Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота
магістра

з теми: **«РОЗРОБКА ПРОГРАМНОГО КОМПЛЕКСУ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ БЕЗПЕКИ ПРОГРАМНИХ ДОДАТКІВ В ОС ANDROID»**

Виконав: студент групи KN1-M21
спеціальності 122 Комп’ютерні науки
Продан Олександр Олександрович

Керівник:
Моцик Р.В.,
кандидат педагогічних наук, доцент, доцент кафедри комп’ютерних наук

Рецензент:
Сморжевський Ю.Л.,
кандидат педагогічних наук, доцент, доцент кафедри математики

Кам’янець-Подільський – 2022

ЗМІСТ

ВСТУП

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.

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 дипломної 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 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.

Структура та обсяг дипломної роботи. Дипломна робота складається зі вступу, двох розділів, висновків, списку використаних джерел та додатків. Повний обсяг дипломної роботи становить 68 сторінок, у тому числі: 67 сторінок основного тексту, 26 рисунків, 2 таблиць, список використаних джерел із 22 найменувань та 1 додатку.

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 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,

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

у 2023 році перевищить 16,8 млрд (рис. 1.1).

Рисунок 1.1 – Світовий тренд мобільних пристроїв, 2019–2023

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

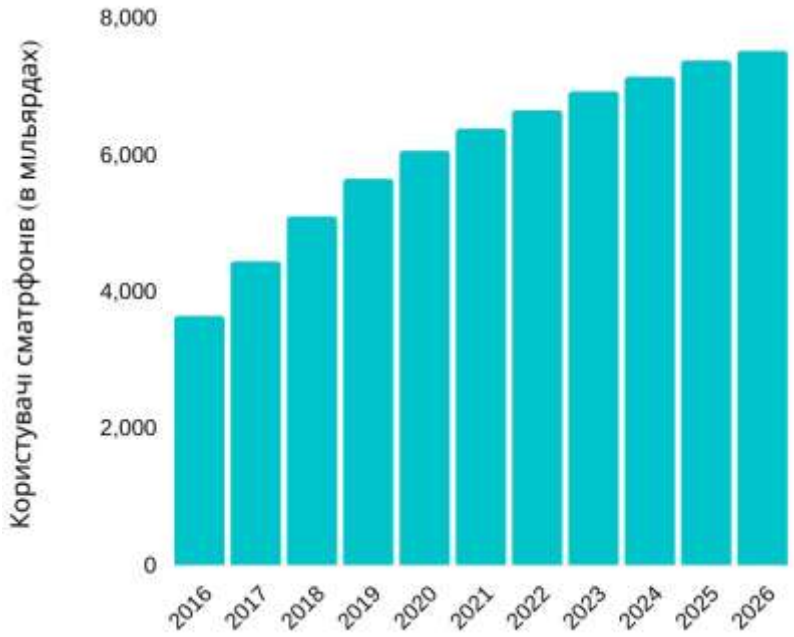


Рисунок 1.2 – Світовий тренд смартфонів, 2016–2026

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 постійним атрибутом життя людини, супроводжуючи їх та надаючи життєво необхідні можливості.

Для України питання, пов’язані із використанням мобільних пристроїв, та смартфонів зокрема, є особливо важливим в руслі схваленої 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].

Важливим етапом побудови стратегії функціональної безпеки мобільних застосувань є розуміння процесу сертифікації обладнання та ліцензування програмного забезпечення [9]. 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. 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]. 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]:

- 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. 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.

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. 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, 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 [5] та його вітчизняний аналог ДСТУ EN 62061:2014 [6]. 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 виділяють такі класи (рис. 1.3):

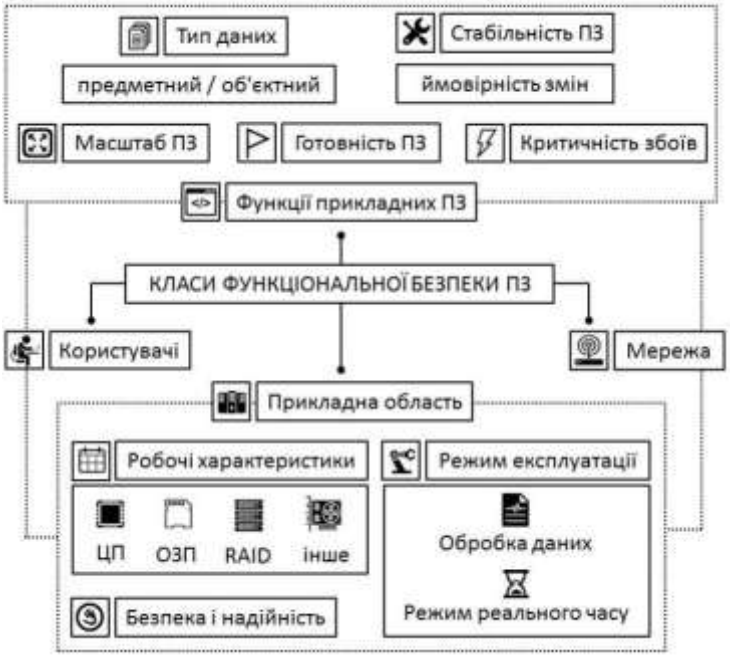


Рисунок 1.3 – Класи функціональної безпеки програмних засобів

- 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 [8]. Під час аналізу здебільшого використовують такі рівні (рис. 1.4):

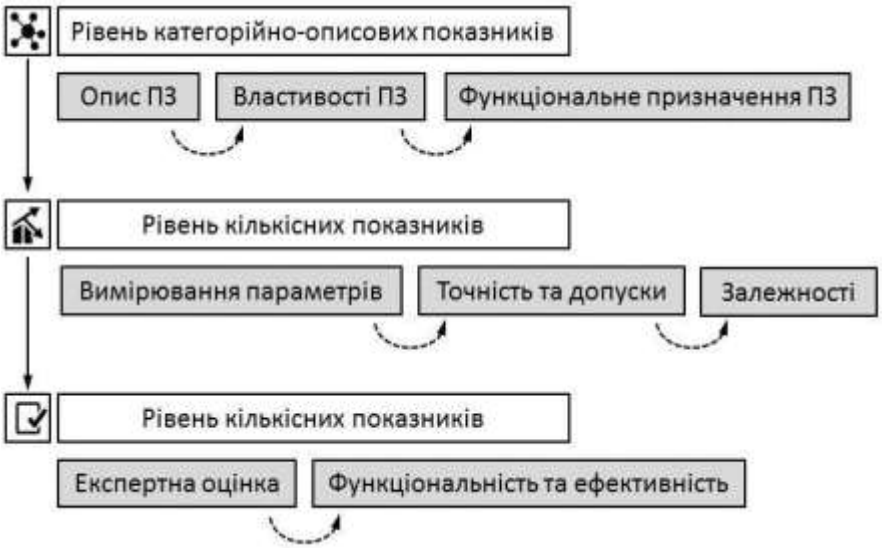


Рисунок 1.4 – Базові рівні оцінки показників програмних засобів

- 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 функціональної придатності є першим етапом визначення параметрів функціональної безпеки ПЗ та інших стандартизованих показників якості комплексу [10].

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 [11]:

- 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 [5, 6] 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 [2, 3, 4], 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 [7], 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) 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) [2]. 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 [3] (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 [6].

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 [7]. 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 [9].

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.

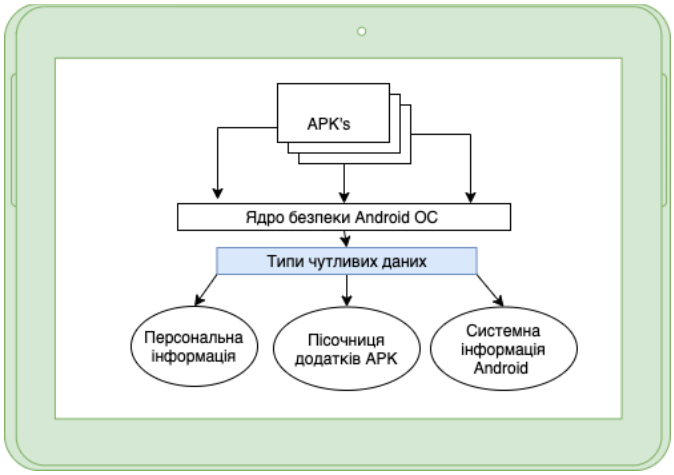


Рисунок 1.9 – Типи сховищ даних, які вимагають прямої згоди користувача

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 [5]:

- 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 [6]. 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 технології 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 [8]. 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 [5, 9, 11] 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 [5, 6]. Слід зазначити, що підходи на основі машинного навчання штучних нейромережевих алгоритмів, що дозволяють визначити рівень загрози на основі базових параметрів програмного коду [7], 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.

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 [6].

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, 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 [5]. 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 [6-8].

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) [7] 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.

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, машиннонезалежного коду низького рівня, що генерується транслятором і виконуваний інтерпретатором. 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 [8]. 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.

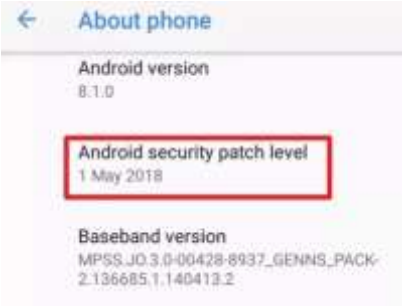


Рисунок 1.11 – Окремий патч системи безпеки ОС Android

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 [5]).

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 [6]. 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 [7], зокрема алгоритм TaintDroid [8]. 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 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 [8]. 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

redelegation), 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 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.

Висновки до 1 розділу

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.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ БЕЗПЕКИ

2.1. Сучасні способи функціонального трасування

Найбільший виклик при функціональному трасуванні API пов'язаний із зворотним інжинірингом (англ. obfuscated) [11] 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 байткод 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 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)) [14].

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 [15]. Якщо метою було просто прочитати мультимедійні файли, це вже можна зробити, просто проаналізувавши папку ./res/*.

2.2. 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 [16]. 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 [16], де наведено кілька прикладів щодо антиреверсивних методів та різних способів їх аналізу. Існує також Android API, який називається SafetyNet [17], 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 [18]. 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. 2.1 прикладі коду з кросплатформеного фраймворка Flutter [19] 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 2.1 – Приклад коду ідентифікаціїї середовища виконання

Прикладом антиємуляційної системи з широкими можливостями є також «Cuckoo» - провідна система автоматизованого аналізу шкідливих програм з відкритим кодом [12], 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.

Інший спосіб перевірити середовище виконання – перевірити властивості системи. Деякі властивості системи на емуляторі відрізняються від властивостей реальних пристроїв, наприклад, марка пристрою, апаратне забезпечення та модель. У таблиці 2.1 наведені значення системних властивостей у віртуальному середовищі, в тому числі в емуляторі.

Таблиця 2.1 – 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) 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 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 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).

2.3. 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) [14]. 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 [18], 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.).

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 2.2

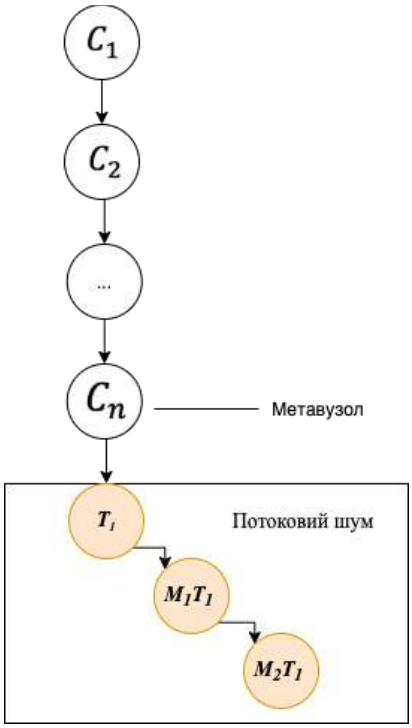


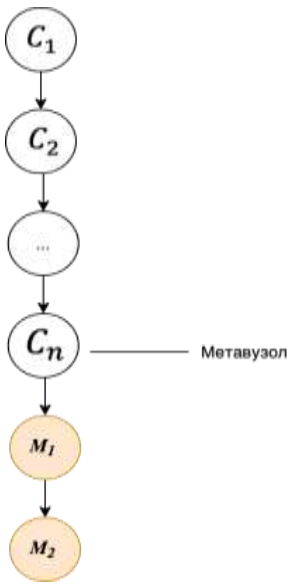
Рисунок 2.2 – Схематичне зображення розгалуження СФЛД при наявності потоків в вихідному коді

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. 2.3).

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 2.3 – 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.

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. 2.4.

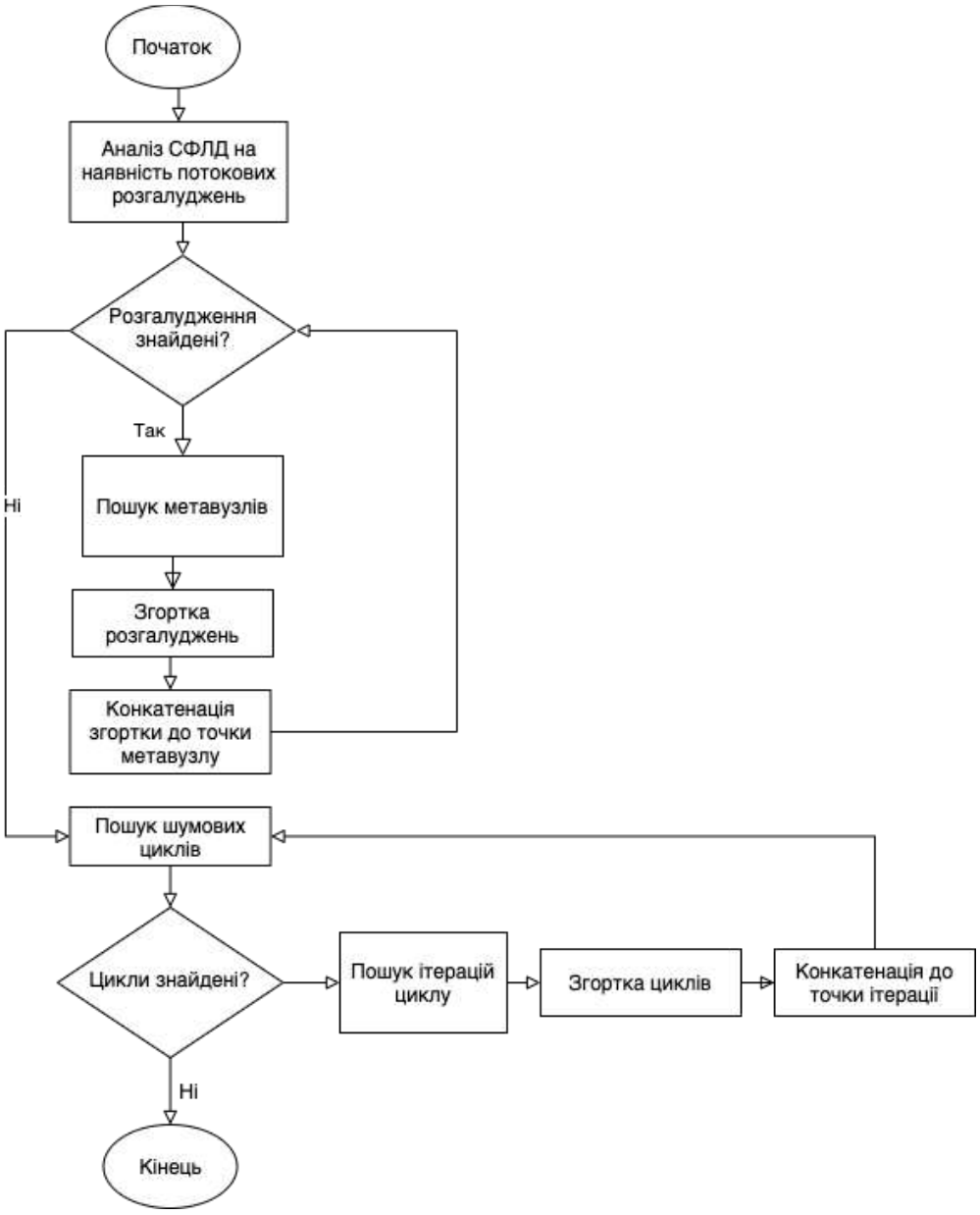


Рисунок 2.4 – Блок-схема алгоритму рекурсивного видалення потокових та циклічних «шумів» з СФЛД

2.4. 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) [12]. 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. 2.5.

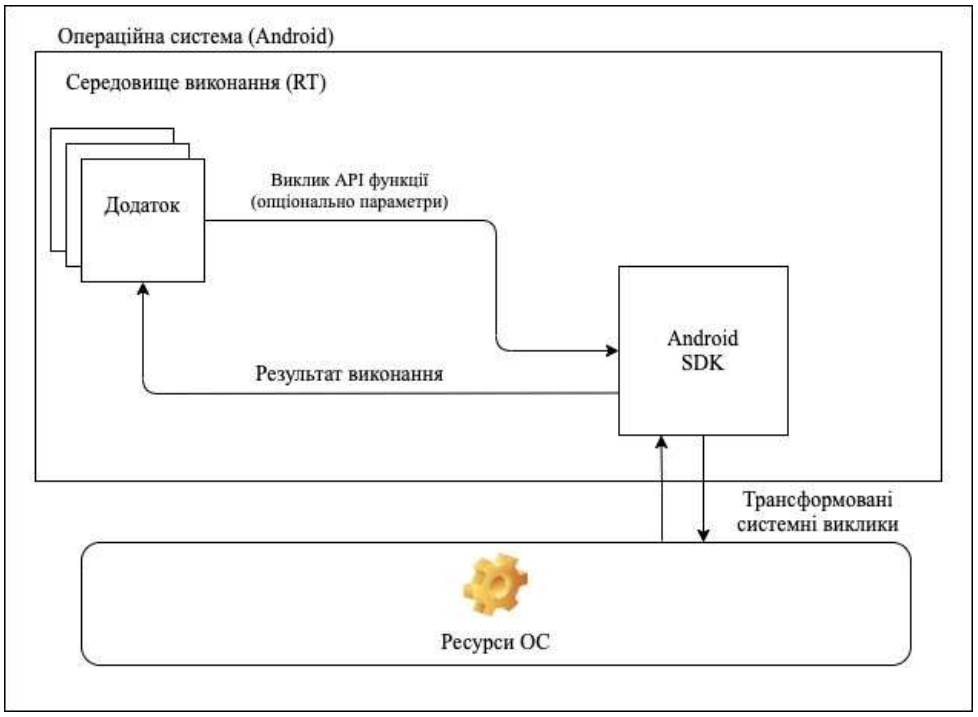


Рисунок 2.5 – Схема комунікації мобільного додатку з ресурсами ОС через виклик API функції

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 (ReadEvaluate-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 (fridaserver), 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 2.6 – 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 [14]. 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. 2.7).

```
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 2.7 – 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. 2.8 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 2.8 – 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. 2.9 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 2.9 – 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. 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.

2.5. 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. 2.10. 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.

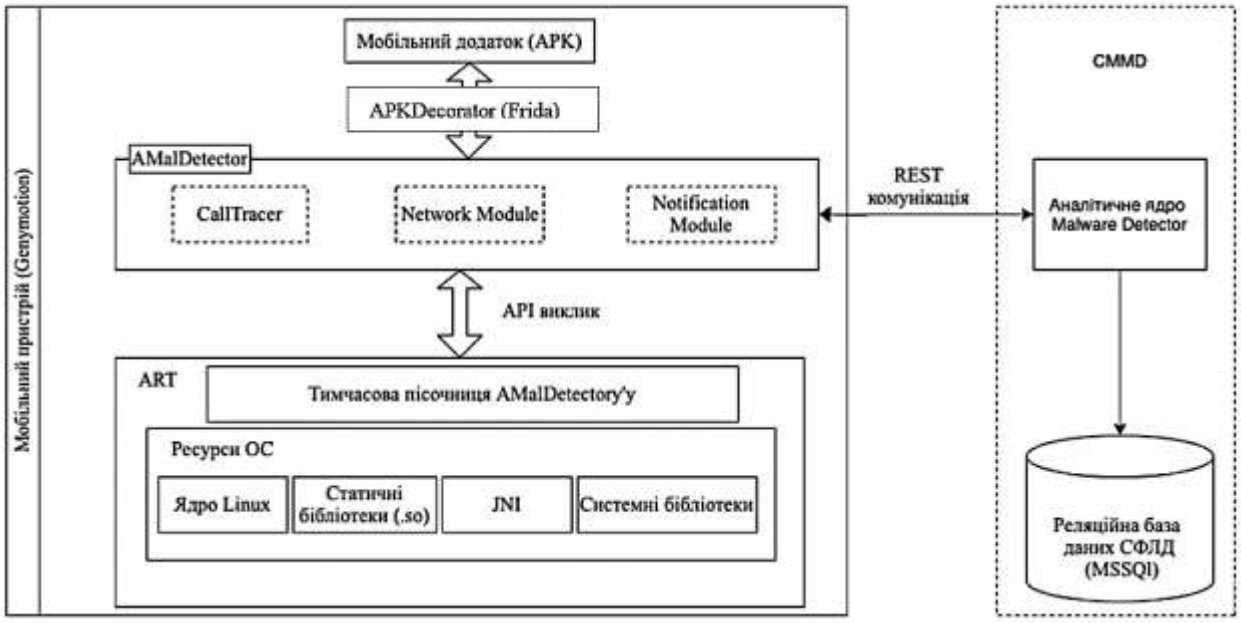


Рисунок 2.10 –Схема використання програмного комплексу

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) [15] 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. 2.11 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.

```
.class public Ledu/test/MonitoringSysCall
method internal static log saveFragment(...args)
const string p1 "packageName: edu.interceptor.logE(...args)"
static invoke {v0}, appendFragmentedCall(..args)
Unit return
.endMethod

.method public static sendAttempt(...args)
newInstance {v1} Ljava.lang.Thread
.newInstance {v2} okHttpClient.invoke(...args)
.method okHttpClient.retryPolicy(...args)
.method public static localCleanup(..args)
Unit return
```

Рисунок 2.11 – Трейс-файл шкідливого додатка

Програмний засіб CMMD реалізований у вигляді веб-додатку як Azureсервіс, налаштування якого представлено на рис. 2.12.

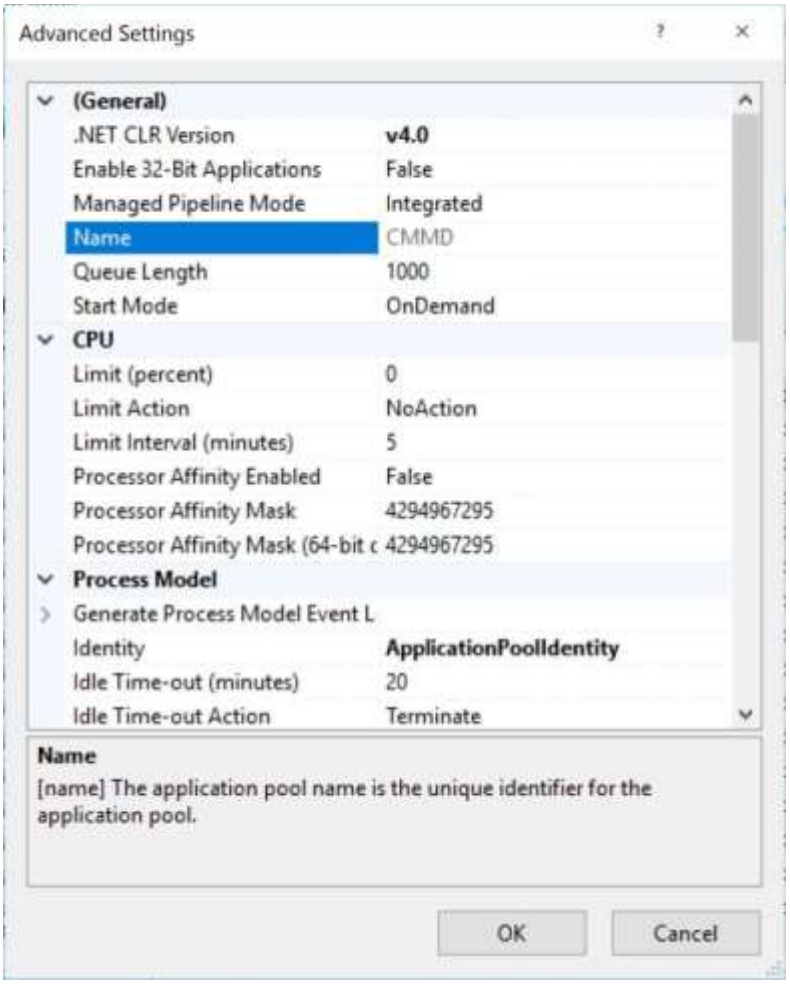


Рисунок 2.12 – Налаштування Azure-сервісу CMMD

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. 2.13.

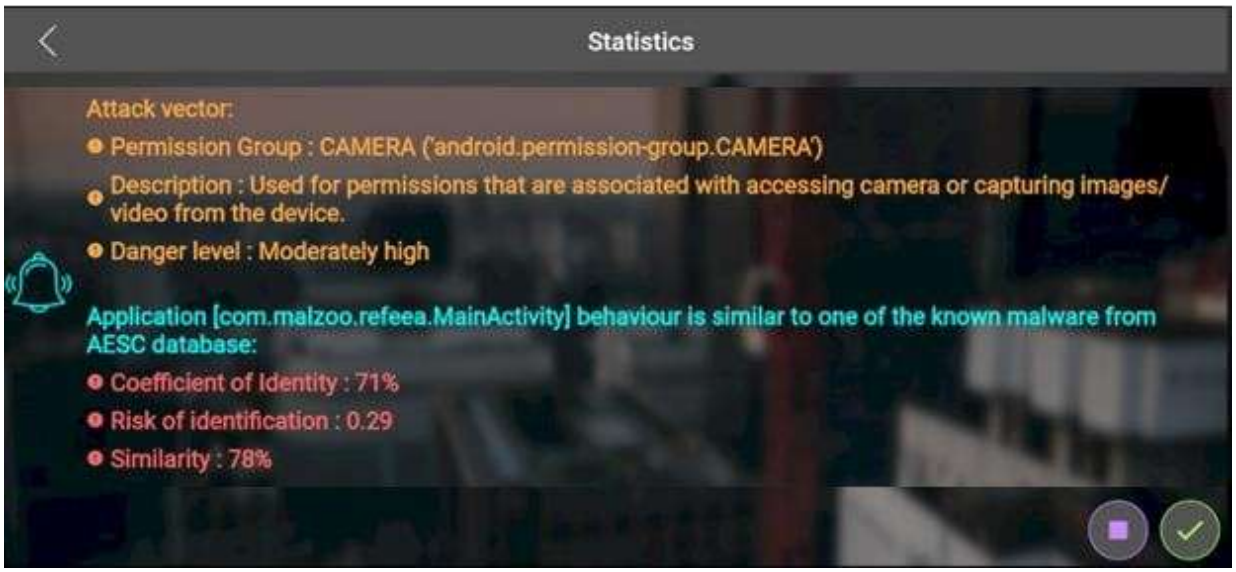


Рисунок 2.13 – Приклад нотифікації користувача

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 [16]

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.

2.6. 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 2.14 – 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. 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 2.15 показано базові поля класу-відповіді модулю CMMD.

Рисунок 2.15 – 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.

Висновки до розділу 2

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 викликів 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. 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.

ВИСНОВКИ

У дипломній роботі сформульовано та вирішене актуальне завдання з подальшого розвитку інформаційної технології забезпечення функціональної безпеки мобільних пристроїв за рахунок удосконалення існуючої системи 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. Розглянуто 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. Описана система 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. Розглянуто 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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile Statistics Report, 2019-2023 – Executive Summary.

https://www.radicati.com/wp/wpcontent/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.

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

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

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).

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.

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.

15. Android Security Internals. (2015). *Network Security*, 2015(6), 4.

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

17. Gentile, M., Summers, A.E. (2006). Random, systematic, and common cause failure: How do you manage them? *Process Safety Progress, 25*(4), 331-338.

18. Zhang, M., & Yin, H. (2016). Automatic Generation of Security-Centric Descriptions for Android Apps. *SpringerBriefs in Computer Science Android Application Security* (pp. 77-98).

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

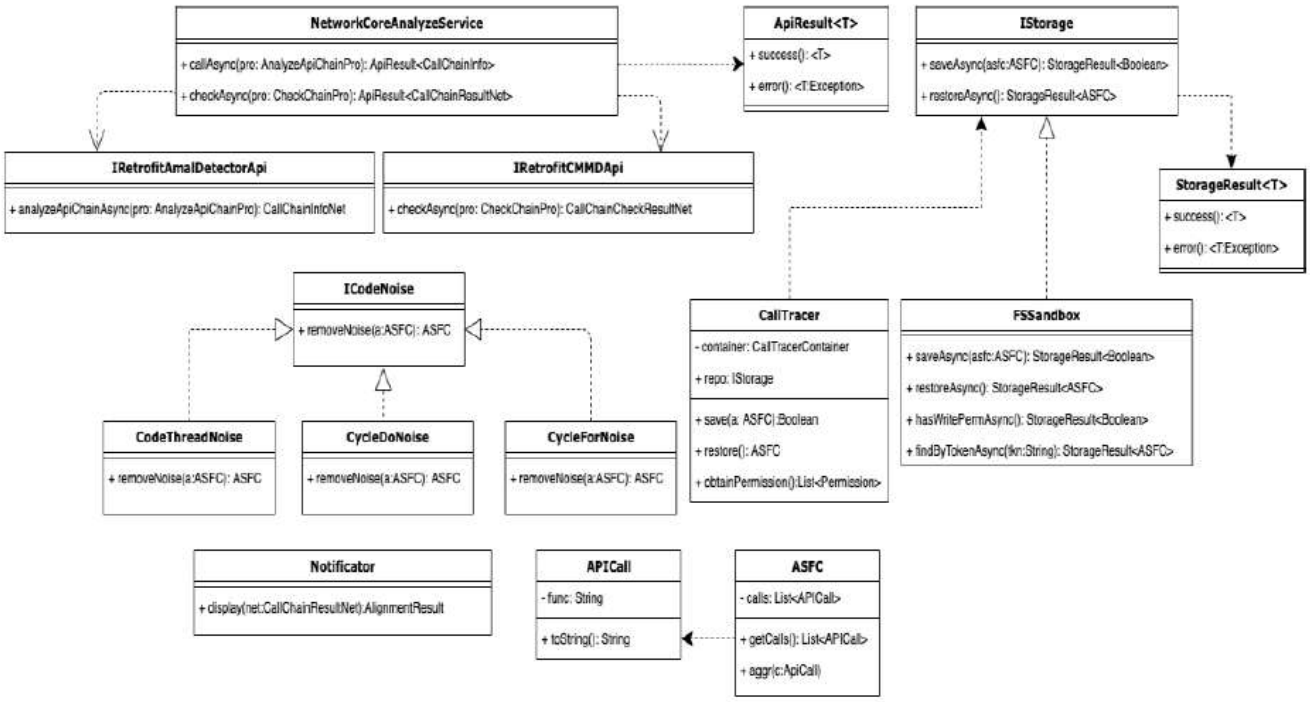
20. Clarke, S. ExVeritas Limited. *An Introduction to Functional Safety and Safety Integrity Levels.*

https://www.exveritas.com/wpcontent/uploads/2013/01/anintroductiontosafetyintegritylevels.pdf.

21. 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.

22. 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.

ДОДАТОК А



UML-ДІАГРАМИ ПРОГРАМНИХ ЗАСОБІВ

Рисунок 1 – UML-діаграма класів програмного засобу AMalDetector

Рисунок 2 – UML-діаграма класів програмного засобу CMMD