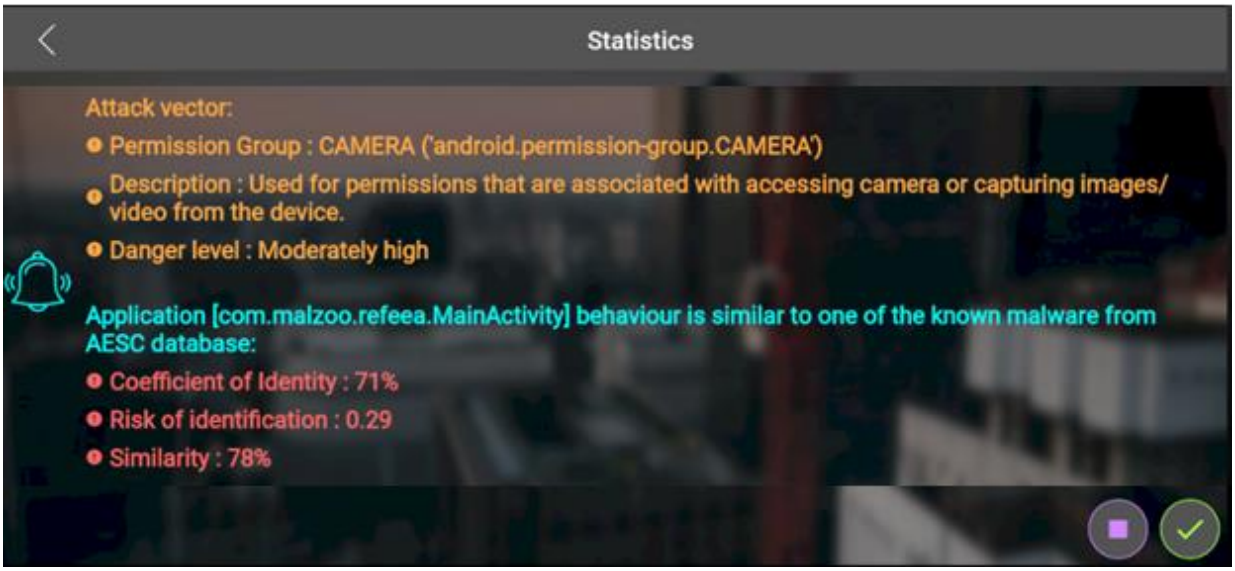


Рисунок 3.13 – Приклад нотифікації користувача

Слід зауважити, незважаючи на те що СФЛД представляє собою послідовність API функцій, рівень небезпеки, який виводиься при нотифікації користувача, обумовлюється найнебезпечнішим дозволом в перехопленій послідовності.

UNL-діаграми класів розроблених програмних засобів AMalDetector та CMMD наведені в додатку В.

Однією із функцій NotificationModule є зупинка додатку по запиту користувача. Подібна функція можлива лише для користувачів з підвищеними привілеями (root), за допомогою команди [136]

adb shell am kill "MALWARE_PACKAGE_NAME".

Однак необхідно враховувати, що зупинка цільового процесу в сучасних версіях ОС Android у більшості випадків достатньо складна задача, оскільки кожна програма працює у власному процесі. Процес додатку залишатиметься запущеним до тих пір, поки операційна система потребуватиме відновлення пам'яті для використання в інших застосунках.

3.6 Взаємодія клієнтського модуля комплексу із сервером

Побудова запиту клієнтського модуля NetworkModule виконується в декілька етапів, які детально розглядаються нижче. Більшість підключених до мережі програм використовують HTTP для надсилання та отримання даних. Платформа Android включає клієнт HttpsURLConnection, який підтримує протокол TLS для шифрування даних, потокове завантаження та завантаження з налаштуваннями тайм-аутів, IPv6 та пул з'єднань.

Переважно сучасні додатки використовують перевірені бібліотеки, які підтримуються Google або її партнерами. При реалізації NetworkModule була використана бібліотека Retrofit, яка, у свою чергу, базується на низькорівневій бібліотеці OkHttp. Дана біліотека дозволяє декларативно описати комунікацію із сервером. Retrofit також підтримує автоматичну серіалізацію об'єктів, запитів та десеріалізацію об'єктів відповідей.

Реалізація власного мережевого модуля являє собою достатньо складне завдання, яке включає в себе підтримку стандарту REST, та обробку граничних ситуацій, які можуть виникнути в будь-який момент: як при підключенні до сервера, так і під час серіалізації результату запиту. Перш ніж додавати мережеву функціональність до своєї програми, потрібно переконатися, що дані та інформація у програмі залишаються в безпеці під час передачі їх через

мережу. Для дотримання цих умов використовувались такі практичні заходи щодо безпеки мережі:

- кількість конфіденційних або особистих даних користувачів, які передаються мережею, було максимально мінімізовано;

- весь мережевий трафік із модуля NetworkModule програмного комплексу AMalDetector пересилається через криптографічний протокол SSL (Security Socket Layer).

Щоб спростити процес виконання мережевих операцій та зменшити дублювання коду в різних частинах програми, використовувалась декомпозиція архітектури AMalDetector на шари, в яких використовували шаблон дизайну (сховище) під назвою Repository, та шар необхідних сервісів. Repository – це клас, який обробляє операції з даними та забезпечує чисту абстракцію API сервера, який викликається програмним додатком для доступу до певних даних або ресурсів ОС Android. На рисунку 3.14 зображений виклик analyzeApiChainAsync інтерфейсу IRetrofitAmalDetectorApi, який виконує клієнтський модуль NetworkModule.

```
private const val api = "AMalDetectorApi"

interface IRetrofitAmalDetectorApi {

    @POST( value: "$api/{token}")
    suspend fun analyzeApiChainAsync(@Body pro: AnalyzeApiChainPro): CallChainInfoNet?
```

Рисунок 3.14 – Виклик API сервісу для аналізу фрагментованої СФЛД

Щоб уникнути ситуації, коли будь-який додаток просто не відповідає на запити користувача, мережеві операції в основному потоці виконувати не можна. За замовчуванням Android вимагає виконання мережевих операцій із потоком, відмінним від основного потоку інтерфейсу користувача. Якщо цього не буде зроблено, може виникнути виключна ситуація NetworkOnMainThreadException [137]. Щоб уникнути цього використовується модифікатор suspend, який сигналізую компілятору, що цей метод необхідно викликати асинхронно, що на

програмному рівні реалізовується за допомогою state-machine. Таким чином не зупиняється основний потік мобільного пристрою. Вхідний параметр pro (від англ. – Per Request Object) має в собі агрегацію фрагментованих СФЛД, яку сервер використовує для порівняння з шаблонними послідовностями API викликів шкідливого ПЗ. Параметр CallChainInfoNet, який повертається, після десеріалізації містить інформацію щодо рівня шкідливості аналізованого програмного додатка. На рисунку 3.15 показано базові поля класу-відповіді модулю CMMD.

```
data class AnalyzeApiChainPro(
    @field:JsonProperty( value: "identityLevel") val identityLevel: Float,
    @field:JsonProperty( value: "similarityLevel") val similarityLevel: Float,
    @field:JsonProperty( value: "cleanup") val cleanupRequired: Boolean,
    @field:JsonProperty( value: "riskDescription") val risk: String,
    @field:JsonProperty( value: "attackVector") val attackVector: String,
    @field:JsonProperty( value: "initialAppToken") val token: String,
)
```

Рисунок 3.15 – Базовий клас-відповідь після аналізу СФЛД

Клієнтський модуль NotificationModule аналізує результат запиту та перевіряє стан мітки cleanup. Якщо встановлене значення true, то це означає, що сервер видалив усі метадані програмного додатку. У такому разі AMalDetector видаляє всі файли трасування з пісочниці. Нижче наведено детальне пояснення всіх полів результуючого об’єкта відповіді модуля CMMD:

- IdentityLevel – коефіцієнт ідентичності аналізуємої СФЛД з послідовністю (послідовностіми) із бази шаблонних API викликів серверного модуля CMMD;

- SimilarityLevel – коефіцієнт подібності аналізованої СФЛД; якщо порівнюваний API виклик не є ідентичним із шаблонним СФЛД викликом у певній позиції, але виклик належмть одній групі привілеїв, що і шаблонний API виклик, цей коефіцієнт збільшується, оскільки є велика вірогідність того, що вектор атаки збігається зі шкідливим ПЗ;

- Cleanup – якщо флаг має значення true, це сигналізує, що збіг не знайдено й необхідно видалити всі локальні трейс-файли з клієнтського боку для

заданого додатку; сервер повертає це значення тільки у випадку, коли не знайдено збігу для фрагментованої СФЛД;

- RiskDescription – базовий опис; якщо є якісь відомості про шкідливий APK в репозиторії СФЛД, але на жаль основні бази шаблонних СФЛД (Malgenome та Drebin) не мають ніяких відомостей про сам шкідливий додаток, тому у поточній реалізації комплексу додатково використовуються репозиторій програмних додатків AndroidZoo, який надає необхідні дані про APK;

- AttackVector – автоматично згенерований опис атаки шкідливого ПЗ, базуючись на псевдосимволі СФЛД.

- InitialAppToken – унікальний ідентифікатор аналізуємого мобільного додатку; використовується для ідентифікації трейс-файлу модулем CallTracer; серверний модуль CMMD використовує його для агрегації фрагментованих СФЛД.

3.7 Висновки до розділу 3

1. Обгрунтовано формування середовища дослідження динаміки поведінки програмних додатків з використанням програмного емулятора ОС Android. Враховуючи здатність шкідливого ПЗ не проявляти себе в умовах емуляційного віртуального середовища запропоновано використання набору антиемуляційних заходів для підвищення вірогідності виявлення шкідливих додатків.

2. Проаналізовані сучасні засоби трасування мобільних додатків та особливості використання емуляторів ОС Android при побудові та дослідженні СФЛД. Обгрунтовано вибір фрейсворка Frida як основного засобу отримання даних про динаміку викліків API функцій програмним додатком. Розроблена функціональна схема перехоплення API функції та скріпт перехоплення API виклику.

3. Враховуючи можливість додатків здійснювати виклики API функцій при багатопотоковому виконанні, що значно ускладнює їх трасування,

запропоновано рекурсивний алгоритм для обробки розгалужених послідовностей, який забезпечує нормалізацію послідовностей викликів при наявності потоків.

4. Визначено набір технологій, якими необхідно володіти при реалізації програмного комплексу для забезпечення моніторингу та аналізу фрагментованих СФЛД. Проаналізовано сучасні способи та особливості комунікації мобільного ПЗ з Android SDK, в тому числі системні виклики на рівні JNI та звичайні виклики API функцій.

5. Розроблена архітектура програмного комплексу забезпечення функціональної безпеки ресурсами ОС Android у складі двох програмних засобів AMalDetector та CMMD. Описані основні модулі програмних засобів, способи їх комунікації між собою та інтеграції з операційною системою.

6. Детально розглянуто процес декорування API викликів, фрагментація СФЛД у трейс-файлах та процес побудови серверного запиту мережевим модулем NetworkModule. Продемонстровано процес перехоплення програмної функції, написаної мовою Kotlin, за допомогою вставки власного коду JavaScript в процес виконання програмного додатка. Обґрунтовано використання сторонньої бібліотеки Retrofit на базі OkHttp для запитів за протоколом HTTPS та комунікації клієнтського модуля AMalDetector із сервером CMMD.

7. Розроблений графічний інтерфейс, який дозволяє сповістити користувача про виявлений шкідливий додаток, демонструючи детальну інформацію про можливий вектор атаки, коефіцієнт ідентичності та подібності, ризик виявлення шкідливого додатка.

8. Розроблений програмний комплекс повною мірою реалізує можливості запропонованої інформаційної технології, використовуючи ефективні рішення та методи передачі даних, моделювання та програмування для оцінки шкідливості аналізуємих програмних додатків у операційній системі Android.

Результати досліджень, приведених в розділі, опубліковані в роботах [98, 114, 134].

РОЗДІЛ 4
РОЗРОБЛЕНА ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ТА ЇЇ ДОСЛІДЖЕННЯ

4.1 Інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв

Схема розробленої інформаційної технології зображена на рис. 4.1.

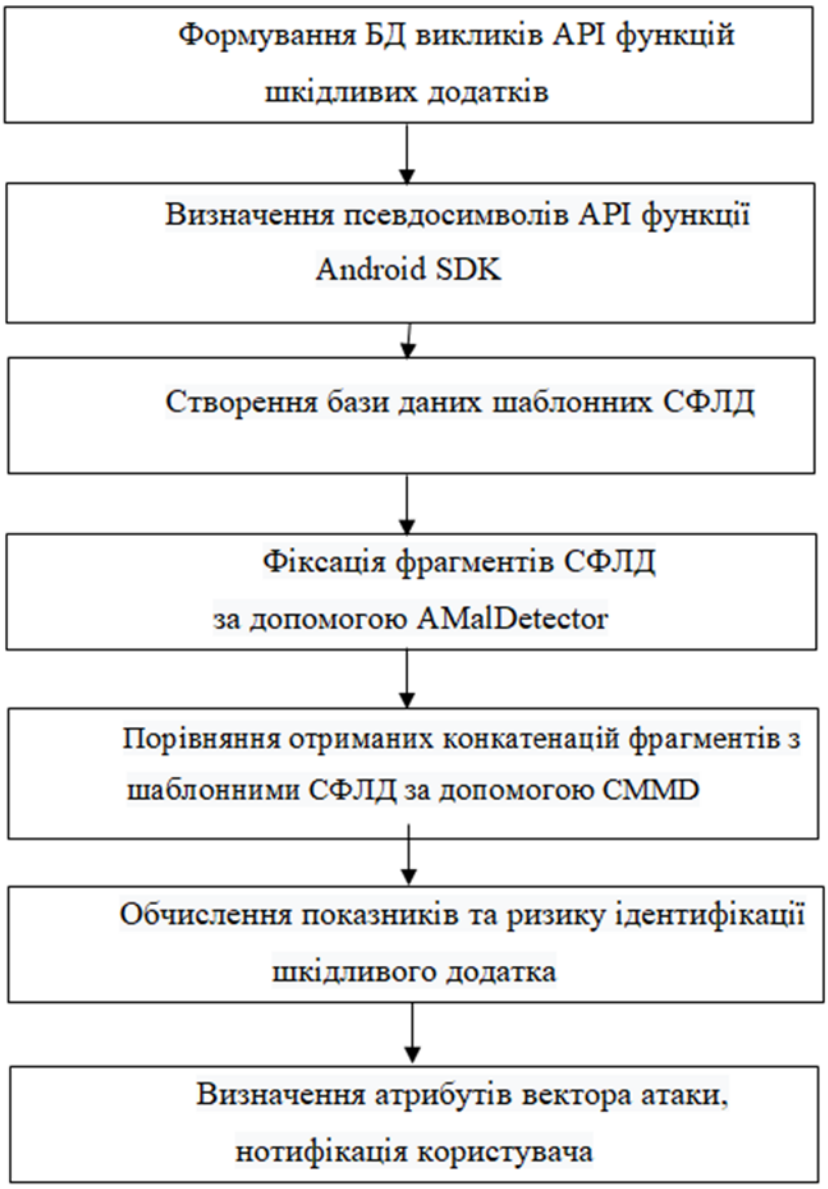


Рисунок 4.1 – Схема розробленої інформаційної технології

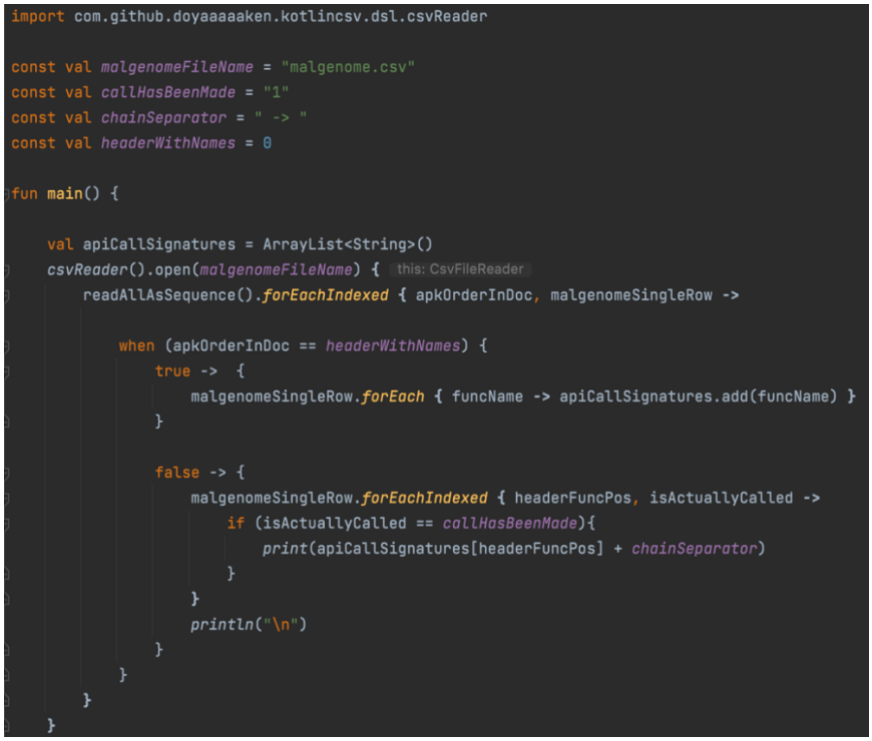
Формування БД викликів API функцій шкідливих додатків відбувається на основі даних дослідницьких проєктів, що надають дані про шкідливі додатки ОС Android.

В рамках даної роботи була використана база даних шкідливого ПЗ Malgenome, але розроблену технологію можна застосувати і при використанні, наприклад, бази даних Drebin [39]. Оскільки бази даних, як правило, надають інформацію у вигляді файлів «.csv» (рис. 2.2), була розроблена функція розкладаня даного файлу на ланцюжки, які містять API функції, що викликаються певним додатком (помічені «1» у відповідних стовпчиках).

Блок-схема алгоритму функції розкладання вхідного .csv фвйла приведена на рис. 4.2.

Рисунок 4.2 – Блок-схема алгоритму функції розкладання вхідного .csv фвйла

Код реалізації даної функції представлений на рис. 4.3.



```
import com.github.doyaaaaaken.kotlincsv.dsl.csvReader

const val malgenomeFileName = "malgenome.csv"
const val callHasBeenMade = "1"
const val chainSeparator = " -> "
const val headerWithNames = 0

fun main() {

    val apiCallSignatures = ArrayList<String>()
    csvReader().open(malgenomeFileName) {  this: CsvFileReader
        readAllAsSequence().forEachIndexed { apkOrderInDoc, malgenomeSingleRow ->

            when (apkOrderInDoc == headerWithNames) {
                true ->  {
                    malgenomeSingleRow.forEach { funcName -> apiCallSignatures.add(funcName) }
                }

                false -> {
                    malgenomeSingleRow.forEachIndexed { headerFuncPos, isActuallyCalled ->
                        if (isActuallyCalled == callHasBeenMade){
                            print(apiCallSignatures[headerFuncPos] + chainSeparator)
                        }
                    }
                    println("\n")
                }
            }
        }
    }
```

Рисунок 4.3 – Код функції розкладання файлу .csv бази даних Malgenome

Після побудови бази шаблонних шкідливих додатків та їхніх метаданих (табл. 2.3 – 2.5) передусім виникає проблема продуктивного пошуку в великому масиві даних. Дану проблему потрібно вирішувати комплексно, використовуючи ефективні алгоритми пошуку. Нині є багато способів підвищити продуктивність пошуку строкового рядку в базі даних, як, наприклад, Full Text Search (FTS) [138]. У межах цієї роботи було використано індексування стовпчика з псевдосимволом відповідного виклику, що дало значне покращення при строковому пошуку у базі даних. Якщо проаналізувати частоту запису та зчитування інформації з шаблонної бази даних, то, очевидно, що кількість зчитування значно перевищить кількість запису, тому можна знехтувати втратами продуктивності при додаванні нової інформації.

По результатах аналізу БД Malgenome були встановлені такі значення характеристик шаблонних СФЛД: середня довжина - 37, максимальна - 112.

Далі API функції кодуються псевдосимволами, з використанням яких будуються шаблонні СФЛД. Визначення псевдосимволів відбувається на

ініціалізаційному етапі формування шаблонних СФЛД після екстракції ланцюжків з файлу “Malgenome”. Слід зазначити що процес формування БД викликів API функцій шкідливих додатків та визначення псевдосимволів API функції Android SDK доволі ресурсномістке завдання, так як вхідні послідовності проходять декілька трансформацій, але виконується один раз на початковій фазі підготовки серверного модулю CMMD.

Клієнтський модуль AMalDetector фіксує фрагменти сигнатури викликів протягом часового інтервалу q. При отриманні чергового фрагмента аналізуємої СФЛД відбувається конкатенація фрагментованих послідовностей [98]. Схематичне зображення сиквенційної агрегації наведено на рис. 4.4.

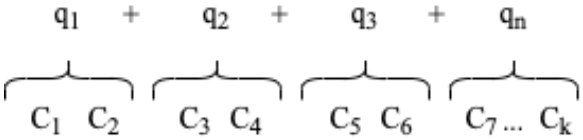


Рисунок 4.4 – Схематичне зображення конкатенації фрагментованої СФЛД по часовому кванту q

За фіксацією фрагментів СФЛД відповідає клієнтський модуль AMalDetector, також за допомогою цього модулю виконується відправка фрагменту СФЛД з усіма метаданими на серверний модуль CMMD для подальшого аналізу. Обчислення базових показників ідентичності та схожості відбувається за допомогою комбінації локального та глобального вирівнювання СФЛД, які детально описані в п. 2.4.

Серверний модуль CMMD обробляє вхідні послідовності API функцій та за допомогою запропонованого методу порівнює отримані фрагменти API функцій з шаблонними СФЛД з БД [98]. В разі наявності збігу аналізуємої СФЛД із шаблонами, присутніми в вихідній БД, формується вектор атаки, який визначає напрям атаки (група дозволів), характер атаки (тобто який саме дозвіл потребує API функція) та рівень небезпеки.

Фінальним результатом роботи інформаційної технлогії є обчислення ризику ідентифікації шкідливого додатку на основі коефіцієнта ідентичності та

нотифікація користувача шляхом візуалізації атрибутів вектора атаки. У разі, якщо показник ідентичності перевищує порогове значення, здійснюється користувачу видається повідомлення про загрозу функціональній безпеці з відповідним ризиком, в іншому випадку метадані по аналізуємому додатку видаляються.

4.2 Матриця невідповідності

При проведенні експериментів для перевірки та оцінювання розробленої технології біла використана матриця невідповідностей задач класифікації. Результати тестів, представлені в даній матриці, можуть бути істинно позитивними (True Positive), істинно негативними (True Negative), хибно негативними (False Negative) та хибно позитивними (False Positive) [139]. Якщо в результаті тесту доведено наявність позитивного результату, результат діагностичного тесту вважається істинно позитивним (TP). Якщо ж доведено відсутність позитивного результату, результат тесту є істинно негативним (TN).

Істинно позитивний та істинно негативний тест свідчать про несуперечливий результат між діагностичним тестом і доведеним станом, який також називають стандартом істинності.

Однак жоден тест не є ідеальним. якщо діагностичний тест вказує на наявність негативного результату, який насправді не є таким, то результат вважаэться хибно позитивним (FP). Таким чином, якщо результат діагностичного тесту свідчить про відсутність позитивного результату, коли він при реально позитивний, то результат тесту є помилково негативним (FN). Як хибно позитивні, так і хибно негативні результати свідчать про те, що ці результати протилежні фактично отриманому стану. Матриця невідповідностей для бінарної класифікації приведена на рис. 4.5.

Результат тесту	Справжній стан	
	Позитивний	Негативний
Позитивний	Істинно позитивний (TP)	Хибно позитивний (FP)
Негативний'	Хибно негативний	Істинно негативний

Рисунок 4.5 – Матриця невідповідностей бінарного класифікатора

Матриця невідповідностей в рамках даної роботи використовувалась для того, щоб краще зрозуміти сильні та слабкі сторони розробленої технології та порівняти результати її роботи з вже існуючими альтернативними підходами та реалізаціями [44]. Оцінка ефективності розробленої інформаційної технології відбувалась з використанням відомих метрик [140]:

$$TPR = \frac{\text{TP}}{\text{TP+FN}} = Recall \tag{4.1}$$

$$FPR = \frac{\text{FP}}{\text{FP+TN}} \tag{4.2}$$

$$Precision = \frac{\text{TP}}{\text{TP+FP}} \tag{4.3}$$

$$\text{F} - \text{score} = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall+Precision}} \tag{4.4}$$

Для обчислення істино позитивної (*True Positive Rate – TPR*, або *Recall - повнота*) та істино негативної (*False Positive Rate - FPR*) метрик класифікації використовувалися такі елементи матриці невідповідності, а саме: *TP (true positive)/TN (true negative)* – число правильно ідентифікованих шкідливих/нешкідливих додатків, та *FP(false positive)/FN (false negative)* – число неправильно ідентифікованих шкідливих/нешкідливих додатків.

Щодо інших метрик, то *Precision (влучність)* – це відношення правильно ідентифікованих шкідливих додатків до загального числа шкідливих додатків, а *F-score* є середнім гармонійним влучності та повноти і дає їх рівноважну (у даному випадку це коефіцієнт 2 у формулі 4.4) агреговану оцінку, яка часто використовуються як сукупний показник ефективності.

З вище представлених формул можемо зробити висновок, що істинно позитивний показник - це частка істинно позитивних результатів, які правильно визначаються за допомогою діагностичних тестів. TPR показує, наскільки ефективним є тест при виявленні шкідливого ПЗ: чим вище числове значення TPR, тим менш вірогідно, що діагностичний тест має хибно позитивний результат. Наприклад, якщо значення метрики істинно позитивних результатів дорівнює 99%, це свідчить про те, що в період проведення діагностичного тесту на аналізуємому APK з наявністю шкідливого коду, існує 99%, що даний додаток буде визнаний як позитивний. Таким чином, діагностичний тест з високим показником метрики TPR буде доцільно використовувати для бінарної класифікації (в рамках даної роботи це шкідливе ПЗ на ОС Android).

4.3 Експерементальна оцінка ефективності за основними показниками

При тестуванні інформаційної технології було проведено 3 експерименти для різних наборів програмних додатків, представлених APK файлами, які були отримані із вже зазначених раніше репозиторіїв шкідливих додатків та з Google Play Store. Експерименти охоплювали три варіанти комбінацій множин шкідливих та нешкідливих додатків. Кожен експеримент характеризувався двома основними параметрами: сумарна кількість додатків та кількість шкідливого програмного забезпечення в межах цього експерименту. Порогове значення визначення шкідливого додатку становило 70%. Результати експериментів представлені в табл. 4.1.

Таблиця 4.1 - Результати експериментів з виявлення шкідливих додатків

№ Експ.	Кількість APK	Malware (%)	TPR (%)	FPR (%)	Влучність	Повнота	F-score
1	100	94	99	6.7	0.96	0.99	0.97
2	250	85	99	5.4	0.99	0.99	0.99
3	500	70	99	1.3	0.99	0.99	0.99

Отримані результати показують, що збільшення числа нешкідливих додатків може призвести до значного зменшенням значення FPR. Цей факт підтверджує правильність гіпотези про те, що ланцюжки СФЛД шкідливих і нешкідливих додатків суттєво відрізняються. Таким чином, запропонований метод динамічного виявлення шкідливих додатків шляхом порівняння СФЛД здатен ефективно вирішувати задачу ідентифікації. При використанні додатків із перевірених офіційних джерел цей метод буде коректно їх класифікувати. Більшість сучасних онлайн-магазинів мобільного програмного забезпечення, включаючи Google Play Store, являють собою середовище, в якому значною мірою переважають нешкідливі програмні додатки, що збігається з вхідними даними експерименту номер 3 із найнижчим показником FPR.

Результати експериментів були порівняні з відомими результатами [141], отриманими за допомогою алгоритмів машинного навчання (MLA – Machine Learning Algorithm), таких як K-найближчий сусід (KNN – K-nearest neighbours) та багатошаровий персептрон (MLP – multilayer perceptron). У табл. 2 наведено дані порівняння методів виявлення шкідливих додатків для вибірки об’ємом в 1100 APK, з яких 100 – шкідливі додатки, відібрані випадковим чином з БД Android Malgenome Project, а 1000 APK нешкідливих додатків отримані з Google Play Store. У табл. 2 запропонований метод позначений як СФЛД.

Таблиця 4.2 – Результати порівняння методів виявлення шкідливих додатків

Метод	TPR (%)	FPR (%)	Влучність	Повнота	F-score
MLP	93.03	6.97	0.93	0.972	0.944
KNN	98.63	1.37	0.98	0.986	0.986
СФЛД	**99.10**	**0.20**	**0.99**	**0.991**	**0.990**

У табл. 4.2 наведені усереднені дані щодо TPR=0,991 для запропонованого методу з використанням СФЛД за результатами 10 експериментів із випадковою вибіркою по 100 базових СФЛД шкідливих додатків (точність результату становить 0,007 для довірчої ймовірності 0,95).

MLA забезпечує достатньо хороше значення TPR, але з досить високим FPR, який погіршує загальний показник F-score, що робить використання цього методу обмеженим у змішаних середовищах [142]. Водночас, зважаючи на порівняння F-score, можна зробити висновок, що даний показник як середнє гармонійне демонструє збалансовану оцінку результатів експериментів. Попри те, що значення метрики F-score в експерименті з СФЛД та експерименті з використанням KNN майже збігаються, останній характеризується значенням FPR=1,37 %, в той час як запропонований метод показує значно нижче значення FPR=0,2 %.

Отже, можна зробити висновок, що спроможність розробленого програмного комплексу хибно ідентифікувати нормальний програмний додаток як шкідливий (помилка 2-го роду) набагато менше (більше ніж у 5 разів). Але якщо оцінювати покращення функціональної безпеки за загальним показником *F-score*, то досягнуте підвищення ефективності класифікації шкідликих додатків становить від 1 до 5 відсотків.

З погляду тривалості виявлення шкідливого програмного забезпечення, яка є значною для клієнтських реалізацій методів MLA безпосередньо на мобільних пристроях, запропонований метод виявлення шкідливого ПЗ не має цього недоліку, оскільки його алгоритми та аналіз СФЛД виконуються на серверній стороні у хмарному середовищі, де обчислювальний потенціал набагато вищий. Тож час відгуку буде визначатися передусім швидкістю мережі. Також перевагою віддаленого сервера є логічна інкапсуляція реалізації разом із неможливою процедурою зворотного інжинірингу в порівнянні з рішенням на базі клієнта Android, де навіть обфускація коду не є надійним способом інкапсуляції реалізації. Але, враховуючи необхідність виконання

порівняння СФЛД при виконанні додатків у реальному часі, для запобігання можливостей впливу на функціональну безпеку мобільного пристрою використання запропонованого методу, як і всіх інших, слід вважати більш доречним на етапі попередньої перевірки додатків.

4.4 Висновки до розділу 4

1. Розроблена інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв області є розвитком існуючих напрацювань в даній області, реалізуючи запропоновані в даному досліджені модель прав доступу ОС Android, модель СФЛД програмних додатків та метод динамічного виявлення шкідливих додатків, використовуючі створений програмний комплекс для інформцвання користувачів програмних пристроїв про потенційний ризик порушення функціональної безпеки.

2. Як первинне джерело даних для розробленої інформаційної технології розглядається база шкідливого ПЗ Malgenome. Однак запропонований метод подудови бази даних СФЛД шкідливих додатків, який базується на процедурі розкладаня даного файлу на ланцюжки та використанні додаткової індексації по псевдлосимволам API функцій, можна вважати універсальним щодо можлисвостей використання інших джерел с даними про шкідливе ПЗ.

3. Обчислення базових показників ідентичності та подібності відбувається за допомогою комбінації локального та глобального вирівнювання СФЛД В разі наявності збігу аналізуємої СФЛД із шаблонними, присутніми в вихідній БД, формується вектор атаки, який визначає напрям атаки (група дозволів), характер атаки (тобто який саме дозвіл потребує API функція) та рівень небезпеки. Завдяки отриманому занченню коефіцієнта ідентичності вдається розрахувати ризик порушення функціональної безпеки з боку програмного додатку та надати користувачу необхідну інформацію.

4. Проведення експериментів з метою перевірки ефективності запропонованої технології відбувається з використанням стандартної матриці

невідповідності. Отримані результати показують, що збільшення числа нешкідливих додатків може призвести до значного зменшенням значення FPR (помилки другого роду). Цей факт підтверджує правильність гіпотези про те, що ланцюжки СФЛД шкідливих і нешкідливих додатків суттєво відрізняються.

5. Перевірка ефективності розробленої інформаційної технології показала показала її значні переваги перед існуючими аналогами динамічного виявлення шкідливих додатків перш за все за показником помилки другого роду (більше ніж у 5 разів) та незначне підвищення ефективності за загальним показником на 1-5 відсотків.

Результати досліджень, приведених в розділі, опубліковані в роботах [44,98].

ВИСНОВКИ

У дисертаційній роботі сформульовано та вирішене актуальне наукове завдання з подальшого розвитку інформаційної технології забезпечення функціональної безпеки мобільних пристроїв за рахунок удосконалення існуючої моделі безпеки ОС Android шляхом впровадження методу динамічного виявлення потенційно небезпечних додатків.

Для досягнення поставленої мети, яка полягає в підвищенні ефективності системи функціональної безпеки мобільних пристроїв за рахунок удосконалення моделі безпеки ОС Android з можливістю врахування ризику використання шкідливих програмних додатків, були отримані такі результати:

1. Визначено поняття функціональної безпеки щодо програмних додатків апаратно-програмної платформи Android. Показано, що аналіз ефективності функціональної безпеки має проводитись на рівні категорій показників, через визначення набору властивостей програмного коду додатку, які характеризують процес його функціонування, а також встановлення рівня шкідливості додатка.

2. Проведено аналіз існуючих систем забезпечення функціональної безпеки мобільних додатків ОС Android як комплексу систем моніторингу та управління. Побудовано узагальнену схему роботи мобільного пристрою з погляду забезпечення функціональної безпеки.

3. Виділено основні типи загроз, пов’язаних із використанням програмних додатків, такі як наявність зловмисного програмного коду, наявність вразливостей у програмних застосунках, перехоплення зловмисниками конфіденційних даних, некоректний опис програмного застосунку.

4. Запропоновано класифікацію типів шкідливих додатків, яка, на відміну від існуючих, базується на групуванні API функцій додатків та дає

можливість оцінити їх за ступенем потенційних впливів при прийнятті рішень на використання додатків за запропонованими атрибутами вектора атаки.

5. Розроблена модель прав доступу при взаємодії ОС Android із програмними додатками, яка встановлює відношення між групами дозволів, дозволами та функціями API та дає можливість ввести кодування функцій для їх ідентифікації.

6. Визначено базову модель псевдосимволу СФЛД, яка складається з етапів послідовного знаходження збігів та індексації групи привілеїв, самого привілею та відповідної API функції. Псевдосимволи, створені на основі цієї моделі, дозволяють унікально ідентифікувати API-функції, що використовує програмний додаток, та прискорити пошук збігів СФЛД.

7. Розроблено метод динамічного аналізу програмних додатків, що базується на побудові сигнатури функціонального ланцюжка додатка API функції та порівняння його з шаблонною СФЛД. Визначено основні етапи процесу аналізу додатків ОС Android та розроблено відповідний математичний апарат на основі відомих алгоритмів вирівнювання послідовностей з біоінформатики. Проведено аналіз ефективності роботи запропонованого методу на основі статистичних даних та зразків зловмисного програмного коду з Android Malgenome Project.

8. Проведено аналіз сучасного програмного забезпечення функціонального трасування API викликів для операційної системи Android. Запропонована функціональна схема перехоплення та декорування API викликів за допомогою розроблених скриптів для Frida Framework.

9. Розроблено програмний комплекс, який працює у хмарному середовищі та забезпечує перевірку програмних додатків, що використовуються мобільними пристроями, на предмет їх збігу зі шкідливими додатками за послідовністю викликів API функцій з одночасним інформуванням користувача про можливі наслідки використання шкідливого програмного забезпечення з визначенням потенційного ризику.

10. Визначено особливості експериментального дослідження запропонованих методів забезпечення функціональної безпеки ОС «Android» та способи можливого ухилення шкідливого додатку від ідентифікації у випадку їх виконання у віртуальному середовищі або емуляторі.

11. Здійснена перевірка достовірності отриманих результатів шляхом їх порівняння з відомими результатами для алгоритмів машинного навчання, які підтвердили високий рівень класифікації шкідливих додатків при використанні запропонованого методу за показником F-score, який навіть перевищує відомі аналоги, при точності обчислення TPR в 0,007 та значному зменшенні помилок 2-го роду більше ніж у 5 разів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile Statistics Report, 2019-2023 – Executive Summary. https://www.radicati.com/wp/wp-content/uploads/2019/01/Mobile_Statistics_Report,_2019-2023_Executive_Summary.pdf

2. Oberlo . (n.d.). *How many people have smartphones in 2021?* https://www.oberlo.com/statistics/how-many-people-have-smartphones.

3. Кабінет Міністрів України (2018). *Про схвалення розвитку цифрової економіки та суспільства України*. https://zakon.rada.gov.ua/laws/show/67-2018-%D1%80#Text.

4. *Державні послуги онлайн*. https://diia.gov.ua/

5. Toninelli, D., Revilla, M. (2016). Smartphones vs PCs: Does the Device Affect the Web Survey Experience and the Measurement Error for Sensitive Topics? A Replication of the Mavletova & Couper's 2013 Experiment. *Survey Research Methods*, *10*(2), 153-169. doi:10.18148/srm/2016.v10i2.6274.

6. Basant, A., & Mittal, N. (2012). Hybrid approach for detection of anomaly network traffic using data mining techniques. *Proc. Technol.*, 6, 996-1003. DOI: 10.1016/j.protcy.2012.10.121.

7. Dusan, S., Vlajic, N., & An, A. (2012). Detection of malicious and non-malicious website visitors using unsupervised neural network learning. *Applied Soft Comput.*, 13: 698-708. DOI: 10.1016/j.asoc.2012.08.028.

8. Ghazali, K.W.M., & Hassan, R. (2011). Flooding distributed denial of service attacks-a review. *J. Comput. Sci.*, 7, 1218-1223.

9. Mahmoudi, C. (2017). Cloud and Mobile Cloud Architecture, Security and Safety. In *Handbook of System Safety and Security* (pp. 199-223). doi:10.1016/b978-0-12-803773-7.00010-3.

10. Roche, E., Hochleitner, M., & Summers, A. (2017). Introduction to functional safety assessments of safety controls, alarms, and interlocks: How efficient are your functional safety projects? *Process Safety Progress*, 36(4), 392-398. doi:10.1002/prs.11886.

11. В.С. Харченко, В.В. Скляр, Е.В. Брежнев. *Безопасность информационно-управляющих систем и инфраструктур*. Palmarium Academic Publishing, 2013. 528 с.

12. Pandita R, Xiao X, Yang W, Enck W, Xie T (2013) WHYPER: towards automating risk assessment of mobile applications. *Proceedings of the 22nd USENIX conference onsecurity*. https://www.usenix.org/system/files/conference/usenixsecurity13/sec13paper_pandita.pdf.

13. Hoque, N., Monowar, H., Bhuyan, M.H., Baishya, R.C., Bhattacharyyaa, D.K., & Kalitab J.K. (2014). Network attacks: Taxonomy, tools and systems.

14. Allen, G. (2015). Android Security and Permissions. Beginning Android (pp. 343-354). doi:10.1007/978-1-4302-4687-9.

15. Android Security Internals. (2015). *Network Security*, 2015(6), 4. https://doi.org/10.1016/S1353-4858(15)30046-5.

16. Butow, E. (2018). *Pro iOS Security and Forensics* (pp. XI, 160). Apress, Berkeley, CA. doi:10.1007/978-1-4842-3757-1.

17. Friesen, J. (2013). Exploring the Collections Framework. Learn Java for Android Development, 327-405. doi:10.1007/978-1-4302-5723-3_9.

18. Jackson, W. (2012). Android Framework Overview. Android Apps for Absolute Beginners, 99-123. doi:10.1007/978-1-4302-4789-0_5.

19. Kołek, K. (2013). Application of Android OS as real-time control platform. *Automatyka/Automatics*, 17(2), 197. doi:10.7494/automat.2013.17.2.197.

20. Liu, C., Zhang, Z., & Wang, S. (2016). An Android Malware Detection Approach Using Bayesian Inference. 2016 IEEE International Conference on Computer and Information Technology (CIT). doi:10.1109/cit.2016.76.

21. Rajalakshmi, B., & Anusha, N. (2017). Sensor based application for malware detection in android OS(Operating System) devices. *2017 International Conference on Information Communication and Embedded Systems (ICICES)*. doi:10.1109/icices.2017.8070722.

22. *Oracle Mobile Security. A Technical Overview*. (May 2015). Oracle white paper. https://www.oracle.com/technetwork/middleware/id-mgmt/overview/omss-technical-wp-2104766.pdf.

23. Gentile, M., & Summers, A.E. (2006). Random, systematic, and common cause failure: How do you manage them? *Process Safety Progress*, *25*(4), 331-338. https://doi.org/10.1002/prs.10145.

24. Zhang, M., & Yin, H. (2016). Automatic Generation of Security-Centric Descriptions for Android Apps. *SpringerBriefs in Computer Science Android Application Security* (pp. 77-98). https://doi.org/10.1145/2810103.2813669

25. An introduction to Functional Safety and IEC 61508. Application Note. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwiRyYzszsnyAhWDh_0HHXR7CWQQFnoECAcQAQ&url=https%3A%2F%2Fwww.mtl-inst.com%2Fimages%2Fuploads%2Fdatasheets%2FApp_Notes%2FAN9025.pdf&usg=AOvVaw3raK26ureh3TcAWagn_fcP.

26. ДСТУ EN 61508-1:2019 (2019). *Функційна безпечність електричних,* електронних*, програмованих систем*. http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=84383.

27. Harris, M. (2014). Data Center Infrastructure Management. *Data Center Handbook* (pp. 601-618). doi:10.1002/9781118937563.ch33.

28. Clarke, S. ExVeritas Limited. *An Introduction to Functional Safety and Safety Integrity Levels.* https://www.exveritas.com/wp-content/uploads/2013/01/anintroductiontosafetyintegritylevels.pdf.

29. Hoque N., Monovar H. (2014). Network attacks: Taxonomy, tools and systems. *J. Netw. Comput. Applic*., 40, 307-324. DOI: 10.1016/j.jnca.2013.08.001i.

30. Kumar, P.A.R., & Selvakumar, S. (2012). Detection of distributed denial of service attacks using an ensemble of adaptive and hybrid neuro-fuzzy systems. *Comput. Commun.*, 36, 303-319. https://doi.org/10.1016/j.comcom.2012.09.010.

31. Kumar, P.A.R., & Selvakumaras, S. (2011). Distributed denial of service attack detection using an ensemble of neural classifier. *Comput. Commun.*, 34, 1328-1341. DOI: 10.1016/j.comcom.2011.01.012.

32. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & K. Rieck (2014). Drebin: Effective and explainable detection of android malware in your pocket. *Proc. of Annual Symposium on Network and Distributed System Security(NDSS). The Internet Society*. https://www.ndss-symposium.org/wp-content/uploads/2017/09/11_3_1.pdf.

33. Yu, J., Huang, Q., &Yian, C. (2016). DroidScreening: A practical framework for real-world Android malware analysis. *Security and Communication Networks*, *9*(11), 1435-1449. doi:10.1002/sec.1430.

34. Bhandari, S., Gupta, R., Laxmi, V., Gaur, M., Zemmari, A., Anikeev, M. (2015). DRACO: DRoid analyst combo an android malware analysis framework. *Proceedings of the 8th International Conference on Security of Information and Networks* (SIN) (pp. 283-289).

35. Somarriba, O., Zurutuza, U., Uribeetxeberria, R., Delosières, L., Nadjm-Tehrani, S. (2016). Detection and Visualization of Android Malware Behavior. *Journal of Electrical and Computer Engineering*, ID 8034967. https://downloads.hindawi.com/journals/jece/2016/8034967.pdf.

36. Lundteigen, M.A., & Rausand M. (n.d.). *Chapter 3. Failures and Failure Classification.* Norwegian University of Science and Technology – NTNU. https://www.ntnu.edu/documents/624876/1277046207/SIS+book+-+chapter+03+-+failures+and+failure+classification/36f29566-bd55-4a91-b002-1e17a177c035.

37. Android Malware Detection And Prevention. (2017). *International Journal of Recent Trends in Engineering and Research*. 3(2): 213-217. doi:10.23883/ijrter.2017.3028.0uhbl.

38. Moreno L, Aponte J, Sridhara G, Marcus A, Pollock L, Vijay-Shanker K (2013) Automatic generation of natural language summaries for Java classes. In: Proceedings of the 2013 IEEE21th international conference on program comprehension (ICPC'13), 2013.

39. Chakraborty, S., Chatterjee, S., Dey, N., Ashour, A. S., &Hassanien, A. E. (2016). Comparative Approach Between Singular Value Decomposition and Randomized Singular Value Decomposition-based Watermarking. *Intelligent Techniques in Signal Processing for Multimedia Security Studies in Computational Intelligence*. (pp. 133-149).

40. Chen, H., Zhao, J., Luo, Q., &Hou, Y. (2017). Distributed randomized singular value decomposition using count sketch. *International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*. doi:10.1109/spac.2017.8304273.

41. Polyakov, S. (2015). Enhancing user search experience in digital libraries with rotated latent semantic indexing. *Proceedings of the Association for Information Science and Technology*, 52(1), 1-4.

42. *SimpleNLG: Java API for Natural Language Generation* (2016) https://code.google.com/p/simplenlg.

43. Shinkarenko, A. (2020). Internet of everything vs internet of things, explained. *Itransition: Software Development Company*. https://www.itransition.com/blog/internet-of-everything-vs-internet-of-things.

44. Казимир В., Карпачев І. (2018), Усик А. Моделі систем безпеки ОС Android. *Технічні науки та технології*. 2 (12). с. 116-126.

45. Kearns, A. (2010). *5-Layer Architecture. DRAFT (v0.2, Nov 2010).* https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjYnZqUmMryAhVVgP0HHVPTDAkQFnoECA8QAQ&url=https%3A%2F%2Fmorphological.files.wordpress.com%2F2011%2F08%2F5-layer-architecture-draft.pdf&usg=AOvVaw3Uqo_c1LL1JMHzz0lM-6Vs.

46. *Functional Safety SMART Current Driver KCD2-SCD-Ex1.ES(.SP), HiC2031ES*. (2020). Pepperl+Fuchs Group. https://files.pepperl-fuchs.com/webcat/navi/productInfo/doct/tdoct6682__eng.pdf?v=20200428105202.

47. Anthe, C., Chrzan, P., Florio, E., Foster, C., Henry, P., Jones, J., Ng, N., O'Sullivan, N., Pecelj, D., Penta, A., Ragragio, I., Rains, T., & Rebriy, P. (2015). *Microsoft Security Intelligence Report*, 19. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwj36Nua4OzyAhVDOBoKHd9OCpgQFnoECAIQAQ&url=https%3A%2F%2Fdownload.microsoft.com%2Fdownload%2F4%2F4%2FC%2F44CDEF0E-7924-4787-A56A-16261691ACE3%2FMicrosoft_Security_Intelligence_Report_Volume_19_English.pdf&usg=AOvVaw1prI6Ax8ptpC832VAS-bB8 .

48. Prabhakaran, V., Arpaci-Dusseau, A.C., & Arpaci-Dusseau, R.H. (2005). Analysis and Evolution of Journaling File Systems. *Proceedings of the annual conference on USENIX Annual Technical Conference* (pp. 105-120). https://www.usenix.org/legacy/events/usenix05/tech/general/full_papers/prabhakaran/prabhakaran.pdf.

49. Hashimoto, M. (2013). *A Survey of Security Research for Operating Systems*. https://www.iisec.ac.jp/proc/vol0005/hashimoto13.pdf.

50. Beckers, Cote, I., Frese, T., Hatebur, D., Heisel, M. (2014). Systematic Derivation of Functional Safety Requirements for Automotive Systems. In *Bondavalli A., Di Giandomenico F. (eds) Computer Safety, Reliability, and Security. SAFECOMP 2014. Lecture Notes in Computer Science*, *8666*. Springer, Cham. https://doi.org/10.1007/978-3-319-10506-2_5.

51. Berntsson, P.S. (2017). *Evaluation of open source operating systems for safety-critical applications. Master's thesis in Embedded Electronic System Design*. Department of Computer Science and Engineering Chalmers University of Technology University of Gothenburg. https://publications.lib.chalmers.se/records/fulltext/249901/249901.pdf.

52. Nardi, A., Armato, A. (2017). Functional Safety Methodologies for Automotive Applications. *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (17496618). DOI: 10.1109/ICCAD.2017.8203886.

53. Sangmin, Ha. (2017). *A Study on Android Malware Flow Analysis based on Dalvik Instruction* (Master's thesis). http://www.riss.kr.

54. Arzt, Steven, et al. (2014). Flowdroid: Precise context, flow, field, objectsensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, *49*(6), 259–269.

55. Sridhara, G., Hill, E., Muppaneni, D., Pollock, L., Vijay-Shanker, K. (2010). Towards automatically generating summary comments for Java methods. *Proceedings of the IEEE/ACM international conference on automated software engineering (ASE'10)* (pp. 43-5). https://dl.acm.org/doi/abs/10.1145/1858996.1859006.

56. Sridhara, G., Pollock, L., Vijay-Shanker, K. (2011). Automatically detecting and describing high level actions within methods. *Proceedings of the 33rd international conference on softwareengineering (ICSE'11).* DOI: 10.1145/1985793.1985808.

57. Yan, H., &Peng, L. (2018). *Android malware detection based on evolutionary super-network*. doi:10.1063/1.5033818.

58. Казимир В., Карпачев І. Функціональна безпека архітектури мобільної операційної системи Android. *Технічні науки та технології*. 2015. № 1(77). С. 127-134.

59. Карпачев И.И., Системо-центрическая безопасность в Android. *Математичне та імітаційне моделювання систем «МОДС 2015» : X Міжнародна науково-практична конференція (Чернігів, 22-26 Червня 2015 р.)*. Чернігів, 2015.с. 422-424

60. Jusoh R., Firdaus A., Anwar S., Osman M.Z., Darmawan M.F., Ab Razak M.F. (2021). Malware detection using static analysis in Android: a review of FeCO (features, classification, and obfuscation). *PeerJ Comput. Sci*., 7, e522 http://doi.org/10.7717/peerjcs.522.

61. Gordon, D. G. (2016). Legal Aspects of Cloud Computing. *Encyclopedia of Cloud Computing* (pp. 462-475). doi:10.1002/9781118821930.ch38.

62. Khaddar, M. A., & Boulmalf, M. (2017). Smartphone: The Ultimate IoT and IoE Device. In *N. Mohamudally, Smartphones from an Applied Research Perspective*. doi:10.5772/intechopen.69734.

63. Manashty, A., & Thompson, J. L. (2017). Cloud Platforms for IoE Healthcare Context Awareness and Knowledge Sharing. *Internet of Things Beyond the Internet of Things*, 303.

64. Safonov, V. O. (2016). Appendix A: Example Of Microsoft Azure Cloud Service: Filemanager. *Trustworthy Cloud Computing* (pp. 299-308). doi:10.1002/9781119114215.app1.

65. Казимир В.В., Карпачев І.І., Литвин С.В., Усік А.М. Архітектура моделей системи безпеки мережі інтернету речей на базі ОС Android. *Математичне та імітаційне моделювання систем «МОДС 2018»* : XIII Міжнародна науково-практична конференція (Київ-Чернігів-Жукін, 25-29 червня 2018 р.). Чернігів, 2018. С. 335-336

66. Linthicum, D. S. (2017). Connecting Fog and Cloud Computing. *IEEE Cloud Computing*, 4(2), 18-20. doi:10.1109/mcc.2017.37.

67. Security in the Cloud. (2017). CCSP® (ISC)2® Certified Cloud Security Professional Official Study Guide, 87-113. doi:10.1002/9781119419372.ch5.

68. Sojaat, Z., & Skalaa, K. (2017). The dawn of Dew: Dew Computing for advanced living environment. 2017 *40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. doi:10.23919/mipro.2017.7973447.

69. SDK Platform Tools release notes. Android Developers. (n.d.). [Електронний ресурс]. URL:https://developer.android.com/studio/releases/platform-tools. (дата звернення 03.03.2019).

70. Shaheen, J.A., Asghar, M.A., Hussain, A. (2017). Android OS with its Architecture and Android Application with Dalvik Virtual Machine Review.

International Journal of Multimedia and Ubiquitous Engineering, 12(7), 19-30. http//dx.doi.org/10.14257/ijmue.2017.12.7.03.

71. Feizollaha A., Anuar N.B., Salleh R., Suarez-Tangilbl G., Furnell S. (2017). AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Computers & Security, 65*, 121-134. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwixv4qD5cvyAhWXg_0HHfeACtYQFnoECAIQAQ&url=https%3A%2F%2Fwww.researchgate.net%2Fpublication%2F310442625_AndroDialysis_Analysis_of_Android_Intent_Effectiveness_in_Malware_Detection&usg=AOvVaw2-9Js9SSoanfJ2SvKkQzLP

72. Sridhara, G., Pollock, L., Vijay-Shanker, K. (2011). Generating parameter comments and integrating with method summaries. *Proceedings of the 2011 IEEE 19th international conferenceon program comprehension (ICPC'11)*.

73. Dissanayake, N., Jayatilaka, A., Zahedi, M., Ali Babar M. (2021). Software Security Patch Management – A Systematic Literature Review of Challenges, Approaches, Tools and Practices. *arXiv:2012.00544v3*. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjeod6lgtTyAhU9hf0HHbbfAucQFnoECAMQAQ&url=https%3A%2F%2Fwww.researchgate.net%2Fpublication%2F346555570_Software_Security_Patch_Management_A_Systematic_Literature_Review_of_Challenges_Approaches_Tools_and_Practices&usg=AOvVaw1_TGQOYPpgmqdUQDlp5HrJ.

74. *Patch Management. A newsletter for IT Professionals*. Issue 6 (n.d.). Office of Information Technology (ITO). https://ito.hkbu.edu.hk/pub/is_newsletter/professional/Issue_06_PM/JUCC%20Newsletter-IT-6%20PM.pdf.

75. Smaragdakis, Y., Balatsouras, G. (2015). Pointer Analysis. *Foundations and Trends® in Programming Languages*, *2*(1), 1-69. https://yanniss.github.io/points-to-tutorial15.pdf.

76. Yu, X., Tian, Z., Qiu J., & Jiang, F. (2018). A Data Leakage Prevention Method Based on the Reduction of Confidential and Context Terms for Smart Mobile

Devices. *Wireless Communications and Mobile Computing*. ID 5823439. https://doi.org/10.1155/2018/5823439.

77. Babak, Ya. N. (2016). Automatic deobfuscation and reverse engineering ofobfuscated code: PhD: 22.09.2016. The University of Arizona.

78. Enck, W., Gilbert, P., Chun, B.-G., Cox, L. P., Jung, J., McDaniel, P., & Sheth, A. N. (2010). *TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. OSDI'10*. https://www.usenix.org/legacy/events/osdi10/tech/slides/enck.pdf.

79. Gheorghe, L., Marin, B., Gibson, G., Mogosanu, L., Deaconescu, R., Voiculescu, V., &Carabas, M. (2015). Smart malware detection on Android. Security and Communication Networks, 8(18): 4254-4272. doi:10.1002/sec.1340.

80. Sharmeen, S., Huda, S., & Abawajy, J. (2019). Identifying Malware on Cyber Physical Systems by incorporating Semi-Supervised Approach and Deep Learning. *IOP Conf. Series: Earth and Environmental Science*, 322, 012012. https://iopscience.iop.org/article/10.1088/1755-1315/322/1/012012/pdf.

81. Shahriar, H., Islam, M., &Clincy, V. (2017). Android malware detection using permission analysis. *Southeast Con 2017*. doi:10.1109/secon.2017.7925347.

82. Казимир В., Карпачев І. Забезпечення контролю доступу застосувань до ресурсів ОС Android методом системної безпеки. *Технічні науки та технології*. 2016. № 4 (6). С. 131-138.

83. *Linux Role-Based Access Control (RBAC) Lab* (2006). Wenliang Du, Syracuse University. https://web.ecs.syr.edu/~wedu/seed/Labs/RBAC_Linux/RBAC_Linux.pdf.

84. Odusami, M., Abayomi-Alli, O., Misra, S., Shobayo, O., Damasevicius, R., Maskeliunas, R. (2018). Android Malware Detection: A Survey. *In: Florez H., Diaz C., Chavarriaga J. (eds) Applied Informatics. ICAI 2018. Communications in Computer and Information Science, 942*. https://doi.org/10.1007/978-3-030-01535-0_19.

85. Saish, K. (2019). *List of 57 Android App Permissions Explained*. Techk47. https://www.techk47.com/android-app-permissions-list.

86. Google Play Store - Free downloads and reviews - CNET Download.com. (n.d.). Retrieved from https://download.cnet.com/s/google-play-store/.

87. Aron, L., Hanacek, P. (2016). Dynamic Permission Mechanism on Android. *Journal of Software*, *11*(12), 1224-1230. http://www.jsoftware.us/vol11/221-CS020.pdf.

88. USING FIREEYE ENDPOINT FOR PCI AND HIPAA/HITECH COMPLIANCE. https://www.fireeye.com/content/dam/fireeye-www/products/pdfs/cg-pci-and-hipaa-compliances.pdf.

89. Kazymyr, V., Karpachev, I. (2015). Improving of existing permission system in Android OS. *International Journal "Information Models and Analyses"*, *4*, 336-344.

90. *2020 State of Malware Report* (2020). Alwarebytes LABS. https://www.malwarebytes.com/resources/files/2020/02/2020_state-of-malware-report.pdf.

91. SECURITY REPORT 2018/19 (2018). AV-TEST. https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf.

92. Yajin, Zhou, Xuxian, Jiang (2012). Dissecting Android Malware: Characterization and Evolution. *Proceedings of the 33rd IEEE Symposium on Security and Privacy* (Oakland 2012). San Francisco. http://www.malgenomeproject.org.

93. Zhi, Xiong (2021). *Dataset for android malware detection*. https://ieee-dataport.org/documents/dataset-android-malware-detection#files.

94. Yajin, J., Xuxian, J. (2015). *Android Malware Genome Project.* North Carolina State University, http://www.malgenomeproject.org

95. Mueller, B. (2016). *Hacking Soft Tokens. Advanced Reverse Engineering on Android*. Vantage Point Security Pte. Ltd. https://gsec.hitb.org/materials/sg2016/whitepapers/Hacking%20Soft%20Tokens%20-%20Bernhard%20Mueller.pdf.

96. PScout: Analyzing the Android Permission Specification. https://security.csl.toronto.edu.

97. Qu, Z., Rastogi, V., Zhang, X., Chen, Y, Zhu, T., Chen, Z. (2014) Autocog: measuring the description-to-permission fidelity in Android applications. *Proceedings of the 21st conference oncomputer and communications security (CCS)*. https://users.cs.northwestern.edu/~ychen/Papers/CCS14.pdf.

98. Карпачев І., Казимир В. Виявлення шкідливих додатків ОС Android по сигнатурі функціонального ланцюжка додатку. *Технічні науки та технології*. 2021. № 4 (18). С. 85-91.

99. Axplorer Android Permission Mappings. A static analysis tool to study Android's application framework's internals. https://github.com/reddr/axplorer.

100. Al-Rayes, H. T. (2012). Studying Main Differences between Android & Linux Operating Systems. *International Journal of Electrical & Computer Sciences IJECS-IJENS, 12*(05), 46-49. http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=01320E30A69700E996370F6C62DE6E5A?doi=10.1.1.654.6317&rep=rep1&type=pdf.

101. Ficco, M. (2020). Comparing API Call Sequence Algorithms for Malware Detection. In *Barolli L., Amato F., Moscato F., Enokido T., Takizawa M. (eds) Web, Artificial Intelligence and Network Applications. WAINA 2020. Advances in Intelligent Systems and Computing, 1150*. Springer, Cham. https://doi.org/10.1007/978-3-030-44038-1_77.

102. Donkor, E. S., Dayie, N. T. K. D., Adiku, T. K. (2014). Bioinformatics with basic local alignment search tool (BLAST) and fast alignment (FASTA). *Journal of Bioinformatics and Sequence Analysis*, *6*(1), 1-6. https://www.researchgate.net/publication/287343451_Bioinformatics_with_basic_local_alignment_search_tool_BLAST_and_fast_alignment_FASTA.

103. Zhang, Y.-Q., Rajapakse, J.C. (Eds.). (2009). *Machine learning in bioinformatics*. John Wiley & Sons, Inc. https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470397428.fmatter.

104. *Sequence Alignment and Dynamic Programming* (n.d.). https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&u

act=8&ved=2ahUKEwjh9LH9wsfyAhXZg_0HHeZiDgkQFnoECA0QAQ&url=https%3A%2F%2Fwww.bibalex.org%2Fcssp%2FPresentations%2FAttachments%2FSequence%2520Alignment%2520and%2520Dynamic%2520Programming.pdf&usg=AOvVaw16Vh3im17cqr1JtRzFafe2.

105. Nicholas, H.B. Jr, Ropelewski, A. J., Deerfield, D. W. (2002). Strategies for Multiple Sequence Alignment. *BioTechniques*, *32*(3), 572-591. https://www.future-science.com/doi/pdf/10.2144/02323rv01.

106. Rosenberg, M. S. (2009). Sequence Alignment. Concepts and History. University of California Press. https://content.ucpress.edu/chapters/10874.ch01.pdf.

107. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48(3), 443–453 (1970.

108. National Center for Biotechnology Information (n.d.). *Basic Local Alignment Search Tool*. https://blast.ncbi.nlm.nih.gov/Blast.cgi.

109. Xiaoqiu, Huang, Ross, C. Hardison, Webb, Miller (1990). A space-efficient algorithm for local similarities. *Bioinformatics*, *6*(4), 373-381. http://www.cs.cornell.edu/courses/cs628/2004fa/secure/papers/Huang_Hardison_Miller_space-efficient_algorithm_for_local_similarities_CABIOS90.pdf.

110. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. *J. Mol. Biol*. 147(1), 195–197 (1981).

111. Zahariev, M., Dahl, V., Chen, W., Lévesque, C. A. (2009). Efficient algorithms for the discovery of DNA oligonucleotide barcodes from sequence databases. *Molecular Ecology Resources*, 9 (Suppl. 1), 58–64. https://onlinelibrary.wiley.com/doi/full/10.1111/j.1755-0998.2009.02651.x.

112. Xu W., Zhang F., Zhu S. (2012). The power of obfuscation techniques in malicious JavaScript code: A measurement study. *7th International Conference. Malicious and Unwanted Software (2012)*. DOI: 10.1109/MALWARE.2012.6461002.

113. Lawrence, A.C. (2008). Optimizing compilation with the value state dependence graph. Great Britain: University of Cambridge. (Cambridge CB3 0FD).

114. Kazymyr, V., Karpachev, I. (2016). Improving time-critical code performance with JNI. *Mathematical machines and systems*, *2*, 72-77.

115. Singh, A. (2009). Identifying malicious code through reverse engineering. New York, NY: Springer.

116. Android Anti-Reversing Defenses (n.d.). *OWASP/owasp-mstg*. https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md#testing-emulator-detection.

117. *SafetyNet Attestation API. Android Developers*. (n.d.). Android Developers. https://developer.android.com/training/safetynet/attestation.

118. Daehee, J., Yunjong, J. (2019). Rethinking anti-emulation techniquesfor large-scale software deployment. *Science Direct*, 83, 186-190. https://cysec.kr/publications/emul-detect.pdf.

119. *Flutter* (n.d.). https://github.com/flutter/plugins/tree/master/packages/device_info.

120. *What is Cuckoo?* (n.d.). Cuckoo. https://cuckoosandbox.org.

121. Voctor, A. (2017). *ANDRIK: Automated Android Malware Analysis*. http://openaccess.uoc.edu/webapps/o2/bitstream/10609/60606/6/vacinsTFM0117memoria.pdf.

122. Ziadia, M., Fattahi, J., Mejri, M., Pricop, E. (2020). Smali+: An Operational Semantics for Low-Level Code Generated from Reverse Engineering Android Applications. *Information*, *11*, 130. https://doi.org/10.3390/info11030130.

123. Xposed Module Repository (n.d.). *Xposed Installer*. https://repo.xposed.info/module/de.robv.android.xposed.installer.

124. Ziyue, Yang (n.d.). *Yzygitzh/ReDroid*. https://github.com/yzygitzh/ReDroid.

125. Lok, *K., (2013) Transparent and precise Malware analysis Using Virtualization: From theory to Practise*. Surface. 5, 66-74 https://core.ac.uk/download/pdf/215694562.pdf.

126. Stordeur, J.U. (2018). *Bypassing Android Anti-Emulation, Part (I).* https://www.juanurs.com/Bypassing-Android-Anti-Emulation-Part-I.

127. *Anti-android emulator detection solution* (n.d.). https://i.cs.hku.hk/fyp/2018/ fyp18033/.

128. Elizarov, R. (n.d.). *Introduction to Coroutines.* https://resources.jetbrains.com/storage/products/kotlinconf2017/slides/2017+KotlinConf+-+Introduction+to+Coroutines.pdf.

129. Chen, H., Li, Z., Jiang, Q., Rasool, A., Chen, L. (2021). A Hierarchical Approach for AndroidMalware Detection Using Authorization-Sensitive Features. *Electronics*, *10*, 432. https://doi.org/10.3390/electronics10040432.

130. *Java™ 2 Platform, Standard Edition 5.0. Troubleshooting and Diagnostic Guide* (2007). Sun Microsystems, Inc. https://www.oracle.com/technetwork/java/jdk50-ts-guide-149808.pdf *(*p. 81-84*)*.

131. Karpachev I., Bounded context in strategic domain driven design. *Математичне та імітаційне моделювання систем «МОДС 2016»*: XI Міжнародна науково-практична конференція (м. Чернігів-с. Жукін, 27 червня – 1 липня 2016 р.). Чернігів, 2016. С. 398-400.

132. Cobbaut, P. (2012). *Linux Fundamentals.* Netsec BVBA. https://www2.bus.umich.edu/sites/default/files/files/rosstech/research-comp/LinuxFundamentals.pdf.

133. Frida Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers https://www.frida.re.

134. Казимир В., Карпачев І., Сіпаков В. Динамічний аналіз послідовностей API-викликів OS Android. *Технічні науки та технології.* 2019. № 4 (18). С. 85-91.

135. Soares, A. and Jr., R. (2017). A Technique for Extraction and Analysis of Application Heap Objects within Android Runtime (ART). *In Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)* (pp. 147-156). https://www.scitepress.org/papers/2017/62041/62041.pdf.

136. Michael, H. (2015). *Android Security*. (p. 25-27).. https://d-nb.info/1154866548/34.

137. Lingling Fan, Ting Su, Sen Chen, Guozhu Meng, Yang Liu, Lihua Xu, Geguang Pu, Zhendong Su. Large-Scale Analysis of Framework-Specific Exceptions in Android Apps. *ICSE '18: Proceedings of the 40th International Conference on Software Engineering.* 2018 . P. 408–419).

138. Cotter, H. (2007). *Search me: using SQL server full-text search*. https://cdn.ttgtmedia.com/searchSQLServer/downloads/insider_fulltextsearch_0121.pdf.

139. Narkhede, S. *Understanding Confusion Matrix*. Medium. (2018). https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

140. Alguliyev, R.M., Aliguliyev, R. M., Imamverdiyev, Y. N., Sukhostat, L. V. (2018). An improved ensemble approach for dos attacks detection. *Радіоелектроніка, інформатика, управління*, 2, 73–82.

141. Narudin, F. A., Feizollah, Ali., Anuar, N. B., Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing, 20(1)*, 343–357. DOI: 10.1007/s00500-014-1511-6.

142. Tiwari, S. (n.d.). *Android Malware Dataset for Machine Learning*. Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/shashwatwork/android-malware-dataset-for-machine-learning?select=drebin-215-dataset-5560malware-9476-benign.csv.

ДОДАТКИ

ДОДАТОК А

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації:

1. Казимир В., Карпачев І. Функціональна безпека архітектури мобільної операційної системи Android. Технічні науки та технології: науковий журнал. 2015. № 1 (77). С. 127-134.

2. Kazymyr V., Karpachev I. Improving of existing permission system in Android OS. *International Journal "Information Models and Analyses*". 2015. Vol.4. P. 336-344. ISSN 1314-6416 (printed). ISSN 1314-6432 (Online).

3. Kazymyr V., Karpachev I. Improving time-critical code performance with JNI. *Mathematical machines and systems*. 2016. Vol. 2.P.72-77.

4. Казимир В., Карпачев І. Забезпечення контролю доступу застосувань до ресурсів ОС Android методом системної безпеки. *Технічні науки та технології*: науковий журнал. 2016. № 4 (6). С. 131-138.

5. Казимир В., Карпачев І., Усик А. Моделі систем безпеки ОС Android. *Технічні науки та технології*: науковий журнал. 2018. № 2 (12). С. 116-126.

6. Казимир В., Карпачев І., Сіпаков В. Динамічний аналіз послідовностей API-викликів OS Android. *Технічні науки та технології*: науковий журнал. 2019. № 4 (18). С. 85-91.

7. Карпачев І., Казимир В. Виявлення шкідливих додатків ОС Android по сигнатурі функціонального ланцюжка додатку. *Технічні науки та технології*: науковий журнал. 2021. № 1 (23). С. 109-117.

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Карпачев И.И. Системо-центрическая безопасность в Android. Математичне та імітаційне моделювання систем «МОДС 2015» : X Міжнародна науково-практична конференція (Чернігів, 22-26 Червня 2015 р.). Чернігів, 2015. С. 422-424.

2. Karpachev I. Bounded context in strategic domain-driven design // *Математичне та імітаційне моделювання систем «МОДС 2016»*: XI Міжнародна науково-практична конференція (м. Чернігів-с. Жукін, 27 червня – 1 липня 2016 р.). Чернігів, 2016. С. 398-400.

3. Казимир В.В., Карпачев І.І., Литвин С.В., Усік А.М. Архітектура моделей системи безпеки мережі інтернету речей на базі ОС Android. *Математичне та імітаційне моделювання систем «МОДС 2018»* : XIII Міжнародна науково-практична конференція (Київ-Чернігів-Жукін, 25-29 червня 2018 р.). Чернігів, 2018. С. 335-336.

ДОДАТОК Б

АКТИ ВПРОВАДЖЕННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ

086719

УКРАЇНА

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

пр.Науки, 14, м.Харків, 61166, тел. (057) 7021-016, факс (057) 7021-013
e-mail: info@nure.ua web-сайт: http// nure.ua

14.09.2018 № 01.Д°/27-1442
на №_____

ДОВІДКА
про впровадження результатів дисертаційної роботи
Карпачева Ігоря Ігоревича

Карпачев Ігор Ігоревич є одним із розробників пілотного проекту - сайту та мобільного додатку інтернет-магазину, який розроблявся у рамках проекту Tempus «Інноваційна гібридна стратегія IT-аутсорсингового партнерства з підприємствами (Tempus IHSITOP)», № 530319-TEMPUS-1-2012-1-DE-TEMPUS-JPHES.

Результати дисертаційної роботи Карпачева І.І., що включають модель прав доступу при взаємодії з ОС Android та метод динамічного виявлення потенційно небезпечних запитів забезпечили надійну роботу мобільного додатку інтернет-магазину в умовах зовнішнього втручання.

Матеріали дисертаційного дослідження Карпачева І.І. також були використані при розробці вебінару на тему «Basic principles of software development for OS Android», який був проведений ним у рамках проекту Tempus IHSITOP з командами розробників програмного забезпечення.

Проректор з наукової роботи Неофітний М.В.

Координатор від України проекту Білоус Н.В.

Tempus IHSITOP, проф. каф.ПІ

datascope

Ref: Mr I.I. Karpachev - Technical Doctor Thesis 11th June 2021

To Whom it may concern,

This letter is written with the intent of certifying and verifying the results of Mr. I. I. Karpachev's technical doctor thesis titled *"Information technology to ensure the functional safety of mobile devices"*.

Mr. I.I. Karpachev's unique method of dynamic malware detection using functional API calls and bespoke developed software are impressive and very novel.

This letter also confirms that Mr. I.I. Karpachev successfully integrated a model of ensuring the functional security of Android mobile applications during the approach of creation and development of many Datascope mobile applications, such as: DataTouch (a market leading Digital application focused in construction industry being utilized all over the world), Datascope's DMS system (a modern Delivery Management System, which is an integral product in Datascope's Suite of applications focused on digitalising Construction logistics) and Form-Builder (Datascope's dynamic interactive forms).

Kind Regards

Samuel W Jones

General Manager
Datascope Systems Limited

Access House, Aviation Park, Chester, CH4 0GZ

Integrating big data to drive innovation, efficiency and savings

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

вул. Шевченка, 95, Чернігів, 14035, Україна

тел. +38(0462) 665-103;
факс +38(0462) 665-105
E-mail: cstu@stu.cn.ua
www.stu.cn.ua
Код ЄДРПОУ 05460798

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

CHERNIHIV POLYTECHNIC NATIONAL UNIVERSITY

95, Shevchenko str., Chernihiv, 14035, Ukraine

26.04.2021 № 201/08-590

На № _____ від _____

ДОВІДКА ПРО ВПРОВАДЖЕННЯ

результатів дисертаційної роботи Карпачева Ігоря Ігоровича «Інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв», представленої на здобуття вченого ступеня кандидата технічних наук зі спеціальності 05.13.06 - інформаційні технології

Результати наукових досліджень, викладені в дисертаційній роботі Карпачева Ігоря Ігоровича «Інформаційна технологія забезпечення функціональної безпеки мобільних пристроїв», що включають модель прав доступу при взаємодії ОС Android із програмними застосуваннями, метод динамічного виявлення потенційно небезпечних застосувань та інформаційна технологія забезпечення функціональної безпеки мобільних за рахунок динамічного управління процесами використання застосувань були впроваджені при створенні системи електронного голосування «Mobile-Rada» за договорами з Чернігівською обласною радою № 480/18, 492/19, 508/20 в період 2018-2020 роки та з Корюківською міською радою № 79/482/19 від 23.04.2018 року, та використовуються на кафедрі інформаційних та комп'ютерних систем при проведенні лекцій та лабораторних робіт з дисциплін «Програмування мобільних пристроїв» в процесі навчання бакалаврів спеціальності 123 – комп'ютерна інженерія та з дисципліни «Статистичні методи обробки інформації» в процесі навчання аспірантів спеціальності 122 – комп'ютерні науки.

Проректор з наукової роботи
Національного університету
«Чернігівська політехніка»
д.е.н., проф. В.Г. Маргасова

ДОДАТОК В

UML-ДІАГРАМИ ПРОГРАМНИХ ЗАСОБІВ

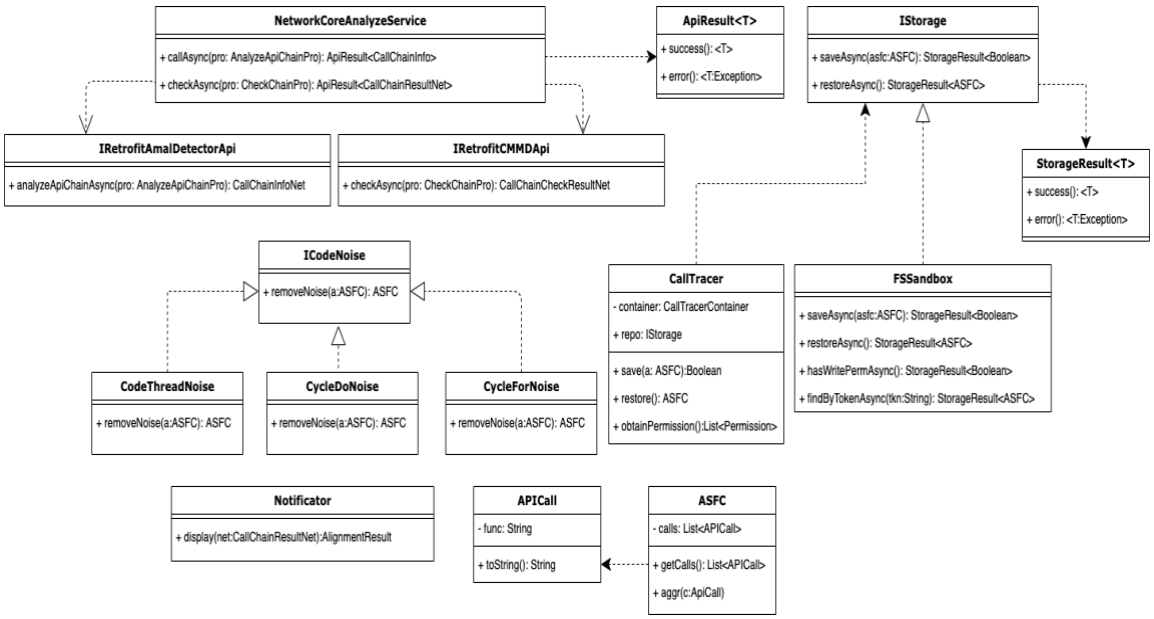


Рисунок В.1 – UML-діаграма класів програмного засобу AMalDetector

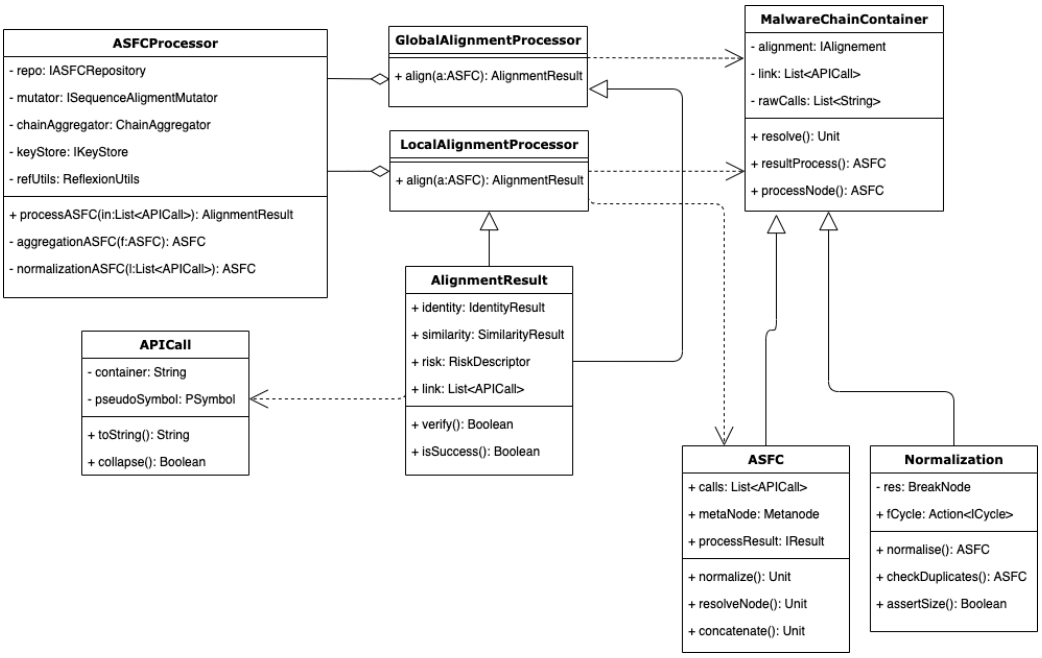


Рисунок В.2 – UML-діаграма класів програмного засобу CMMD

ДОДАТОК Г

```
class AfscProcessor(
    private val repo: IASFCRepository,
    private val sequenceAlignmentMutator: ISequenceAlignmentMutator
) : AbstractAfscProcessor() {

    fun processASFC(input: List<APICall>): AlignmentResult {
        val normalized = normalizationASFC(input)
        val localAlignmentResult = sequenceAlignmentMutator.tryOrFailLocal(sequenceAlignmentMutator)
        return if (localAlignmentResult.isSuccess) {
            repo.storeByToken(localAlignmentResult.asfcFragmentedByQ)
            localAlignmentResult
        } else sequenceAlignmentMutator.tryOrFailGlobal(normalized)
    }
    // If fragmented ASFC exists, aggregate it, return fragmented otherwise
    private fun aggregationASFC(fragmentedASFC: ASFC): ASFC = when (repo.metadataExists(fragmentedASFC)) {
        true -> {
            val initialAsfc = repo.findMetadata(fragmentedASFC.token)
            // Reorder pattern ASFC based on input intercepted fragmented ASFC
            initialAsfc.reorderAccordingTo(fragmentedASFC)
            initialAsfc.concatenateWith(fragmentedASFC)
        }
        false -> fragmentedASFC
    }

    private fun normalizationASFC(interceptedChain: List<APICall>): ASFC {
        val container = ASFC()
        // Build API call list result
        interceptedChain.forEach { container.add(it) }
        // Get rid of cycles and thread "noise"
        container.normalize()
        // Aggregate if required
        return aggregationASFC(container)
    }
}
```

Рисунок Г.1 – Вихідний код функції «ASFCProcessor» порівняння послідовностей серверного модулю «CMMD»

ДОДАТОК Д

РОЗРАХУНКИ МЕТРИК БІНАРНОГО КЛАСИФІКАТОРУ ПО РЕЗУЛЬТАТАМ ЕКСПЕРЕМЕНТІВ ТА МАТРИЦІ НЕВІДПОВІДНОСТІ

Таблиця Д.1 – Результати класифікації матриці невідповідності експерименту №1 – 100 APK з яких 94 шкідливі додатки та 6 не шкідливі додатки

Матриця невідповідності	Результат класифікації
TP	93
FP	4
FN	1
TN	2

Розрахунки метрик класифікації експерименту №1:

TPR = 93/94=0.99

FRP = 4/6=0.67

Precision=93/97=0.96

Recall=TPR=0.99

F-Score= 0.97

Таблиця Д.2 – Результати класифікації матриці невідповідності експерименту №2 – 250 APK з яких 213 шкідливі додатки та 37 не шкідливі додатки

Матриця невідповідності	Результат класифікації
TP	211
FP	2
FN	2
TN	35

Розрахунки метрик класифікації експерименту №2:

TPR=211/213=0.99

FPR=2/37=0.054

Precision=211/213=0.99

Recall=0.99

F-Score =0.99

Таблиця Д.3 – Результати класифікації матриці невідповідності експерименту №3– 500 APK з яких 350 шкідливі додатки та 150 не шкідливі додатки

Матриця невідповідності	Результат класифікації
TP	346
FP	2
FN	4
TN	148

Розрахунки метрик класифікації експерименту №3:

TPR=346/350=0.99

FPR=2/150=0.013

Precision=346/348=0.99

Recall=0.99

F-Score=0.99