Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Поліщука Владислава Володимировича

на здобуття ступеня вищої освіти Бакалавра

Система виявлення шкідливих програм в операційних системах мобільних пристроїв

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

КРБКБ.220123.22.01.14 ПЗ

Виконав студент 4 курсу, група КБ-22-1 _____ Владислав ПОЛІЩУК
Підпис, дата Ініціали, прізвище

Керівник канд. тех. наук, доцент _____ Юрій КЛЬОЦ
Науковий ступінь, вчене звання Підпис, дата Ініціали, прізвище

Нормоконтролер д-р філософії _____ Наталія ПЕТЛЯК
Науковий ступінь, вчене звання Підпис, дата Ініціали, прізвище

До захисту допускаю:

Зав. кафедри кібербезпеки _____ Юрій КЛЬОЦ
Підпис, дата Ініціали, прізвище

10 06 2026р.

Хмельницький, 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

20 січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
Поліщука Владислава Володимировича

1 Тема роботи Система виявлення шкідливих програм в операційних системах мобільних пристроїв

Керівник роботи к.т.н, доц. зав. кафедри кібербезпеки Юрій Павлович Кльоц

Затверджено наказом ректора університету від 20 січня 2026 № 7

2 Строк подання студентом кваліфікаційної роботи на кафедру _____

3 Вихідні дані до роботи створити систему виявлення шкідливих програм в операційних системах мобільних пристроїв на основі аналізу інсталяційного пакета застосунку та пов'язаних із ним даних виконання (поведінкових ознак).

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз операційних систем для мобільних пристроїв. Розробка архітектури та алгоритму системи виявлення шкідливих програм в операційних системах мобільних пристроїв. Реалізація та тестування системи виявлення шкідливих програм в операційних системах мобільних пристроїв.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Архітектура операційної системи. Архітектура системи виявлення шкідливих програм. Результати тестування системи виявлення шкідливих програм.

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Петляк Н.С., д-р філософії, доцент кафедри кібербезпеки		

7 Дата видачі завдання 20 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	лютий	
Ознайомлення з предметною областю	лютий	
Дослідження існуючих рішень	лютий	
Постановка задачі	березень	
Визначення загальних принципів рішення задачі	березень	
Деталізація принципів рішення задачі	квітень	
Розробка проєктних рішень	квітень	
Апробація проєктних рішень	травень	
Оформлення пояснювальної записки згідно вимог	травень	
Оформлення графічної частини	червень	
Захист КР	червень	

Студент _____ Владислав ПОЛІЩУК

Керівник кваліфікаційної роботи _____ Юрій КЛЬОЦ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Система виявлення шкідливих програм в операційних системах мобільних пристроїв».

Авторка роботи: Поліщук Владислав Володимирович

Керівник роботи: Кльоц Юрій Павлович

Пояснювальна записка: 70 с., 2 додатки, 14 рис., 1 табл., 7 формул, 44 джерела.

Графічна частина: 3 плакати.

Ключові слова: шкідливе програмне забезпечення, мобільна операційна система, android, система дозволів, статичний аналіз, динамічний аналіз, поведінковий аналіз, apk-файл, виявлення загроз.

У кваліфікаційній роботі досліджено проблему виявлення шкідливого програмного забезпечення в операційних системах мобільних пристроїв. Проведено аналіз сучасного стану кіберзагроз, особливостей архітектури мобільних операційних систем (на прикладі Android), структури прикладних програм та механізмів безпеки, зокрема моделі «пісочниці», системи дозволів, міжпроцесної взаємодії та цифрового підпису коду.

Визначено основні типи мобільних шкідливих програм, способи їх поширення та характерні ознаки функціонування. Обґрунтовано необхідність застосування комплексного підходу до виявлення шкідливого програмного забезпечення, що поєднує статичний аналіз структури застосунку (маніфест, дозволи, компоненти, цифровий підпис) та динамічний аналіз його поведінки під час виконання.

Розроблено архітектуру та алгоритм функціонування системи виявлення шкідливих програм в ОС мобільних пристроїв. Реалізовано прототип системи, здійснено його тестування та оцінено ефективність виявлення потенційно небезпечних застосунків. Показано, що запропонований підхід дозволяє підвищити рівень захищеності мобільних пристроїв шляхом своєчасного виявлення нових і модифікованих загроз.

ABSTRACT

Theme of the qualification work: «Malware Detection System for Mobile Device Operating Systems».

Author of the work: Polishchuk Vladyslav Volodymyrovych.

Supervisor: Klots Yurii Pavlovych.

Explanatory note: 70 p., 2 appendices, 10 figures, 7 tables, 7 formulas, 44 references.

Graphic part: 3 posters

Keywords: malware, mobile operating system, android, permission system, static analysis, dynamic analysis, behavioral analysis, apk file, threat detection.

The qualification thesis investigates the problem of detecting malicious software in mobile operating systems. An analysis of the current state of cyber threats, the architecture of mobile operating systems (using Android as an example), the structure of application software, and security mechanisms – such as the sandbox model, permission system, inter-process communication framework, and code signing – was carried out.

The main types of mobile malware, their propagation methods, and characteristic behavioral features were identified. The necessity of applying a comprehensive approach to malware detection that combines static analysis of application structure (manifest file, permissions, components, digital signature) with dynamic analysis of application behavior during execution is substantiated.

The architecture and operating algorithm of a malware detection system for mobile operating systems were developed. A prototype of the system was implemented, tested, and evaluated in terms of its effectiveness in identifying potentially malicious applications. The results demonstrate that the proposed approach improves the security level of mobile devices by enabling timely detection of new and modified threats.

ЗМІСТ

ВСТУП

На сьогодні мобільні пристрої є невід'ємною складовою повсякденного життя сучасної людини. Вони широко застосовуються як у професійній діяльності, так і в особистому спілкуванні, для виконання різноманітних операцій та зберігання значного обсягу особистої інформації, зокрема мультимедійних файлів, документів, паролів, контактів, банківських реквізитів тощо. За статистичними даними, операційна система Android функціонує на більшості мобільних пристроїв, що свідчить про її домінуюче положення на світовому ринку. Розвинений функціонал, відкритість платформи та концентрація великого обсягу персональних даних зумовлюють підвищений інтерес з боку зловмисників, які прагнуть отримати несанкціонований доступ до цієї інформації з метою її викрадення або неправомірного використання [1,2].

Мобільні операційні системи мають розвинені механізми захисту, зокрема систему розмежування доступу, модель дозволів та засоби шифрування. Проте наявність уразливостей та можливість зловживання наданими дозволами створюють умови для проникнення шкідливого програмного забезпечення та порушення конфіденційності даних користувачів [2,3].

За даними антивірусної статистики, кількість шкідливих програм для мобільних операційних систем у 2024 році перевищила 33 мільйони, що значно більше порівняно з попередніми роками. Така динаміка свідчить про ускладнення кіберзагроз і активізацію діяльності кіберзлочинців. Зростання кількості шкідливих програм пов'язане з підвищенням популярності мобільних платформ, розширенням їх функціональних можливостей і збільшенням обсягу персональних даних, що обробляються під час використання мобільних пристроїв. Крім того, активне використання мобільних сервісів електронної комерції та онлайн-банкінгу стимулює створення нових типів атак [4,5].

Більшість антивірусних засобів функціонують на основі сигнатурного аналізу, тобто порівняння програмного коду з відомими зразками шкідливого

програмного забезпечення. Такий підхід є ефективним щодо вже відомих загроз, однак має обмеження. Якщо сигнатура нової або модифікованої шкідливої програми відсутня в базі даних, вона може залишатися невиявленою та завдавати шкоди користувачеві або системі загалом [1].

Процес формування сигнатури потребує часу та детального аналізу зразка шкідливої програми. Після її створення необхідно забезпечити оновлення антивірусних баз на великій кількості пристроїв. В умовах швидкого поширення програмного забезпечення через мережу Інтернет нові загрози можуть розповсюджуватися значно швидше, ніж відбувається оновлення засобів захисту [1].

У зв’язку з цим актуальним є розроблення системи виявлення шкідливих програм, яка поєднує аналіз структури застосунку з дослідженням його поведінки під час виконання та дозволяє виявляти підозрілу активність незалежно від наявності сигнатури в базі даних. Використання комбінованого підходу дає змогу підвищити ефективність виявлення нових і модифікованих шкідливих програм та сприяє підвищенню загального рівня захищеності мобільних операційних систем і персональних даних користувачів [1,6].

РОЗДІЛ 1. АНАЛІЗ ОПЕРАЦІЙНИХ СИСТЕМ ДЛЯ МОБІЛЬНИХ ПРИСТРОЇВ

1.1 Огляд сучасного стану захищеності операційних систем для мобільних пристроїв

Мобільні пристрої є невід'ємною частиною нашого повсякденного життя, і кількість персональних даних, які вони зберігають, постійно збільшується. На відміну від персональних комп'ютерів, ОС для мобільних пристроїв розвиваються більш динамічно. За наявними даними Вікіпедії [7], ОС Android встановлена на 78,9% пристроїв. З огляду на те, що частка пристроїв з ОС Android на ринку постійно зростає, кількість потенційних користувачів сучасних мобільних пристроїв і надалі невпинно збільшуватиметься.

При цьому сфера захисту інформації не завжди підтримується розробником і встигає за розвитком мобільних пристроїв. Наразі вони широко використовуються для виконання численних операцій, таких як робота з програмою банк-клієнт, електронною поштою, документами, хмарними сервісами, фото, відео тощо. Мобільний пристрій є сховищем персональної інформації, у разі втрати контролю над яким можна втратити всі особисті дані, фінанси тощо [8,9].

За останні два роки кількість шкідливих програм, спрямованих на ОС для мобільних пристроїв, зросла більш ніж у 10 разів і становила 33 мільйонів у 2024 році [1,6].

За цей період часу в ОС для мобільних пристроїв було виявлено [10-13]:

- 4 643 582 шкідливі інсталяційні пакети;
- 295 539 нових шкідливих програм;
- 12 100 мобільних банківських троянських програм.

Ще десять років тому на мобільних пристроях шкідливих програм практично не існувало, оскільки їх функціональні можливості були досить

обмеженими, а також були відсутні сервіси та послуги, які вони нині пропонують користувачеві. Розробники ОС для мобільних пристроїв з моменту створення враховували у своїх продуктах максимальний рівень захищеності від шкідливих програм. ОС для мобільних пристроїв не дозволяли шкідливим програмам «захоплювати» керування пристроєм.

Наразі ситуація змінилася, насамперед завдяки розширенню можливостей самих мобільних пристроїв. Сучасний мобільний пристрій – це інтегрально-модульна система: повноцінний робочий інструмент, центр розваг і засіб керування особистими фінансами. Зростання функціональних можливостей мобільних пристроїв вимагає від зловмисників розроблення нових шкідливих програм, а також витончених способів їх поширення та зараження [10-13].

З 2022 року кількість мобільних шкідливих програм зросла більш ніж на порядок і станом на кінець 2024 року таким чином перевищила 33 мільйони [4,5].

Показово змінюється розподіл шкідливих програм за типами (рис. 1.1): троянські програми, здатні надсилати SMS-повідомлення, та експлойти, які використовують дефекти алгоритму написаного коду, поступаються «дорогою» рекламним шкідливим програмам і троянським програмам, спрямованим на банківські продукти.

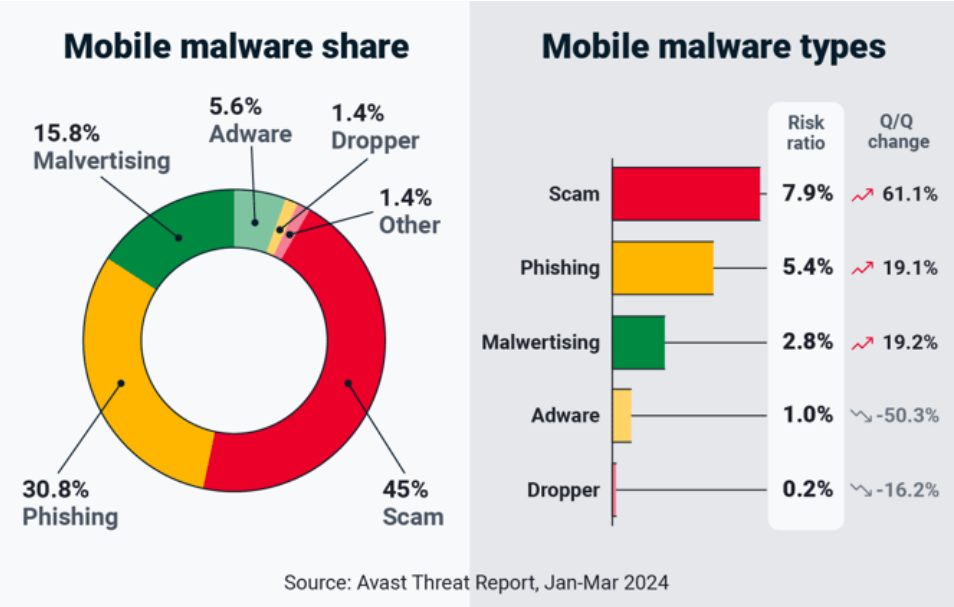


Рис. 1.1 – Розподіл мобільних шкідливих програм за функціями

При цьому зменшення частки шкідливих програм певного типу не свідчить про те, що вони не застосовуються і не розробляються, про що говорить зростання загальної кількості шкідливих програм.

Під час роботи з мобільним пристроєм користувач не завжди замислюється над можливою втратою важливої для нього інформації: втрата пристрою, його продаж тощо – усі ці та інші чинники не є критичними порівняно з навмисними атаками на мобільний пристрій, оскільки в цьому випадку ризик втрати конфіденційної інформації зростає.

На тлі аналізу загальної ситуації у сфері інформаційної безпеки, що стосується шкідливих програм в ОС для мобільних пристроїв, можна зробити висновок, що зростання популярності таких ОС і їхніх функціональних можливостей, розширення функціоналу призводить до кількісного та якісного зростання шкідливих програм, про що свідчить наведена статистика [10-14].

1.2 Аналіз архітектури та компонентів операційних систем для мобільних пристроїв

ОС для мобільних пристроїв являє собою значною мірою «усічене» за функціональністю ядро ОС Linux, а також вбудоване прикладне і системне програмне забезпечення [15,16].

З точки зору архітектури вона являє собою повний програмний стек, який складається з чотирьох рівнів, описаних у таблиці 1.1.

В основі архітектури лежить «усічене» ядро ОС Linux, яке слугує проміжним рівнем між апаратним і програмним забезпеченням. Воно забезпечує функціонування всієї системи та надає системні служби ядра: керування пам’яттю, енергосистемою і процесами, забезпечує безпеку, роботу з мережею та драйверами [15,16].

Рівнем вище розташований набір бібліотек і середовище виконання [15,16]. Бібліотеки виконують такі функції:

- надають реалізовані алгоритми для вищерозташованих рівнів;

– забезпечують підтримку файлових форматів;

– здійснюють кодування і декодування інформації, наприклад, мультимедійні кодеки;

– виконують організацію відображення графіки тощо.

Таблиця 1.1 – Чотири рівня архітектури

Назва	**Опис**
Базовий рівень (ядро ОС Linux)	Рівень абстракції між апаратним рівнем і програмним стеком
Набір бібліотек і середовище виконання	Забезпечує основний базовий функціонал для роботи програм, містить віртуальну машину та базові бібліотеки Java, необхідні для запуску програм
Рівень каркаса програм	Забезпечує розробникам доступ до інтерфейсу програмування, що надається компонентами системи рівня бібліотек
Рівень програм	Набір попередньо встановлених базових системних програм

На рисунку 1.2 представлено архітектуру ОС для мобільних пристроїв (на прикладі ОС Android).

Бібліотеки реалізовані мовою C/C++ і скомпільовані під конкретне апаратне забезпечення мобільного пристрою, з яким вони постачаються виробником у попередньо встановленому вигляді. Перелік і опис основних типів бібліотек наведено в таблиці 1.2.

Середовище виконання включає бібліотеки ядра, за рахунок яких забезпечується значна частина низькорівневої функціональності, доступної бібліотекам ядра мови Java, а також віртуальну машину, що дозволяє запускати програми. Кожна програма запускається у власному екземплярі віртуальної

машини, таким чином забезпечується ізоляція працюючих програм в ОС для мобільних пристроїв одна від одної [17,18].

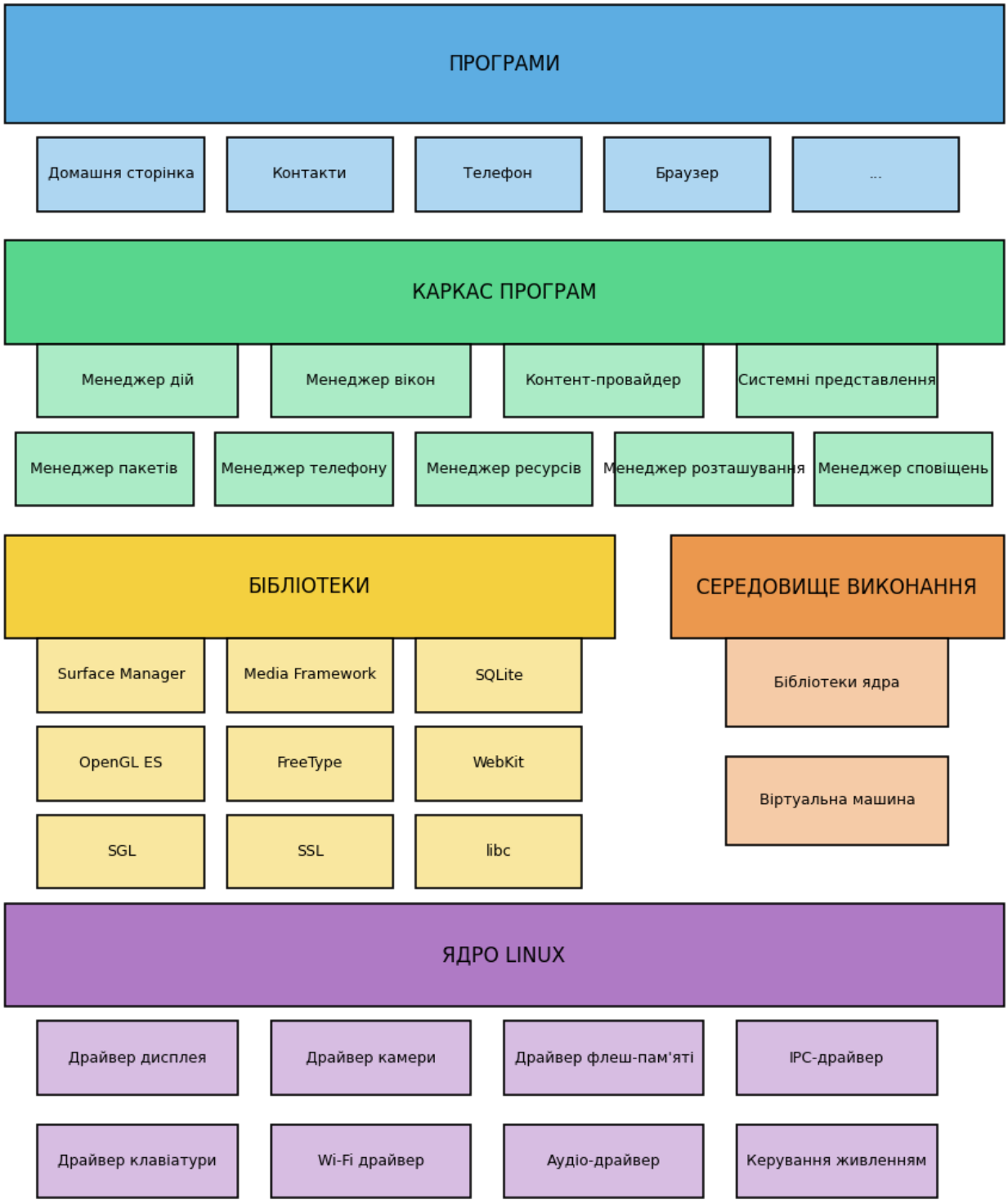


Рис. 1.2 – Архітектура ОС (на прикладі ОС Android)

Для виконання програми на віртуальній машині здійснюється компіляція Java-класів у виконувані файли з розширенням .dex за допомогою інструмента dx, що входить до складу середовища розробки програм (Android SDK). DEX (Dalvik

EXecutable) – формат виконуваних файлів для віртуальної машини, оптимізований для використання мінімального обсягу пам'яті. Під час використання IDE Eclipse і плагіна під назвою «засоби розробки програм» компіляція класів Java у формат .dex відбувається автоматично [17,18].

Таблиця 1.2 – Перелік основник бібліотек

Назва	**Опис**
Менеджер поверхонь	Є композитним менеджером вікон. Команди, що надходять для організації відображення графіки, збираються в позакадровий буфер, накопичуються в ньому, утворюючи певну композицію, а потім виводяться на екран мобільного пристрою. Ця бібліотека дозволяє створювати різні ефекти, прозорість вікон і плавні переходи.
Каркас мультимедіа Java-програм	Бібліотеки, реалізовані на основі кодека PacketVideo OpenCORE. Необхідні для запису та відтворення аудіо- і відеоконтенту, а також для виведення зображень.
SQLite	Легка та продуктивна реляційна система керування базами даних, використовується як основний «движок» для роботи з базами даних.
3D-бібліотеки	Використовуються для високорівнево оптимізованого відображення 3D-графіки; за наявності технічної можливості задіюється апаратне прискорення. Бібліотеки реалізовані на основі програмного інтерфейсу OpenGL|ES. OpenGL|ES (OpenGL for Embedded Systems) – підмножина графічного програмного інтерфейсу, адаптована для роботи у вбудованих системах.
FreeType	Бібліотека для роботи з растровими зображеннями, для растеризації (перетворення векторного зображення на піксельне) шрифтів і виконання операцій над ними.
LibWebCore	Бібліотеки ядра вбудованого браузера.

Таблиця 1.2 – Перелік основник бібліотек (кінець)

Назва	Опис
SGL (графічне ядро «Skia»)	Графічна бібліотека, що забезпечує відображення двовимірної графіки та є основою для рендерингу елементів інтерфейсу користувача.
SSL	Бібліотеки для підтримки однойменного криптографічного протоколу SSL, що забезпечує захищену передачу даних мережею.
Libc	Стандартна бібліотека мови С, а саме її реалізація, налаштована для роботи на пристроях на базі ОС Linux.

Архітектура середовища виконання реалізована таким чином, що функціонування програм здійснюється суворо в межах оточення віртуальної машини, що дозволяє захистити ядро від можливої загрози з боку інших її складових. Тому якщо спрацює код із помилками або шкідливе програмне забезпечення, вони не зможуть вивести з ладу ОС [17,18].

Рівнем вище розташований каркас програм, особливості його архітектури дозволяють будь-якій програмі використовувати вже реалізовані можливості інших програм, до яких надано доступ. У таблиці 1.3 наведено опис компонентів каркаса програм.

Каркас програм надає у розпорядження програм додатковий допоміжний функціонал, який реалізує принцип багаторазового використання компонентів програм у межах політики безпеки.

Рівень програм є найближчим до користувача рівнем. На цьому рівні користувач взаємодіє зі своїм мобільним пристроєм, а також тут представлений набір встановлених базових програм.

Для інсталяції прикладних програм користувач може скористатися вбудованим хмарним сервісом із каталогом програм, який дозволяє здійснювати їх придбання та встановлення. Щоб інсталювати прикладну програму в ОС для мобільних пристроїв, створюється файл із розширенням .apk, який містить

виконувані файли та допоміжні компоненти (файли з даними та ресурсами). Після інсталяції прикладної програми кожна програма функціонує у власному ізольованому екземплярі віртуальної машини [17-19].

Таблиця 1.3 – Компоненти в складі каркасу програм

Назва	**Опис**
Набір представлень	Компонент, необхідний для створення візуальних складових програм (списків, текстових полів, таблиць).
Контент-провайдери (джерела даних)	Здійснюють керування даними, які одні програми відкривають для інших, щоб останні могли використовувати їх у своїй роботі.
Менеджер ресурсів	Містить функції для доступу до ресурсів програми (рядків, значень, графіки, макетів користувацького інтерфейсу тощо).
Менеджер сповіщень	Реалізує можливість для програм відображати повідомлення для користувача в рядку стану.
Менеджер дій	Керує життєвими циклами програм, зберігає історію роботи з діями, надає систему навігації між діями.
Менеджер місцезнаходження	Дозволяє програмам періодично отримувати оновлені дані про поточне географічне розташування мобільного пристрою.

Більшість мобільних пристроїв містить такі апаратні складові [15-19]:

– процесор, продуктивність якого обмежена з метою зменшення тепловиділення;

– мікросхеми пам’яті;

– чип накопичувача і – в окремих моделях – слот для додаткового накопичувача;

– акумуляторну батарею;

– графічний процесор;

- звуковий процесор і динаміки;
- сенсори, серед яких акселерометри, компас, світлочутливі датчики тощо;
- GPS-приймач;
- антену Wi-Fi і – в окремих моделях – стільникову антену;
- чип Bluetooth;
- FM-тюнер;
- камеру (одну або більше).

Таким чином, виконавши аналіз архітектури та компонентів ОС для мобільних пристроїв (на прикладі ОС Android), можна зробити висновок, що вона являє собою складно організовану структуру, яка функціонує за визначеними алгоритмами. В її основі лежить «усічене» ядро ОС Linux, яке керує роботою ОС для мобільних пристроїв. Каркас програм, програми, середовище виконання та бібліотеки є її компонентами і формують архітектуру ОС для мобільних пристроїв. Разом із ядром вони забезпечують виконання всього функціоналу, а також захист інформації за допомогою вбудованих засобів безпеки. Складність системи зумовлена наявністю широкого функціоналу та високими вимогами до роботи мобільного пристрою [20,21].

1.3 Аналіз структури та складових прикладних програм для мобільних пристроїв

Прикладна програма ОС для мобільних пристроїв (на прикладі ОС Android) має структуру, представлену на рисунку 1.3 [22].

ФАЙЛ МАНІФЕСТУ складається з переліку апаратних і програмних вимог, необхідних для запуску та роботи прикладної програми. Він має назву AndroidManifest.xml і містить усю інформацію про прикладну програму, яку надає ОС під час звернення до неї. Кожна прикладна програма має власний файл маніфесту, який являє собою XML-код. Він необхідний для того, щоб [22-26]:

– надавати ім’я Java-пакета прикладної програми та надалі виступати як унікальний ідентифікатор;

– описувати компоненти, що входять до складу прикладної програми;

– містити перелік усіх необхідних дозволів для доступу до програмних і апаратних складових;

– оголошувати дозволи, які інші прикладні програми повинні мати для взаємодії з компонентами цієї програми;

– виконувати інші запити до бібліотек тощо.

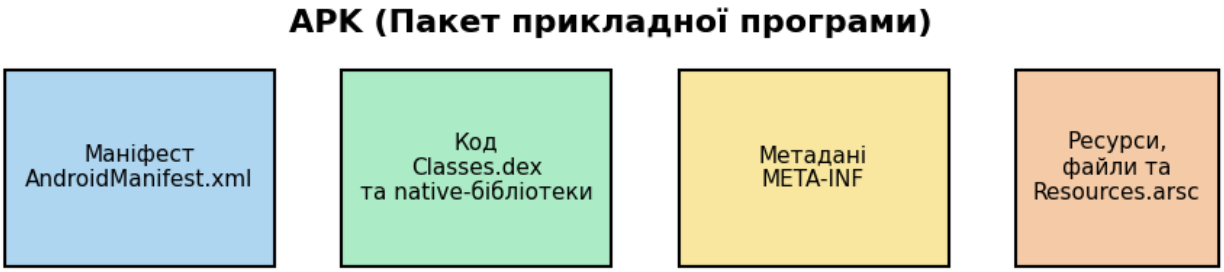


Рис.1.3 – Структура прикладної програми ОС Android

Елемент ***manifest*** є кореневим елементом файла маніфесту [26]. У своєму складі має чотири атрибути:

– ***xmlns:android*** – цей атрибут визначає простір імен;

– ***package*** – визначає унікальне ім’я пакета прикладної програми, яке регламентується під час його створення;

– ***android:versionCode*** – містить внутрішній номер версії, необхідний для порівняння версій прикладних програм;

– ***android:versionName*** – містить номер версії прикладної програми, який відображається для користувача.

Елемент ***permission*** [27] оголошує дозволи, необхідні для обмеження доступу до певних компонентів або функціональності цієї прикладної програми. У цьому елементі міститься опис прав, які запитують інші прикладні програми для отримання доступу до потрібної програми, а також прикладна програма може захистити власні компоненти (служби, контент-провайдери) шляхом

використання дозволів. Вона може використовувати будь-які системні дозволи або дозволи, оголошені іншими програмами, а також визначати власні дозволи.

Елемент ***uses-permission*** [27] здійснює запит переліку дозволів, які будуть надані прикладній програмі для коректного функціонування. Вони надаються і запитуються на етапі інсталяції прикладної програми. Наприклад, такі дозволи:

– INTERNET – запит прав на доступ до Інтернету;

– READ_CONTACTS – запит прав на читання, але не на запис даних з адресної книги користувача;

– WRITE_CONTACTS – запит прав на запис, але не на читання даних з адресної книги користувача;

– RECEIVE_SMS – запит прав на обробку вхідних SMS-повідомлень;

– ACCESS_COARSE_LOCATION – запит прав на визначення приблизного місцезнаходження за допомогою веж стільникового зв'язку або точок доступу Wi-Fi;

– ACCESS_FINE_LOCATION – запит прав на точне визначення місцезнаходження за допомогою навігації тощо.

Елемент ***permission-tree*** [27] оголошує базове ім'я для дерева дозволів, але не сам дозвіл, а лише простір імен, у межах якого можуть бути розміщені подальші дозволи.

Елемент ***permission-group*** [27] визначає ім'я для набору логічно пов'язаних дозволів. Цей елемент оголошує лише категорію, до якої можуть бути віднесені дозволи.

Елемент ***instrumentation*** [27] оголошує об'єкт instrumentation, який надає можливість контролювати взаємодію програм в ОС.

Елемент ***uses-sdk*** [27] указує необхідну сумісність прикладної програми з певною версією ОС.

Елемент ***uses-configuration*** [27] указує необхідну для прикладної програми апаратну та програмну конфігурацію мобільного пристрою.

Елемент ***uses-feature*** [27] оголошує певні параметри функціональності, необхідні для роботи прикладної програми. Отже, прикладна програма, яка не має відповідних функціональних можливостей, не буде інстальована.

Елемент ***supports-screens*** [27] задає роздільну здатність екрана, необхідну для коректного відображення прикладних програм в ОС для мобільного пристрою.

Елемент ***application*** [27,28] – один із основних елементів файла маніфесту, що містить опис компонентів прикладних програм, доступних у пакеті: стилі, значок тощо. Містить дочірні елементи, які оголошують кожен із компонентів, що входять до складу прикладних програм.

Елемент ***activity*** [27,28] оголошує активність (дію) і містить низку атрибутів, що визначають дозволи, орієнтацію екрана тощо.

Елемент ***intent-filter*** [27,28] визначає типи дій, за допомогою яких викликаються компоненти прикладної програми (вибір фотографії, здійснення дзвінка тощо) і на які можуть відповідати активність (вікно програми), служба або приймач дій. Фільтр дій оголошує можливості батьківського компонента: що можуть виконувати активність або служба та які типи повідомлень приймач може обробляти. Фільтр дій надає компонентам-клієнтам можливість отримувати дії оголошеного типу, фільтрувати ті, що не мають значення для компонента, і містить такі дочірні елементи:

- елемент ***action*** – додає дію до фільтра дій;
- елемент ***category*** – визначає категорію компонента, яку має обробити дія;
- елемент ***data*** – додає специфікацію даних до фільтра дій.

Елемент ***meta-data*** [27,28] визначає пару «ім’я–значення» для додаткових довільних даних, якими можна забезпечити батьківський компонент.

Елемент ***activity-alias*** [27,28] – це аналог (псевдонім) Activity.

Елемент ***service*** [27,28] оголошує службу як один із компонентів програми.

Елемент ***receiver*** [27,28] оголошує приймач широкомовних повідомлень як один із компонентів програми. Приймачі широкомовних повідомлень надають

можливість прикладній програмі отримувати дії, передані системою або іншими програмами, навіть коли інші компоненти програми не працюють.

Елемент ***provider*** [27,28] оголошує контент-провайдера (джерело даних) для керування доступом до баз даних. Елемент ***grant-uri-permission*** є дочірнім елементом для provider і визначає, кому можуть бути надані дозволи на підмножини даних контент-провайдера. Елемент ***path-permission*** – також дочірній елемент для provider – визначає шлях і необхідні дозволи для певної підмножини даних у межах постачальника оперативної інформації.

Елемент ***uses-library*** [27,28] визначає загальнодоступну бібліотеку, з якою має бути скомпонована програма, і вказує системі на необхідність включення коду бібліотеки до завантажувача класів пакета програми. Кожен проєкт за замовчуванням пов'язаний із бібліотеками, що містять основні пакети для збирання програми. Проте деякі пакети розміщені в окремих бібліотеках, які автоматично не компонуються з програмою. Якщо програма використовує пакети з цих бібліотек або з бібліотек сторонніх розробників, необхідно виконати явне зв'язування з ними – файл маніфесту обов'язково має містити окремий елемент uses-library.

КОД ПРОГРАМИ, що виконується в ОС, представлений у вигляді байт-коду, скомпільованого для середовища виконання.

МЕТАДАНІ містять контрольну суму та електронний цифровий підпис для контролю цілісності програми.

РЕСУРСИ ТА ФАЙЛИ ПРОГРАМИ – це зображення, мультимедійні файли та інші ресурси.

Код програми складається з таких компонентів [29,30]:

– Java-класів, які є підкласами основних класів із середовища розробки (Android SDK) (View, Activity, ContentProvider, Service, BroadcastReceiver, Intent);

– Java-класів, які не мають батьківських класів у середовищі розробки (Android SDK).

Кожна програма запускається у власному процесі, тому вона ізольована від інших запущених програм, і некоректно працююча програма не може безперешкодно зашкодити іншим. Водночас однією з основних особливостей

програм ОС є можливість використовувати компоненти інших програм за умови надання відповідних прав. Наприклад, якщо необхідний компонент для відображення тексту вже реалізований в іншій програмі, його можна використати повторно. У такому випадку програма не копіює потрібний код і не створює на нього посилання, а надсилає запит на виконання частини коду іншої програми, у якій наявний необхідний компонент [29,30].

Процес інсталяції прикладної програми включає такі етапи [31].

Пошук прикладної програми. Програма може бути отримана з хмарного сервісу, з невідомого джерела, передана через персональний комп'ютер або інші носії інформації.

Існує три способи інсталяції прикладних програм в ОС для мобільних пристроїв [31]:

– за допомогою вбудованої системної програми – пакетного інсталятора;
– через хмарний сервіс або інше джерело мережі Інтернет (каталог програм);
– за допомогою команди adb install, яку використовують розробники для встановлення програм із персонального комп'ютера.

Запит дозволів. Під час інсталяції програма запитує в користувача перелік необхідних дозволів, а також здійснюється перевірка апаратних і програмних можливостей пристрою, необхідних для її роботи.

Встановлення та запуск. Виконується інсталяція коду прикладної програми в ОС для мобільних пристроїв. Програма починає функціонувати в ізольованому середовищі, кожній програмі призначається власний каталог і користувач.

Таким чином, після огляду та аналізу структури прикладних програм і процесу їх інсталяції в ОС для мобільних пристроїв можна зробити висновок, що прикладна програма не запускається і не функціонує довільно без наявності відповідних прав і програмно-апаратних ресурсів з боку ОС. Ключову роль у цьому процесі відіграє користувач, який надає дозволи на запитувані ресурси та права, необхідні для роботи програми, а також файл маніфесту, що містить усі вимоги для її функціонування [31].

Отже, цей файл потребує детального аналізу для побудови повної схеми функціонування прикладної програми, зокрема визначення її призначення, впливу на ОС для мобільних пристроїв та дій, які вона виконує під час роботи.

1.4 Огляд і аналіз загроз та вразливостей в операційних системах мобільних пристроїв

Архітектура ОС для мобільних пристроїв має інтегрально-модульну структуру, а також вбудовані засоби захисту, які функціонують таким чином, що всі програми мають обмежені права доступу й не мають доступу до захищених даних інших програм.

Загрозою для мобільних ОС є спеціальні програми-експлойти [32], які дозволяють отримати права привілейованого користувача, що, у свою чергу, надає можливість виконувати будь-які дії на апаратному та програмному рівнях: редагувати системні файли, керувати роботою апаратних компонентів мобільного пристрою, створювати та змінювати ОС без обмежень. Цей тип найбільш небезпечних шкідливих програм здатний надати зловмиснику повну інформацію про користувача, пристрій, місцезнаходження та дозволяє здійснювати віддалене керування мобільним пристроєм. Зловмисники використовують цей вид загроз, а також вразливості для реалізації комплексних атак і отримання прав доступу найвищого рівня, щоб безперешкодно встановлювати програми без дозволу користувача. Розробка таких шкідливих програм потребує значного часу, високої кваліфікації спеціаліста та наявності актуальної вразливості в алгоритмах ОС. Зазначені чинники зумовлюють низьку поширеність цього типу шкідливого програмного забезпечення.

Загалом шкідливі програми рідко використовують експлойти [32]. Вони вводять користувача в оману та намагаються спровокувати його на виконання необхідних дій – надати шкідливій програмі всі можливі дозволи на використання ресурсів, потрібних для її функціонування. Таким чином, надаючи шкідливій

програмі необхідні можливості, нічого не підозрюючий користувач добровільно дозволяє їй здійснювати шкідливі дії. Такі програми можуть поширюватися як через відомі джерела – хмарні сервіси, так і через будь-які інші ресурси.

Однією з важливих складових системи безпеки ОС для мобільних пристроїв є система дозволів, яка водночас є і потенційною вразливістю [33]. Під час інсталяції будь-якої програми користувачеві надається перелік усіх дозволів на доступ до програмних і апаратних сервісів, які будуть доступні програмі відповідно до її призначення. Після завершення інсталяції програма може виконувати свої дії з наданими правами без участі користувача, зокрема у фоновому режимі. Унаслідок цього користувач може не бути поінформованим про її діяльність. Система дозволів створена для забезпечення безпеки та контролю встановлюваних програм, однак більшість користувачів часто не звертають уваги на перелік запитуваних прав. Зазначена вразливість є однією з найсерйозніших, оскільки саме на етапі встановлення можна запобігти інсталяції шкідливої програми та уникнути можливих збитків [33].

Загрозу може становити також використання модифікованих версій ОС для мобільних пристроїв, які можуть бути змінені будь-яким користувачем для власних потреб [34]. У такі ОС можуть бути вбудовані шкідливі програми. Якщо програму підписано цифровим підписом еталонного образу системи, вона отримує ті самі права, що й сама система, у якій працює. В межах ОС Android з відкритим вихідним кодом підписи для образів є приватними, тому подібний сценарій можливий, наприклад, у разі викрадення відповідного підпису. Подібний спосіб зараження застосовувався шкідливою програмою Android.SmsHider, яка могла у фоновому режимі встановлювати троянську програму з APK-файлу на пристроях із модифікованими ОС.

Вбудовані системні програми також можуть містити вразливості. Наприклад, уразливості вбудованого браузера можуть дозволяти шкідливим програмам виконувати код і отримувати доступ до захищених даних браузера [35,36].

Певною вразливістю є й відкритість ОС для мобільних пристроїв [35,36]. Оскільки вихідний код перебуває у відкритому доступі, зловмисники можуть активно аналізувати його з метою пошуку вразливостей і помилок для подальшого використання. Серед основних вразливостей виділяють [35,36]:

– можливість встановлення програм як з офіційного каталогу, так і з будь-якого іншого доступного джерела. Програми з довірених хмарних сервісів проходять перевірку антивірусним програмним забезпеченням, тому ймовірність появи шкідливих програм там нижча, ніж у невідомих джерелах;

– можливість для будь-якого користувача чи розробника розміщувати власні програми в офіційному каталозі. Відкритість ОС Android дозволяє зловмисникам поширювати шкідливе ПЗ;

– відсутність або несвоєчасність системних оновлень з боку розробника.

Важливу роль відіграє людський фактор [33,37]: шкідливі програми поширюються через рекламу в інших програмах із використанням маніпулятивних повідомлень («Необхідно терміново оновити систему», «Ваша версія програми застаріла», «Негайно встановіть оновлення» тощо), а також через спам-розсилки SMS. Наприклад, Android.Crusewind функціонує таким чином: користувач отримує SMS-повідомлення з інформацією про необхідність оновлення налаштувань доступу до Інтернету та посиланням. Після переходу за цим посиланням користувач завантажує файл, що містить шкідливу програму.

Існують різні типи шкідливих програм [37,38].

Троянська програма, що надсилає SMS-повідомлення – програми цього типу надсилають SMS-повідомлення з підвищеною тарифікацією на короткі номери, внаслідок чого частина вартості таких повідомлень перераховується на рахунок зловмисника. Подібні програми практично не відрізняються одна від одної, за винятком незначних змін в інтерфейсі та різноманітності коротких номерів, на які здійснюється відправлення SMS-повідомлень. Здебільшого програми цього типу поширюються під виглядом популярних застосунків, таких як браузер Opera Mini, ICQ, Skype, Angry Birds тощо, при цьому використовується

відповідна іконка, характерна для цієї популярної програми. Як приклад можна навести шкідливі програми сімейства Android.SmsSend.

Звичайні троянські програми – здійснюють збір конфіденційної інформації про користувача, додають закладки в браузер, виконують різноманітні команди, що надходять від зловмисників, надсилають SMS-повідомлення, встановлюють інші програми тощо. Для реалізації можливості встановлення програм без викликання підозри з боку користувача необхідні права привілейованого користувача – тобто права, з якими працює ядро системи. Прикладами таких шкідливих програм є Android.Wukong, Android.DreamExploid, Android.Geinimi, Android.Spy та інші.

Комерційні шпигунські програми – ці програми використовуються для стеження за діями користувачів: здійснюють перехоплення вхідних і вихідних SMS-повідомлень, дзвінків, записів із диктофона, відстеження місцезнаходження пристрою, збір даних із браузера тощо. З огляду на те, що більшість таких програм потребують початкового налаштування та ручної інсталяції, вони становлять суттєву загрозу, оскільки після встановлення працюють у фоновому режимі та не створюють ярлика серед інших програм. Виявити їхню присутність можна лише за певними ознаками, зокрема через перегляд системного меню зі списком програм або аналіз системних процесів.

Рекламні модулі – програми, що містять у своєму складі сервісний модуль, який функціонує у фоновому режимі – після закриття самої програми – та відображає рекламу в області сповіщень. Цей тип рекламних повідомлень часто використовується для публікації повідомлень, замаскованих під системні, наприклад: «Потрібне термінове оновлення системи» або «З’явилася нова версія програми», що може слугувати каналом поширення шкідливого програмного забезпечення. У хмарних сервісах міститься велика кількість програм із подібним підходом до відображення реклами. Прикладами є Adware.Airpush, Adware.Leadbolt, Adware.Startapp.

Інші – шкідливі програми, що поєднують у собі різні функції, описані вище.

1.5 Постановка задачі

Проведений у першому розділі аналіз архітектури ОС для мобільних пристроїв, структури прикладних програм та існуючих загроз показав, що незважаючи на наявність вбудованих механізмів ізоляції процесів, системи дозволів та цифрового підпису програм, проблема ефективного виявлення шкідливого програмного забезпечення в мобільних ОС залишається актуальною.

Розглянута модель безпеки Android базується на ізоляції програм у межах окремих процесів та контролі доступу через систему дозволів. Проте аналіз структури прикладних програм (файлу AndroidManifest.xml, компонентів application, activity, service, receiver, provider, а також запитуваних дозволів) показав, що саме ці механізми можуть використовуватися зловмисниками для реалізації шкідливої функціональності. Більшість сучасних шкідливих програм не експлуатують уразливості ядра, а вводять користувача в оману та отримують необхідні дозволи легальним шляхом під час інсталяції.

Крім того, відкритість платформи, можливість встановлення програм із різних джерел, використання обфускації коду, динамічне завантаження компонентів і приховане функціонування у фоновому режимі значно ускладнюють виявлення шкідливого ПЗ традиційними сигнатурними методами. Сигнатурний підхід є ефективним лише для вже відомих зразків і не забезпечує своєчасного виявлення нових або модифікованих загроз.

Таким чином, виникає необхідність розроблення системи виявлення шкідливого програмного забезпечення в ОС мобільних систем, яка базується на комплексному аналізі структури прикладних програм, їхніх дозволів, компонентної архітектури та поведінкових характеристик у процесі виконання.

Метою роботи є розроблення системи виявлення шкідливих програм в ОС мобільних пристроїв, що забезпечує підвищення рівня захищеності шляхом виявлення потенційно небезпечних програм на основі аналізу їхніх структурних та поведінкових ознак.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– проаналізувати вбудовані і сторонні механізми захисту в ОС для мобільних пристроїв та визначити ключові точки контролю безпеки;

– дослідити структуру прикладних програм (.apk) та визначити інформативні параметри для виявлення шкідливої активності (маніфест, дозволи, компоненти, цифровий підпис);

– сформувати набір ознак для ідентифікації потенційно небезпечних програм;

– розробити архітектуру системи виявлення шкідливого ПЗ, що поєднує статичний аналіз (структура та дозволи) і динамічний аналіз (поведінка програми);

– реалізувати прототип системи та провести експериментальну оцінку її ефективності;

– сформулювати практичні рекомендації щодо впровадження розробленої системи в мобільних інформаційних середовищах.

РОЗДІЛ 2. АРХІТЕКТУРА ТА АЛГОРИТМ СИСТЕМИ ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ В ОПЕРАЦІЙНИХ СИСТЕМАХ МОБІЛЬНИХ ПРИСТРОЇВ

2.1 Аналіз механізмів захисту в операційних системах для мобільних пристроїв

В ОС для мобільних пристроїв з відкритим вихідним кодом необхідна надійна архітектура безпеки та відповідне програмне забезпечення, оскільки вона містить значний обсяг інструментів, сервісів і персональної інформації. В ОС Android реалізовано багаторівневу систему безпеки, яка забезпечує гнучкість, необхідну для відкритої платформи, та гарантує захист для всіх користувачів. У системі реалізовано контроль безпеки як з боку розробників, так і з боку користувачів [39].

Однією з важливих функцій безпеки, реалізованих в ОС для мобільних пристроїв, є «пісочниця» (sandbox), яка дозволяє суттєво обмежити функціональні можливості ядра ОС і тим самим певною мірою підвищити її загальну захищеність [39].

Модель безпеки ОС для мобільних пристроїв використовує механізми безпеки, реалізовані в ядрі ОС Linux. Вона є багатокористувацькою ОС, а її ядро може розділяти ресурси користувачів так само, як ізолює процеси. В ОС Linux один користувач не може отримати доступ до файлів іншого користувача без надання відповідного дозволу. Кожен процес виконується з певним ідентифікатором: використовується ідентифікатор користувача – UID і групи – GID. Процес не буде запущений, якщо не встановлено ідентифікатор користувача або ідентифікатор групи, тобто відповідні біти SUID і SGID для виконуваного файлу [39].

На рис. 2.1 представлено модель безпеки сучасної ОС для мобільних пристроїв (на прикладі ОС Android). Вона має добре організовану систему безпеки як на програмному, так і на апаратному рівнях. У цій системі реалізовано

та використовуються сучасні інформаційні технології та низка перспективних методів забезпечення захисту інформації на пристрої, описаних нижче.

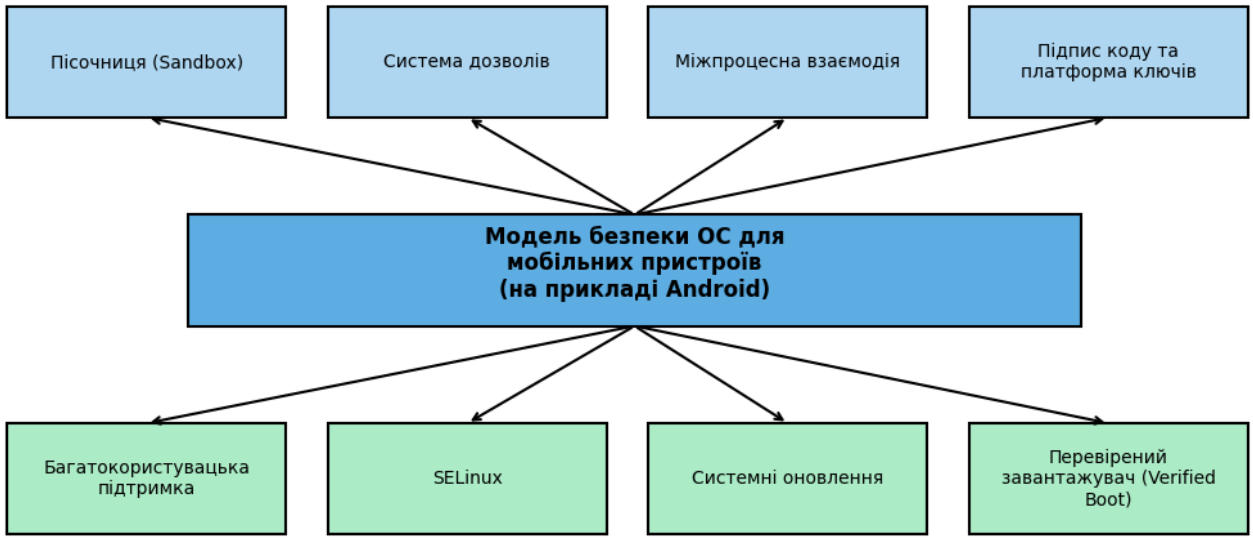


Рис.2.1 – Модель безпеки (на прикладі ОС Android)

Модель безпеки ОС для мобільних пристроїв включає наступні 8 модулів.

Пісочниця. Під час інсталяції програм в ОС кожній програмі за замовчуванням присвоюються унікальні ідентифікатори користувача (UID) та групи (GID), кожній програмі відповідає окремий користувач. Ім'я користувача має формат «app_x», а ідентифікатор користувача обчислюється за формулою (Process.FIRST_APPLICATION_UID + x). Process.FIRST_APPLICATION_UID дорівнює 10000. Ці ідентифікатори програми не змінюються. Список встановлених програм зберігається у файлі packages.list, розташованому в каталозі «/data/system/» [39].

Кожна програма має власний домашній каталог, наприклад «/data/data/package_name», де package_name – ім'я пакета, наприклад com.ex.ex1. Ім'я пакета задається у властивості пакета у файлі AndroidManifest.xml. Цей каталог є внутрішнім сховищем програми, де вона зберігає власні дані, і до якого розробники отримують доступ через функції Context.getFilesDir() або Context.getDir() [39].

Права доступу до каталогу визначені як drwxr-x--x, тобто лише власник і користувачі, що входять до групи власника, мають повний доступ до нього. Кожна програма визначається як унікальний користувач, що означає відсутність доступу за замовчуванням до інформації інших програм [39].

Крім того, на рівні ядра унікальні UID і GID кожної програми використовуються для розмежування доступу до ресурсів, таких як пам’ять і процесор. Таким чином, на рівні ядра для кожної програми створюється власна пісочниця – ізольоване середовище, яке обмежує дії програми та захищає систему. ОС автоматично присвоює унікальний UID кожній програмі під час інсталяції та запускає її як окремий процес із власним UID. Також кожна програма отримує окремий каталог, до якого лише вона має права читання та запису [39].

Отже, програми ізолюються як на рівні процесів (окремий процес для кожної програми), так і на рівні файлової системи (окремий каталог для кожної програми). Механізм пісочниці в мобільній ОС базується на присвоєнні кожній програмі унікального UID і GID та створенні для неї окремого непривілейованого користувача.

Система дозволів. Програми ізолюються та отримують доступ лише до власних файлів і загальнодоступних ресурсів ОС для мобільних пристроїв. Для розширення функціональності програма може отримати додаткові права – дозволи, які надають можливість використовувати додаткові функції та ресурси, необхідні для її роботи [39].

Дозволи можуть контролювати доступ до драйверів, інтернет-з’єднання, даних, сервісів та інших важливих функцій.

Програми запитують перелік дозволів через файл AndroidManifest.xml. Під час інсталяції ОС зчитує список дозволів, і користувач приймає рішення щодо надання прав програмі. Після надання прав вони більше не запитуються повторно та автоматично застосовуються до програми. Існує кілька рівнів доступу, які визначають, наскільки широкими є повноваження програми під час виконання.

Фреймворк міжпроцесорної взаємодії. Програми можуть виконувати запити до системних сервісів для отримання доступу до ресурсів ОС, наприклад

до навігаційних даних. Системні сервіси, у свою чергу, можуть надавати таку інформацію програмам.

Оскільки програми та системні сервіси виконуються в різних процесах, для організації взаємодії використовується механізм обміну даними між процесами. Він також необхідний для взаємодії програм між собою.

В ОС для мобільних пристроїв обмін сигналами та даними між процесами організований за допомогою фреймворку міжпроцесорної взаємодії, а в окремих випадках використовуються сокети домену Unix. Усі інші способи взаємодії реалізуються через цей фреймворк [40].

Фреймворк міжпроцесорної взаємодії надає можливість [40]:

– синхронно та асинхронно викликати методи віддалених об’єктів так, ніби вони локальні;

– обмінюватися файловими дескрипторами між процесами;

– отримувати автоматичне сповіщення у разі переривання взаємодії з певним процесом тощо.

Взаємодія між процесами організована за синхронною клієнт-серверною моделлю. Клієнт ініціює з’єднання і очікує відповіді від сервера, тобто обмін відбувається послідовно. Це дозволяє розробнику викликати віддалені методи так, ніби вони виконуються в межах одного процесу.

Схема взаємодії процесів через фреймворк міжпроцесорної взаємодії наведена на рис. 2.1: програма-клієнт, що виконується в процесі А, отримує доступ до функціональності сервісу, який працює в процесі В.

Уся взаємодія між клієнтом і сервером у межах фреймворку міжпроцесорної взаємодії здійснюється через спеціальний драйвер пристрою ОС Linux, який розташований у каталозі «/dev/binder», за допомогою класу під назвою Stub [40].

Підпис коду та платформа ключів. Усі сторонні та вбудовані системні програми в ОС для мобільних пристроїв повинні бути підписані їх розробником. Оскільки це APK-файли формату Java-пакета JAR, використовується механізм підпису коду на основі JAR-підпису [40].

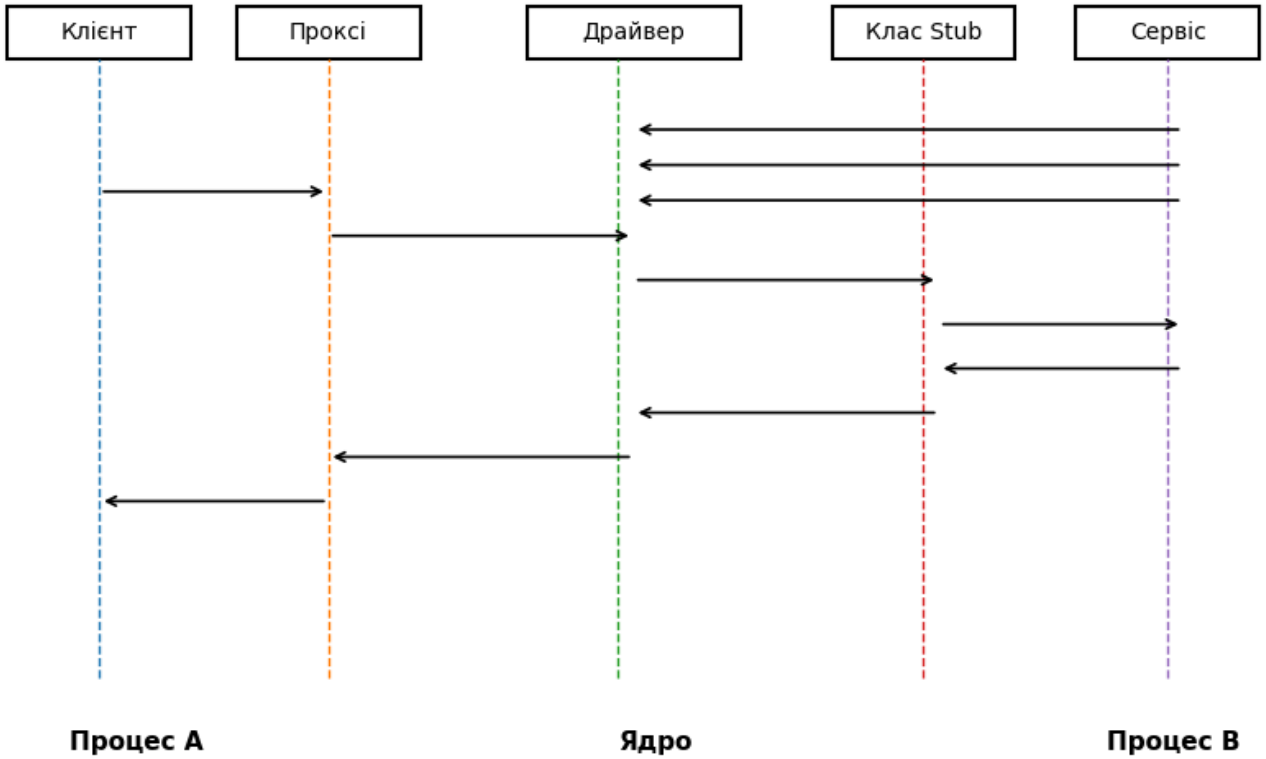


Рис. 2.2 – Схема функціонування фреймворку міжпроцесорної взаємодії

ОС застосовує підпис APK для перевірки того, що оновлення програми надходять від того самого автора, а також для встановлення довірчих відносин між програмами. Обидві ці функції безпеки реалізуються шляхом порівняння сертифікатів підпису.

Вбудовані системні програми підписуються ключем платформи. Різні компоненти системи можуть обмінюватися ресурсами та працювати в одному процесі, якщо вони підписані тим самим ключем платформи.

Багатокористувацький режим. Цей режим надає можливість використовувати пристрій кільком користувачам, при цьому кожен має власний виділений простір і ресурси, недоступні іншим [40].

Мандатне керування доступом (SELinux). Функція SELinux додає додатковий рівень безпеки в ОС для мобільних пристроїв, що називається «мандатне керування доступом». Цей рівень конфігурується за допомогою загальносистемних політик безпеки та забезпечує додаткові можливості керування доступом користувачів і програм як до власних даних, так і до

системних ресурсів, причому це відбувається у фоновому режимі для користувача [41].

З технічної точки зору, ця функція дозволяє створювати політики, що визначають дозволені та заборонені типи взаємодії для кожного процесу в межах загального контексту безпеки. SELinux може працювати у факультативному режимі, в якому політики не застосовуються примусово, а порушення лише фіксуються в журналі.

Системні оновлення. Є важливим механізмом забезпечення безпеки ОС для мобільних пристроїв. Вони містять виправлення виявлених уразливостей, покращення механізмів захисту та оновлення компонентів системи. Регулярне встановлення оновлень дозволяє зменшити ризик експлуатації відомих вразливостей шкідливим програмним забезпеченням та підвищує загальний рівень захищеності пристрою [41].

Перевірений завантажувач. Забезпечує контроль цілісності програмного забезпечення на етапі запуску пристрою. Під час завантаження система перевіряє цифрові підписи компонентів ОС і блокує запуск у разі виявлення змін або несанкціонованих модифікацій. Це дозволяє запобігти запуску скомпрометованої або зміненої версії операційної системи та захищає пристрій від втручання на низькому рівні [41].

Наразі існує багато антивірусних програм: AhnLab V3 Mobile; Anguanjia; Antiy AVL for Android; Avast Mobile Security; Avira Antivirus Security; Bitdefender Mobile Security; ESET Mobile Security; G Data Internet Security; Qihoo 360 Antivirus; Quick Heal Total Security; Trend Micro Mobile Security.

Антивірусні програми працюють на основі сигнатурного аналізу. Відповідно, якщо шкідливі програми відсутні в базі сигнатур або якщо база сигнатур не була оновлена з певних причин, такі загрози не будуть виявлені. Сам процес виявлення, аналізу та додавання сигнатури до антивірусної бази є трудомістким і потребує значного часу.

Загалом ОС Android має організовану модель безпеки, у якій реалізовано сучасні інформаційні технології захисту. Проте будь-яка інформаційна система, навіть найсучасніша, потребує постійного вдосконалення механізмів захисту,

оскільки з часом зловмисники знаходять нові вразливості та способи обходу існуючих засобів безпеки.

Зокрема, одним із механізмів захисту є система дозволів, які програма запитує під час інсталяції. Однак не кожен користувач аналізує перелік запитуваних функцій.

Отже, ОС залишається вразливою до нових шкідливих програм. Нові або модифіковані шкідливі програми, які відсутні в базі сигнатур антивірусного програмного забезпечення, можуть отримати несанкціонований доступ і негативно вплинути на ОС для мобільних пристроїв. Статистика свідчить про значне зростання кількості шкідливих програм, що є підставою для розроблення сучасного засобу виявлення нових шкідливих програм, здатного без наявності сигнатури в базі здійснювати їх виявлення шляхом аналізу поведінкових характеристик програм.

2.2 Дослідження множини шкідливих програм

Наразі шкідливі програми в ОС для мобільних пристроїв (на прикладі Android) перебувають у вільному доступі та можуть бути знайдені в мережі Інтернет. Аналіз шкідливих програм дозволив виявити властивості та особливості, притаманні їхній поведінці, зрозуміти принципи їх функціонування та сформувати експериментальну вибірку для розроблення майбутньої системи виявлення шкідливого програмного забезпечення.

У таблиці 2.1 наведено перелік досліджуваної множини шкідливих програм.

Аналіз шкідливих програм проводився у три етапи:

– аналіз файлу маніфесту та ресурсів програми, оскільки огляд структури в першому розділі показав важливість цього файла в ОС для мобільних пристроїв;

– декомпіляція програми з метою приведення коду до «читабельного» вигляду, оскільки програмний код перебуває у скомпільованому стані;

– аналіз коду та файлу маніфесту програм на наявність «підозрілих» рядків, що не відповідають заявленому функціоналу програми.

Таблиця 2.1 – Шкідливі програми, які досліджувалися

Назва	Опис
Android/SpySMS Trojan Spy.AndroidOS.Zbot.a Android.Smssniffer AndroidSpySMS AndroidOS_SMSREP.B	Перехоплює SMS-повідомлення та надсилає на сервер запит, у якому передає номер «зараженого» мобільного пристрою, вміст повідомлень і ідентифікатор «зараженого» пристрою. Така поведінка характерна для програм, націлених на перехоплення SMS-кодів для двофакторної банківської автентифікації.
DroidKungFu	Надає можливість отримати доступ до мобільного пристрою з правами привілейованого користувача для збору інформації про пристрій і користувача.
FlashPlayer	Надає можливість отримати доступ до мобільного пристрою з правами привілейованого користувача для збору інформації про пристрій і користувача.
SimpleLocker	Взаємодіє із сервером і шифрує файли на пристрої.
Vanilla Music Player	Надсилає SMS-повідомлення на короткі номери.

Будь-яка програма є файлом формату apk – архівним файлом типу zip. У файлі apk містяться стандартні файли та каталоги: meta-inf, res, assets, androidmanifest.xml, resources.arsc, classes.dex.

Каталог meta-inf містить інформацію про файл apk: контрольні суми, сертифікати.

Каталоги res, assets, resources.arsc містять мультимедійні файли, необхідні для функціонування програми.

Файл androidmanifest.xml містить інформацію про програму, а саме: апаратні та програмні вимоги для її роботи, перелік дозволів, що запитуються програмою, назви задіяних класів і бібліотек.

Файл classes.dex містить програмний код.

Цей файл було декомпільовано та відкрито за допомогою програми apktools, що дозволяє привести код аналізованої програми до читабельного вигляду.

Ключову роль відіграє файл androidmanifest.xml, який містить перелік прав на запуск відповідних сервісів і керування ними. Система дозволів в ОС для мобільних пристроїв (на прикладі Android) включає 152 дозволи. Кожен дозвіл відповідає за доступ до певного апаратного або програмного ресурсу. Проаналізувавши їх перелік, можна зробити висновок щодо критичності запуску певного сервісу. У [42] описано всі 152 дозволи.

Шляхом аналізу критичності та значущості кожному дозволу було присвоєно значення:

0 – прийнятний (безпечний);

1 – небезпечний, тобто при наданні доступу можуть бути задіяні програмно-апаратні ресурси, керування якими шкідливою програмою може завдати шкоди.

Опис свідчить, що певна кількість дозволів є критичною, тобто після надання програмі відповідних прав мобільний пристрій виконуватиме функції, притаманні шкідливим програмам.

У результаті детального аналізу вихідного коду досліджуваних шкідливих програм, а також додаткової інформації з мережі Інтернет, було виявлено ознаки, що характеризують їх поведінку:

– наявність файлу формату .txt, що містить списки номерів (наприклад, smsc.txt (7921, 7923, 7924 тощо)), який зберігається у прихованому вигляді та містить координати, адреси й номери для надсилання платних SMS або інших запитів;

– наявність обфускації коду – шифрування або заплутування коду з метою ускладнення його аналізу;

– наявність класу, що відповідає за шифрування (file encryptor (context)), який використовує методи шифрування даних;

– наявність класу telephony, що відповідає за роботу з телефонними функціями;

– наявність класу smsreceiver, що відповідає за обробку SMS-повідомлень;

– наявність класу smsblockerthead;

– наявність підкласу smsblockerthead;

– служба org.torproject.android.service.tor_service, яка дозволяє встановлювати анонімне мережеве з’єднання, захищене від прослуховування;

– наявність файлу в базі сигнатур мобільної антивірусної програми. Класичний метод сигнатурного аналізу працює ефективно, а в поєднанні з динамічним аналізом ефективність і якість виявлення зростає;

– система дозволів на використання сервісів ОС для мобільних пристроїв (на прикладі Android). За правильного підходу та аналізу системи дозволів можна запобігти інсталяції шкідливої програми на початковому етапі.

Узагальнивши та проаналізувавши всю зібрану інформацію, можна сформувати модель поведінки шкідливих програм на основі виявлених ознак.

У результаті виконаної роботи було сформовано експериментальну вибірку для аналізу поведінки з метою виявлення шкідливих програм в ОС для мобільних пристроїв.

2.3 Архітектура системи виявлення шкідливих програм в операційних системах мобільних пристроїв

Описані в першому розділі проблеми використання наявних методів виявлення шкідливих програм в мобільних операційних системах дають підстави для розроблення іншого підходу до побудови системи захисту. Аналіз існуючих рішень показав, що окреме застосування сигнатурних або поведінкових методів

не забезпечує достатнього рівня ефективності в умовах постійної модифікації шкідливого коду, використання обфускації та прихованих механізмів активації.

У зв'язку з цим доцільним є формування архітектури системи виявлення, яка базується на поєднанні декількох взаємопов'язаних модулів аналізу. Такий підхід забезпечує динамічне розв'язання задачі виявлення шкідливих програм, підвищує адаптивність системи та дозволяє реалізувати гнучкий і масштабований механізм захисту мобільних пристроїв.

Ідея, покладена в основу запропонованої архітектури, полягає у послідовному виявленні та аналізі потенційно небезпечного застосунку на різних етапах його життєвого циклу. Насамперед здійснюється фіксація факту інсталяції або запуску мобільного застосунку, після чого виконується первинна перевірка його структури та параметрів безпеки. На цьому етапі аналізуються запитувані дозволи, компоненти застосунку, цифровий підпис, а також наявність характерних ознак шкідливого коду.

У разі виявлення підозрілих характеристик система ініціює поглиблений аналіз поведінки застосунку під час виконання в контрольованому середовищі. Такий підхід дозволяє зафіксувати спроби несанкціонованого доступу до ресурсів пристрою, ініціювання мережевих з'єднань, прихованого надсилання повідомлень або інших дій, що можуть свідчити про наявність шкідливого функціоналу. Після підтвердження підозрілої активності застосунок може бути заблокований або обмежений у правах доступу, а його поведінкові ознаки зберігаються для подальшого аналізу.

Подальшим етапом є використання отриманих ознак для виявлення аналогічних загроз серед інших застосунків. Це дозволяє формувати узагальнений профіль шкідливої поведінки та підвищувати ефективність системи за рахунок накопичення знань про характерні сценарії функціонування шкідливих програм.

За основу побудови системи було взято типову архітектуру мобільної операційної системи, що базується на ізоляції застосунків, моделі дозволів та контрольованому процесі інсталяції. Запропонований метод виявлення шкідливих програм інтегрується в механізми безпеки мобільної ОС та забезпечує можливість

виявлення загроз як на етапі встановлення застосунку, так і під час його виконання.

Для відображення структури та функцій системи виявлення була розроблена функційна модель. Функційна модель являє собою структуроване подання функціональних компонентів системи, інформаційних потоків між ними та об’єктів, що беруть участь у процесі аналізу. Побудова моделі здійснюється методом декомпозиції – від загального процесу забезпечення безпеки мобільної операційної системи до окремих модулів аналізу, оцінювання та прийняття рішення.

Контекстний рівень функційної моделі (рис. 2.3) відображає, що на вхід системи виявлення надходить інсталяційний пакет застосунку та пов’язані з ним дані виконання. На виході система формує оцінку рівня ризику, класифікацію застосунку (безпечний, підозрілий або шкідливий), а також інформацію щодо виявлених ознак небезпечної поведінки.

Для визначення переліку функцій, що повинні бути реалізовані в системі захисту мобільної операційної системи, а також для встановлення взаємозв’язків між ними було виконано декомпозицію процесу виявлення шкідливих програм на окремі функціональні підсистеми.

Рис. 2.3 – Контекстний блок функційної моделі системи виявлення шкідливих програм у мобільній операційній системі

У результаті проведеної декомпозиції було сформовано набір процесів, що відображають функціональні особливості системи виявлення шкідливих програм у мобільній операційній системі. Виділені процеси спрямовані на розв’язання типових задач забезпечення інформаційної безпеки мобільних пристроїв та

охоплюють повний цикл аналізу застосунку – від його перевірки до прийняття рішення щодо подальшого функціонування.

До основних процесів системи належать:

– процес первинного аналізу застосунку на етапі інсталяції;

– процес дослідження структури застосунку та запитуваних дозволів;

– процес виявлення характерних ознак потенційно шкідливої поведінки;

– процес поведінкового моніторингу застосунку під час виконання;

– процес формування узагальненого набору ознак та оцінювання рівня ризику;

– процес прийняття рішення щодо дозволу, обмеження або блокування застосунку;

– процес накопичення та оновлення інформації про виявлені загрози.

Для забезпечення ефективного функціонування системи зазначені процеси повинні реалізовуватися на різних етапах життєвого циклу застосунку та на різних рівнях мобільної операційної системи.

Зокрема, процеси первинного аналізу та дослідження структури застосунку доцільно реалізовувати на етапі інсталяції, що дозволяє запобігти встановленню потенційно небезпечного програмного забезпечення ще до його запуску. На цьому рівні аналізуються інсталяційний пакет, цифровий підпис, перелік дозволів і компоненти застосунку.

Процес поведінкового моніторингу має реалізовуватися під час виконання застосунку. Він забезпечує фіксацію фактичних дій програми, зокрема спроб доступу до конфіденційних ресурсів, ініціювання мережевих з’єднань або створення фонових служб. Такий підхід дозволяє виявляти шкідливу активність, яка не може бути однозначно визначена шляхом статичного аналізу.

Процес формування ознак та оцінювання ризику об’єднує результати попередніх етапів та забезпечує визначення рівня потенційної небезпеки застосунку. На основі отриманої оцінки реалізується процес прийняття рішення – дозвіл на встановлення або виконання, обмеження функціональності чи повне блокування застосунку.

Процес накопичення інформації про виявлені ознаки шкідливої активності забезпечує підвищення ефективності системи в майбутньому, оскільки дозволяє формувати узагальнені профілі небезпечної поведінки та використовувати їх для повторного аналізу інших застосунків.

На виході зазначених процесів формуються різні типи даних, зокрема оцінка рівня ризику, класифікація застосунку, рішення щодо його подальшого використання та структурований звіт про виявлені ознаки потенційної загрози. Узагальнений перелік вихідних даних наведено в таблиці 2.2.

Таблиця 2.2 – Результати роботи системи

Процес	**Дані на виході**
Аналіз застосунку на етапі інсталяції	Результат перевірки APK; перелік виявлених підозрілих дозволів; інформація про структуру застосунку
Виявлення ознак потенційно шкідливої поведінки	Список виявлених індикаторів (підозрілі виклики API, приховані служби, спроби доступу до ресурсів)
Поведінковий моніторинг застосунку	Журнал зафіксованих дій під час виконання; інформація про мережеву активність та доступ до ресурсів
Формування ознак та оцінювання ризику	Узагальнений профіль застосунку; розрахований рівень ризику
Прийняття рішення та реагування	Рішення (дозвіл / обмеження / блокування); статус виконання застосунку
Накопичення інформації про загрози	Оновлена база ознак шкідливої активності; збережені профілі проаналізованих застосунків

На основі аналізу функційної моделі було зроблено висновок, що ефективна система виявлення шкідливих програм у мобільній операційній системі повинна мати модульну та ієрархічну архітектуру. Такий підхід дозволяє розподілити

функціональне навантаження між окремими компонентами системи, забезпечити їх спеціалізацію та спростити подальшу модернізацію або масштабування рішення.

Модульність архітектури зумовлена необхідністю поєднання різних методів аналізу – статичних і динамічних – а також забезпечення гнучкого механізму прийняття рішення. У межах мобільної операційної системи всі функціональні компоненти повинні працювати узгоджено та інтегруватися з механізмами керування застосунками, моделлю дозволів і системними службами безпеки.

Запропонована архітектура системи складається з кількох логічно пов’язаних модулів, кожен із яких виконує окрему групу задач у межах загального процесу виявлення шкідливого програмного забезпечення.

Першим етапом є модуль аналізу застосунку на етапі інсталяції. Його основне призначення – здійснення первинної перевірки інсталяційного пакета застосунку. На цьому рівні виконується контроль цілісності файлів, перевірка цифрового підпису, аналіз метаданих та відповідності застосунку встановленим політикам безпеки. Реалізація цього модуля на етапі інсталяції дозволяє запобігти встановленню потенційно небезпечного програмного забезпечення ще до початку його виконання.

Наступним компонентом є модуль статичного аналізу. Він здійснює дослідження структури застосунку без його запуску. У процесі аналізу вивчаються запитувані дозволи, компоненти застосунку (служби, приймачі повідомлень, провайдери даних), наявність потенційно небезпечних викликів API, а також ознаки обфускації або прихованого функціоналу. Результатом роботи цього модуля є формування набору статичних характеристик, які можуть свідчити про підвищений рівень ризику.

Оскільки частина шкідливих механізмів активується лише під час виконання, статичний аналіз доповнюється модулем поведінкового моніторингу. Цей модуль функціонує під час роботи застосунку та фіксує фактичні дії програми. До таких дій належать спроби доступу до конфіденційних ресурсів пристрою, ініціювання мережевих з’єднань, створення фонових процесів, взаємодія з іншими застосунками, зміна системних налаштувань або маніпуляції з

файлами користувача. Поведінковий аналіз дозволяє виявити ті загрози, які не можуть бути однозначно ідентифіковані лише за структурними характеристиками.

Отримані в процесі статичного та динамічного аналізу дані передаються до модуля формування ознак і оцінювання ризику. У цьому модулі відбувається узагальнення зібраної інформації та перетворення її у формалізований опис поведінки застосунку. На основі визначених критеріїв або алгоритмів оцінювання розраховується рівень ризику. Такий підхід дозволяє перейти від набору розрізнених характеристик до кількісної або категоріальної оцінки безпеки застосунку.

Модуль прийняття рішення є завершальним етапом обробки інформації. Він використовує отриману оцінку ризику для визначення подальших дій щодо застосунку. Залежно від результатів аналізу система може дозволити встановлення або виконання програми, обмежити її функціональність шляхом зміни дозволів або повністю заблокувати її роботу. У разі виявлення небезпечної активності користувач або адміністратор отримує відповідне повідомлення з поясненням причин прийнятого рішення.

Окремим важливим компонентом архітектури є підсистема накопичення та оновлення інформації про загрози. Вона забезпечує збереження результатів аналізу, виявлених ознак шкідливої поведінки та історії прийнятих рішень. Накопичення таких даних дозволяє підвищувати ефективність системи в майбутньому, формувати профілі небезпечної активності та використовувати їх для повторного аналізу нових застосунків.

Концептуальний алгоритм функціонування системи можна подати як послідовність взаємопов’язаних етапів. Спочатку система фіксує надходження інсталяційного пакета або запуск застосунку. Далі виконується статичний аналіз структури та дозволів. У разі виявлення підозрілих характеристик або за визначеною політикою безпеки ініціюється поведінковий моніторинг. Після збору необхідних даних формується узагальнений профіль застосунку та обчислюється рівень ризику. Завершальним кроком є прийняття рішення та реалізація відповідної політики реагування.

Взаємодія між модулями здійснюється через внутрішні механізми мобільної операційної системи. Кожен компонент має чітко визначені вхідні та вихідні параметри, що забезпечує узгодженість функціонування всієї системи. Така організація дозволяє забезпечити структурованість обробки інформації, зменшити складність реалізації та підвищити надійність роботи системи виявлення.

Таким чином, запропонована архітектура забезпечує комплексний підхід до виявлення шкідливих програм, поєднуючи аналіз структури застосунку, моніторинг його поведінки та механізм оцінювання ризику. Чітке розмежування функцій між модулями та послідовність обробки даних створюють основу для побудови ефективної системи захисту мобільних пристроїв від сучасних загроз.

Загальна архітектура системи виявлення шкідливих програм в ОС складається з таких елементів, які наведено на рис. 2.4.

Рис. 2.4 – Архітектура системи виявлення шкідливих програм

2.4 Алгоритм функціонування системи виявлення шкідливих програм

Функціонування системи виявлення шкідливих програм у мобільній операційній системі ґрунтується на послідовній обробці застосунку з

використанням статичних та динамічних методів аналізу. Алгоритм роботи модулів реалізує поетапний механізм оцінювання ризику та прийняття рішення щодо подальшого функціонування програмного забезпечення.

Процес починається з надходження інсталяційного пакета застосунку (APK) у систему. Це може відбуватися під час встановлення нового застосунку або оновлення вже встановленого. Подія інсталяції ініціює запуск механізмів перевірки.

На першому етапі виконується інсталяційна перевірка. На цьому рівні здійснюється аналіз цілісності пакета, перевірка цифрового підпису, дослідження базових метаданих застосунку та відповідність політикам безпеки операційної системи. Результати перевірки передаються до модуля керування та прийняття рішень.

Другим етапом є статичний аналіз структури та дозволів. У межах цього етапу досліджуються компоненти застосунку, перелік запитуваних дозволів, наявність потенційно небезпечних викликів API, а також інші структурні ознаки. Метою є виявлення ризикових характеристик без фактичного запуску застосунку.

Після завершення статичного аналізу модуль керування визначає доцільність запуску динамічного етапу. Якщо виявлено підозрілі ознаки або рівень ризику перевищує встановлений поріг, ініціюється поведінковий моніторинг.

Поведінковий моніторинг здійснює контроль виконання застосунку в процесі його роботи. На цьому етапі фіксуються спроби доступу до конфіденційних ресурсів пристрою, мережеві з’єднання, запуск фонових служб, взаємодія з іншими компонентами системи. Зібрані дані передаються до модуля формування ознак та оцінювання ризику.

Модуль формування ознак агрегує результати статичного та динамічного аналізу, формує узагальнений профіль поведінки застосунку та розраховує рівень ризику.

Формування набору ознак:

Нехай: P_i – індикатори дозволів; S_j – структурні індикатори; D_k – поведінкові індикатори. Кожна ознака приймає значення: 0 – якщо вона відсутня та 1 – якщо вона виявлена.

Кожному дозволу надається коефіцієнт небезпечності w_i. Наприклад: READ_SMS – 5; SEND_SMS – 6; READ_CONTACTS – 4; INTERNET – 2.

Тоді ***ризик дозволів*** розраховується за формулою:

$$R_P = \sum_{i=1}^{n} w_i P_i \quad (1)$$

Ризик за структурними ознаками розраховується за формулою:

$$R_S = \sum_{j=1}^{n} v_j S_j \quad (2)$$

де v_j – вагові коефіцієнти, які визначаються експертним методом на основі моделі загроз мобільної ОС. Тобто:

- аналізуються типові сценарії шкідливої активності;
- визначається вплив кожної ознаки на безпеку;
- присвоюється коефіцієнт небезпечності в межах шкали.

Для кожної ознаки визначаються:

- можливий вплив на конфіденційність (C),
- можливий вплив на цілісність (I),
- можливий вплив на доступність (A).

Оцінка проводиться за шкалою 0-3: 0 – відсутній вплив; 1 – низький; 2 – середній; 3 – високий. Вага визначається як:

$$v_j = C_j + I_j + A_j \quad (3)$$

Максимальне значення = 9.

Ризик за поведінкою розраховується за формулою***:***

$$R_D = \sum_{k=1}^{l} u_k D_k \quad (4)$$

Вага u_k визначається на основі двох параметрів: потенційна шкода I_k та ймовірність зловживання E_k, де 1 – незначний вплив/рідко використовується у шкідливому ПЗ; 2 – локальний вплив/зустрічається іноді; 3 – витік некритичних даних/типова поведінка троянів; 4 – витік персональних даних/часто використовується; 5 – фінансові або системні наслідки/критично характерна для шкідливого ПЗ.

Формула ваги:

$$u_k = \frac{I_k * E_k}{5} \quad (5)$$

Окрім сумарного бального оцінювання окремих дозволів та поведінкових індикаторів, у системі враховуються конкретні поєднання ознак, які характерні для реальних сценаріїв шкідливої активності в мобільних ОС. Такі комбінації формують ***додатковий ризик*** R_C.

Приклади критичних комбінацій:

– SMS-шахрайство (преміум-сервіси). Ознаки: дозвіл SEND_SMS; дозвіл READ_SMS; відсутність явного інтерфейсу для роботи з SMS; фоновий сервіс. Додатковий ризик C_1=15;

– викрадення контактів. Ознаки: READ_CONTACTS; активні мережеві; з'єднання; передача даних після запуску. Додатковий ризик C_2=12;

– прихована передача даних. Ознаки: READ_EXTERNAL_STORAGE; INTERNET; частота мережевих запитів > встановленого порогу; відсутність активної взаємодії з користувачем. Додатковий ризик C_3=10;

– автозапуск + фонове виконання. Ознаки: RECEIVE_BOOT_COMPLETED; наявність background service; відсутність видимого UI. Додатковий ризик C_4=10;

– підозріле оновлення. Ознаки: зміна цифрового підпису при оновленні; додавання нових небезпечних дозволів. Додатковий ризик C_5=20;

– доступ до телефонії + мережа. Ознаки: READ_PHONE_STATE; INTERNET; часті зовнішні з'єднання. Додатковий ризик C_5=7.

Розрахунок сумарного штрафу:

$$R_C = \sum_{a=1}^{6} C_a \quad (6)$$

Підсумковий ризик:

$$R_{total} = \frac{R_P + R_S + R_D + R_C}{R_{\max}} * 100 \quad (7)$$

R_{max} – це максимально можливий бал ризику, який може отримати застосунок за умови, що:

- виявлено всі небезпечні дозволи,
- наявні всі структурні ризикові ознаки,
- поведінка перевищує всі пороги,
- спрацювали всі критичні комбінації.

Тобто це просто сума максимально можливих значень кожного блоку.

Інтерпретація значень R_{total} є наступною:

0-30 – безпечна програма; 31-60 – підозріла програма, обмеження функціоналу; 61-100 – небезпечна програма, повне блокування.

Отримані результати передаються до модуля керування та прийняття рішень, який визначає остаточну дію: дозволити виконання застосунку, обмежити його функціональність або заблокувати.

Усі зібрані дані, сформовані ознаки та прийняті рішення зберігаються у базі даних системи. Це забезпечує накопичення інформації про загрози та підвищення ефективності подальшого аналізу.

Таким чином, алгоритм роботи системи забезпечує послідовний, багаторівневий аналіз застосунку та дозволяє реалізувати комплексний механізм виявлення шкідливих програм у мобільному середовищі.

2.5 Висновки до розділу

У другому розділі було здійснено комплексне дослідження механізмів захисту мобільних операційних систем, проаналізовано особливості

функціонування шкідливих програм та розроблено архітектуру і алгоритм роботи системи їх виявлення.

Проведено аналіз моделі безпеки мобільної операційної системи (на прикладі Android). Встановлено, що багаторівнева архітектура захисту, яка включає механізм пісочниці, систему дозволів, підпис коду, SELinux, перевірений завантажувач та системні оновлення, забезпечує базовий рівень ізоляції застосунків і контролю доступу до ресурсів. Водночас визначено, що навіть за наявності сучасних механізмів безпеки мобільна ОС залишається вразливою до нових та модифікованих шкідливих програм, особливо у випадку використання обфускації коду, прихованих механізмів активації та соціальної інженерії. Окремо встановлено обмеженість суто сигнатурного підходу до виявлення загроз.

Досліджено множину реальних шкідливих програм для мобільної ОС. Проведений статичний та частково динамічний аналіз дозволив виявити характерні ознаки їхньої поведінки: перехоплення SMS-повідомлень, використання телефонних API, шифрування файлів, встановлення мережевих з’єднань, наявність прихованих служб, обфускацію коду та використання критичних дозволів. На основі узагальнення отриманих результатів сформовано експериментальну вибірку та визначено набір індикаторів, які можуть бути використані як формалізовані ознаки для побудови системи виявлення шкідливих програм.

Розроблено архітектуру системи виявлення шкідливих програм у мобільній операційній системі. Запропоновано модульну та ієрархічну структуру, що поєднує:

- модуль інсталяційної перевірки;
- модуль статичного аналізу;
- модуль поведінкового моніторингу;
- модуль формування ознак та оцінювання ризику;
- модуль прийняття рішення;
- підсистему накопичення та оновлення інформації про загрози.

Обґрунтовано доцільність комплексного підходу, який інтегрує статичний і динамічний аналіз, що дозволяє підвищити адаптивність системи до нових типів шкідливого програмного забезпечення.

Розроблено алгоритм функціонування системи, який реалізує послідовний механізм аналізу застосунку на різних етапах його життєвого циклу – від інсталяції до виконання. Запропоновано формалізовану модель оцінювання ризику на основі:

- вагових коефіцієнтів дозволів;
- структурних індикаторів;
- поведінкових індикаторів;
- критичних комбінацій ознак.

Визначено спосіб розрахунку підсумкового ризику та критерії прийняття рішення щодо дозволу, обмеження або блокування застосунку. Такий підхід забезпечує кількісну оцінку рівня небезпеки та створює основу для автоматизації процесу виявлення загроз.

Отже, у другому розділі було не лише теоретично обґрунтовано та структурно розроблено модель системи виявлення шкідливих програм у мобільній операційній системі, а й сформовано практичну основу для створення комплексного механізму захисту мобільних пристроїв. Отримані результати дозволяють перейти від ізольованого застосування окремих методів аналізу до побудови цілісної системи, здатної ефективно функціонувати в умовах постійної появи нових загроз.

Запропонований підхід забезпечує можливість раннього виявлення потенційно небезпечних застосунків на етапі інсталяції, що зменшує ризик компрометації пристрою. Доповнення статичного аналізу поведінковим моніторингом під час виконання програми дозволяє виявляти приховані або відкладені механізми шкідливої активності, які не можуть бути визначені лише за структурними ознаками. Таким чином, система орієнтована як на протидію

відомим загрозам, так і на виявлення нових модифікацій шкідливого програмного забезпечення.

Формалізація процесу оцінювання ризику та використання кількісної моделі розрахунку рівня небезпеки забезпечують об’єктивність і прозорість прийняття рішень щодо подальшого функціонування застосунку. Чітко визначені критерії та вагові коефіцієнти створюють передумови для автоматизації аналізу та подальшого вдосконалення алгоритмів виявлення.

Модульна архітектура системи забезпечує гнучкість, можливість масштабування та інтеграції з існуючими механізмами безпеки мобільної операційної системи. Накопичення інформації про виявлені ознаки шкідливої поведінки дозволяє підвищувати точність класифікації та зменшувати кількість помилкових спрацювань.

Отже, результати другого розділу мають вагоме теоретичне й практичне значення, формуючи методичну та алгоритмічну основу для реалізації програмного прототипу системи, експериментальної перевірки її ефективності та подальшого впровадження з метою підвищення рівня безпеки мобільних операційних систем.

РОЗДІЛ 3. РОЗРОБКА ТА ТЕСТУВАННЯ ПРОТОТИПУ СИСТЕМИ ВИЯВЛЕННЯ ШКІДЛИВИХ ПРОГРАМ В ОПЕРАЦІЙНИХ СИСТЕМАХ МОБИЛЬНИХ ПРИСТРОЇВ

3.1 Формування вимог до системи

Розроблена в роботі система виявлення шкідливих програм в операційних системах мобільних пристроїв є комплексним програмним рішенням, призначеним для забезпечення додаткового рівня захисту мобільної ОС шляхом аналізу застосунків на різних етапах їх життєвого циклу. Необхідність створення такої системи обґрунтована результатами аналізу архітектури мобільної операційної системи, механізмів її безпеки та дослідження множини шкідливих програм, проведеного у попередніх розділах.

Як показано раніше, мобільна ОС (на прикладі Android) має багаторівневу модель безпеки, що включає механізм sandbox, систему дозволів, міжпроцесну взаємодію, цифровий підпис застосунків та механізми контролю цілісності. Проте зростання кількості шкідливих програм, використання обфускації коду, прихованих механізмів активації та експлуатації наданих дозволів свідчить про необхідність додаткових засобів виявлення, здатних аналізувати не лише сигнатури, а й поведінкові характеристики програм.

Запропонована система за своєю суттю є системою виявлення та реагування на інциденти безпеки на рівні мобільного пристрою. Вона поєднує функції:

- аналізу інсталяційного пакета застосунку (APK);
- дослідження структури та компонентів програми;
- аналізу запитуваних дозволів;
- виявлення потенційно небезпечних викликів API;
- поведінкового моніторингу під час виконання;
- формування профілю застосунку та оцінювання рівня ризику;
- прийняття рішення щодо подальшого функціонування застосунку.

Оскільки система реалізує функції виявлення загроз, до неї можуть бути висунуті вимоги, аналогічні вимогам до засобів виявлення та реагування на комп'ютерні інциденти. Під час формування вимог було враховано положення міжнародних і національних нормативних документів у сфері інформаційної безпеки, зокрема:

– ISO/IEC 15408 (Common Criteria), який визначає підходи до формування функціональних вимог безпеки та вимог довіри до реалізації засобів захисту;

– ДСТУ ISO/IEC 27001:2023, що регламентує процеси управління інформаційною безпекою, включаючи виявлення та реагування на інциденти;

– методичні рекомендації ДССЗЗІ України щодо виявлення та реагування на комп'ютерні інциденти.

З урахуванням функціонального призначення розроблена система належить до активних засобів виявлення, оскільки за результатами аналізу вона не лише формує інформацію про можливу загрозу, а й може ініціювати обмеження або блокування виконання застосунку.

У наукових джерелах [43,44] системи виявлення загроз класифікують за методами аналізу. Відповідно до цієї класифікації, у запропонованій системі поєднуються:

– статичні методи (аналіз структури APK, файла AndroidManifest.xml, дозволів, компонентів, ознак обфускації);

– поведінкові методи (фіксація фактичних дій застосунку під час виконання);

– елементи профільного підходу (формування узагальненого набору ознак та оцінювання ризику).

Тип системи та її функціональне призначення визначають сукупність вимог до її функцій безпеки. З огляду на це, вимоги до системи доцільно згрупувати у чотири взаємопов'язані категорії.

Вимоги до функцій безпеки та середовища функціонування. Система повинна коректно інтегруватися з механізмами мобільної ОС, враховувати модель sandbox, систему дозволів і механізми міжпроцесної взаємодії. Вона не повинна порушувати ізоляцію застосунків та стабільність роботи ОС.

Вимоги до функціональних можливостей. Система повинна забезпечувати:

- збір даних про структуру та поведінку застосунку;
- аналіз отриманих даних у режимі, наближеному до реального часу;
- виявлення потенційно шкідливої активності на основі сукупності ознак;
- фіксацію результатів аналізу та формування журналів подій;
- оцінювання рівня ризику.

Вимоги до реалізації функціональних можливостей. Має бути забезпечено:

- протоколювання подій безпеки;
- можливість тестування окремих модулів;
- керування режимами роботи системи;
- контроль цілісності її компонентів;
- механізм оновлення бази ознак шкідливої поведінки.

Вимоги довіри до безпеки системи. До них належать вимоги щодо керування конфігурацією, документування, тестування, оцінювання вразливостей, захисту від несанкціонованого вимкнення або модифікації компонентів системи.

Функціональна структура системи передбачає наявність модулів збору даних (аналог сенсорів) та модулів аналізу (аналог аналізаторів). Модулі збору здійснюють отримання інформації про інсталяційний пакет або про дії застосунку під час виконання. Модулі аналізу виконують обробку отриманих даних, формують профіль поведінки та приймають рішення щодо рівня ризику. Рішення про визнання застосунку безпечним, підозрілим або шкідливим приймається на основі узагальнення результатів статичного та поведінкового аналізу.

Таким чином, формування вимог до системи базується на поєднанні:

- особливостей архітектури мобільної ОС;
- результатів аналізу шкідливих програм;
- сучасних підходів до побудови систем виявлення загроз;
- положень міжнародних і національних стандартів у сфері інформаційної безпеки.

Сформульовані вимоги визначають функціональну та архітектурну основу подальшого проєктування системи виявлення шкідливих програм у мобільній операційній системі.

3.2 Проєктування та реалізація прототипу системи

Відповідно до розробленої архітектури системи виявлення шкідливих програм в операційних системах мобільних пристроїв, прототип будується як сукупність взаємопов’язаних функціональних модулів, що забезпечують поетапний аналіз застосунку та прийняття рішення щодо його подальшого функціонування. Така структура узгоджується з функційною моделлю, розглянутою раніше, де виділено процеси первинного аналізу, дослідження структури, поведінкового моніторингу, формування ознак, оцінювання ризику та реагування.

Основним принципом побудови прототипу є модульність. Кожен модуль реалізує окремий логічно завершений етап аналізу, а взаємодія між модулями здійснюється через внутрішній механізм передавання даних. Це дозволяє забезпечити масштабованість системи та можливість подальшого вдосконалення окремих компонентів без зміни загальної архітектури (рис. 3.1). Реалізація прототипу наведена в таблиці 3.1.

Першим етапом функціонування системи є первинний аналіз інсталяційного пакета застосунку. Модуль первинного аналізу активується під час надходження APK-файла та виконує перевірку його цілісності, аналіз цифрового підпису, витягування файла AndroidManifest.xml і формування переліку запитуваних дозволів. На цьому етапі здійснюється попередня оцінка критичності дозволів і виявлення очевидних ознак небезпечної конфігурації. Таким чином реалізується можливість реагування ще до запуску застосунку, що дозволяє запобігти встановленню потенційно небезпечного програмного забезпечення.

Наступним компонентом є модуль статичного аналізу структури застосунку. Його завданням є дослідження внутрішньої організації програми без її

виконання. У межах цього модуля аналізуються компоненти застосунку, виклики системних API, використання служб, приймачів повідомлень та інших елементів, що можуть свідчити про наявність прихованого функціоналу. Також виявляються ознаки обфускації або нестандартної структури коду. Результатом роботи модуля є формування набору статичних ознак, які передаються до підсистеми оцінювання ризику.

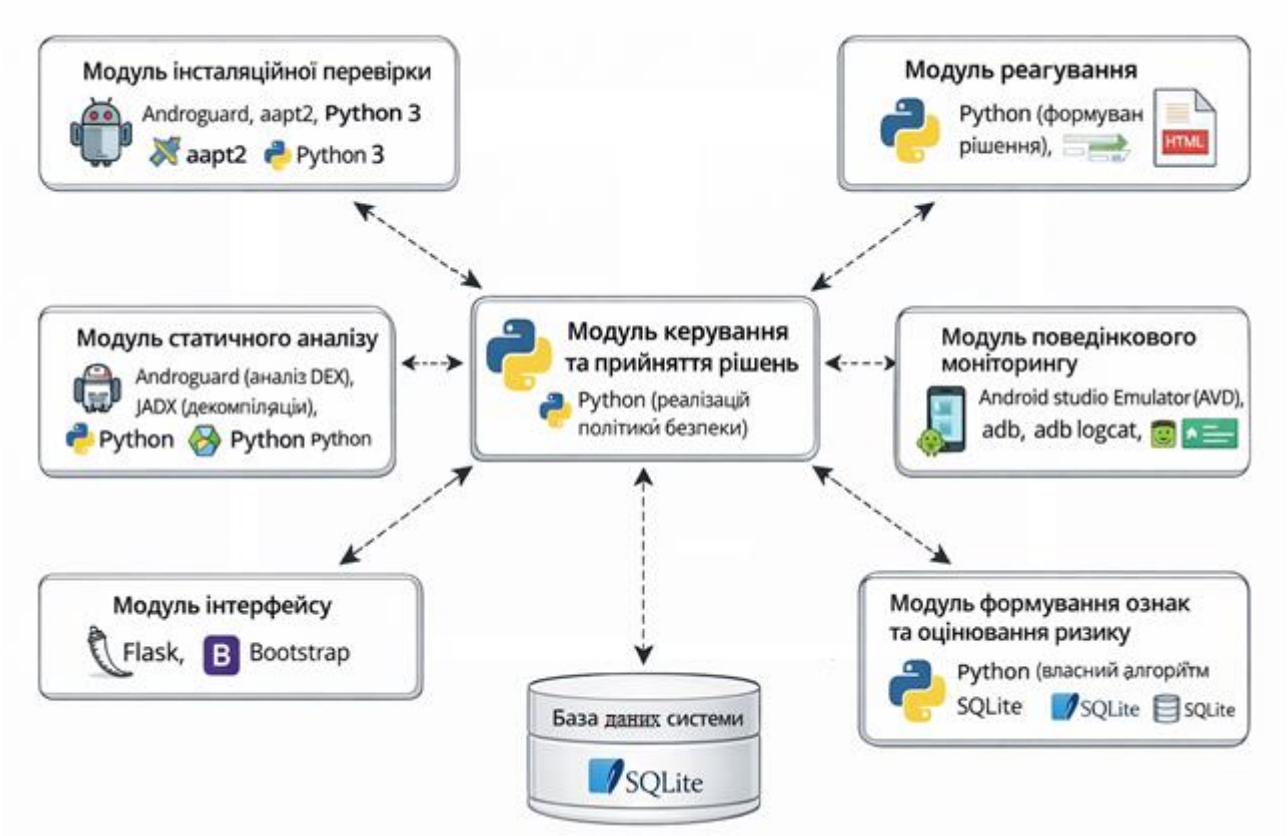


Рис. 3.1 – Архітектура прототипу системи виявлення шкідливих програм

Оскільки частина шкідливих механізмів активується лише під час виконання програми, у структурі прототипу передбачено модуль поведінкового моніторингу. Він функціонує під час запуску застосунку та забезпечує фіксацію фактичних дій, зокрема звернень до конфіденційних ресурсів, ініціювання мережевих з’єднань, створення фонових служб або спроб зміни системних параметрів. Зібрані події реєструються у журналі та передаються до модуля формування ознак. Таким чином система здатна виявляти загрози, що не можуть бути ідентифіковані виключно за допомогою статичного аналізу.

Таблиця 3.1 – Реалізація модулів прототипу

Модуль системи	**Інструменти реалізації**
Модуль інсталяційної перевірки	Androguard, aapt2, Python 3
Модуль статичного аналізу	Androguard (аналіз DEX), JADX (декомпіляція), apktool (ресурси), Python
Модуль поведінкового моніторингу	Android Studio Emulator (AVD), adb, adb logcat, adb shell dumpsys
Модуль формування ознак та оцінювання ризику	Python (власний алгоритм скорингу), SQLite
Модуль керування та прийняття рішень	Python (реалізація політики безпеки)
Модуль реагування	Python (формування рішення), генерація звіту (HTML)
Модуль інтерфейсу	Flask, Bootstrap
База даних системи	SQLite

Центральним елементом архітектури є модуль формування ознак і оцінювання ризику. У ньому агрегуються результати статичного та динамічного аналізу, здійснюється їх узагальнення та формування профілю застосунку. На основі визначених критеріїв або алгоритмів оцінювання розраховується інтегральний показник ризику, що дозволяє класифікувати застосунок як безпечний, підозрілий або шкідливий. Саме цей модуль реалізує основну складову системи.

Модуль прийняття рішення та реагування реалізує політику безпеки системи. Відповідно до отриманого рівня ризику він може дозволити встановлення або запуск застосунку, обмежити його функціональність або повністю заблокувати виконання. Реагування може здійснюватися як на етапі інсталяції, так і під час виконання застосунку, якщо зафіксовано небезпечну активність. Усі прийняті рішення реєструються та зберігаються для подальшого аналізу.

Для забезпечення узгодженої роботи всіх компонентів у прототипі реалізовано механізм координації та внутрішнього обміну даними. Він забезпечує передавання результатів аналізу між модулями, ініціювання процедур моніторингу та реагування, а також синхронізацію обробки подій. Окрему роль відіграє підсистема зберігання даних, що накопичує інформацію про проаналізовані застосунки, виявлені ознаки, події безпеки та прийняті рішення. Накопичені дані використовуються для повторного аналізу та удосконалення алгоритмів оцінювання ризику.

3.3 Тестування роботи системи

З метою перевірки ефективності розробленого прототипу системи виявлення шкідливих Android-застосунків було проведено експериментальне тестування у контрольованому середовищі емуляції мобільної операційної системи. Розгортання виконувалося відповідно до описаної раніше архітектури системи, що включає модуль інсталяційної перевірки, модуль статичного аналізу, модуль поведінкового моніторингу, модуль формування ознак та оцінювання ризику, модуль керування та прийняття рішень, модуль реагування, модуль інтерфейсу та базу даних системи. Тестування проводилося з використанням Android Studio Emulator (AVD) рис. 3.2. Для збору даних про виконання застосунків застосовувалися засоби adb, adb logcat та adb shell dumpsys. Збереження результатів аналізу здійснювалося в базі даних SQLite.

Для проведення експерименту було сформовано тестову вибірку з 40 Android-застосунків, з яких 20 були шкідливими, а 20 – легітимними. До категорії шкідливих належали зразки троянських програм, шпигунського програмного забезпечення та рекламних модулів, отримані з відкритих дослідницьких джерел. Легітимні застосунки було завантажено з офіційних репозиторіїв. Для кожного застосунку виконувалася послідовність етапів: аналіз APK-файлу на етапі інсталяції із витягом AndroidManifest.xml та переліку дозволів, статичний аналіз структури коду та використаних API, запуск застосунку в середовищі емуляції,

збір журналів подій, формування набору статичних і динамічних ознак, розрахунок інтегрального показника ризику та прийняття рішення щодо класифікації застосунку. Рішення формувалося відповідно до встановленого порогового значення ризику та реалізовувалося у вигляді рекомендації дозволити, обмежити або заблокувати застосунок.

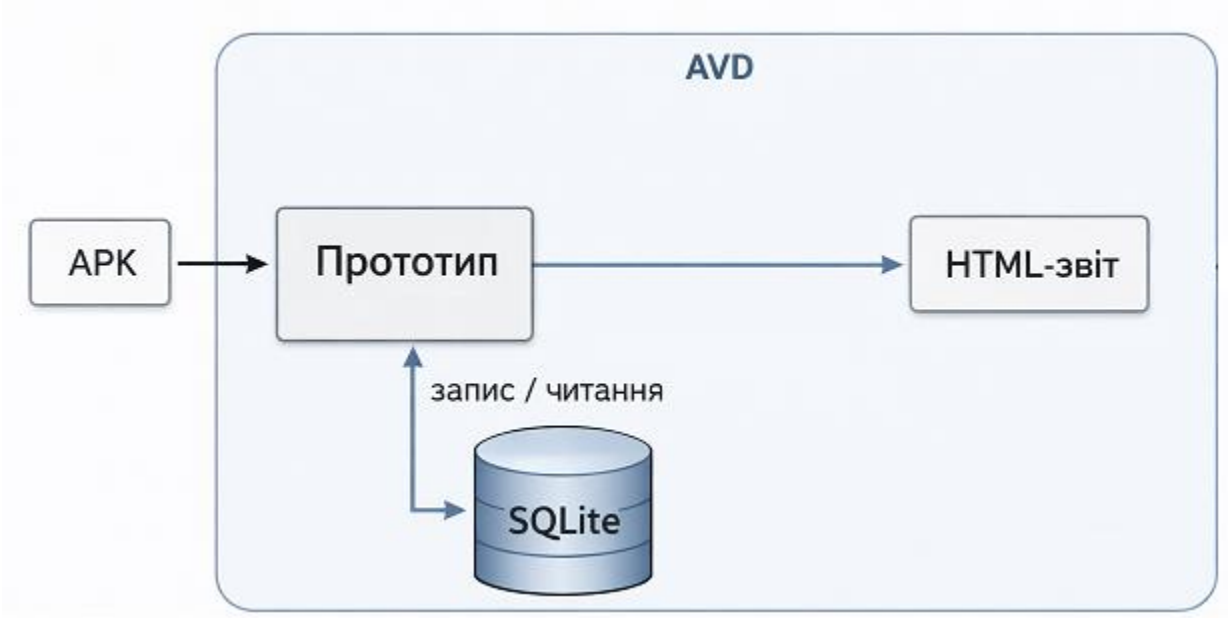


Рис. 3.2 – Схема тестового середовища прототипу

Оцінювання ефективності розробленого прототипу проводилося шляхом порівняння з двома поширеними засобами захисту мобільних пристроїв: Google Play Protect та ESET Mobile Security для Android. Google Play Protect є вбудованим механізмом безпеки операційної системи Android, що здійснює автоматичну перевірку встановлених застосунків на відповідність відомим загрозам із використанням сигнатурного та евристичного підходів. ESET Mobile Security є комерційним антивірусним рішенням, яке поєднує сигнатурний аналіз із локальними евристичними механізмами виявлення підозрілої активності.

Під час тестування Google Play Protect коректно виявив 15 шкідливих застосунків із 20, не продемонструвавши хибних спрацювань, однак не ідентифікував 5 зразків, що не містили відомих сигнатур або проявляли шкідливу поведінку лише після активації. ESET Mobile Security коректно виявив 17 шкідливих застосунків, допустивши одне хибне спрацювання щодо легітимного

застосунку з розширеним набором дозволів, та не виявив 3 шкідливі зразки. Розроблений прототип коректно ідентифікував 18 шкідливих застосунків, допустивши одне хибне спрацювання та не виявивши 2 зразки. Зведені результати тестування наведено в таблиці 3.2.

Таблиця 3.2 – Результати тестування прототипу системи

	Кількість коректно виявлених шкідливих програм	**Кількість хибних спрацювань**	**Кількість не виявлених**
Google Play Protect	15	-	5
ESET Mobile Security	17	1	3
Розроблений прототип	18	1	2

Отримані результати свідчать про те, що використання комбінованого підходу, який поєднує аналіз дозволів, статичне дослідження структури застосунку та поведінковий моніторинг під час виконання, дозволяє підвищити ефективність виявлення шкідливого програмного забезпечення порівняно з виключно сигнатурними методами. Зокрема, розроблений прототип продемонстрував здатність виявляти зразки, які не містили відомих сигнатур, але проявляли характерні ознаки шкідливої поведінки під час виконання в середовищі емуляції.

Додатково було проведено перевірку коректності функціонування кожного модуля системи окремо. Перевірка здійснювалася шляхом контрольного запуску тестових застосунків із заздалегідь відомими характеристиками та подальшого аналізу отриманих результатів.

Коректність роботи модуля інсталяційної перевірки перевірялася шляхом порівняння витягнутого переліку дозволів і службових параметрів застосунку з

фактичним вмістом файлу AndroidManifest.xml. Для цього використовувалися контрольні APK-файли, в яких були свідомо додані або видалені окремі дозволи. Результати, отримані за допомогою Androguard та aapt2, звірялися з декомпільованою структурою застосунку. Невідповідностей виявлено не було.

Перевірка модуля статичного аналізу здійснювалася шляхом аналізу застосунків, що містили відомі виклики критичних API (наприклад, доступ до SMS, контактів, файлової системи або мережевих ресурсів). Після декомпіляції коду за допомогою JADX та аналізу DEX-файлів перевірялося, чи коректно система фіксує відповідні виклики та формує статичні ознаки. Отримані результати відповідали реальній структурі коду.

Модуль поведінкового моніторингу тестувався шляхом запуску застосунків у середовищі Android Studio Emulator із виконанням сценарію взаємодії користувача. За допомогою adb logcat та dumpsys контролювалося, чи фіксуються події доступу до системних ресурсів, створення фонових процесів та ініціалізації мережевих з’єднань. Для перевірки використовувалися як легітимні застосунки з активною мережею взаємодією, так і зразки шкідливого ПЗ, що здійснювали приховану активність. Зібрані журнали подій відповідали фактичній поведінці застосунків.

Коректність модуля формування ознак та оцінювання ризику перевірялася шляхом контрольного розрахунку інтегрального показника ризику для декількох тестових випадків. Для цього вручну визначалися очікувані значення вагових коефіцієнтів на основі виявлених ознак, після чого результати порівнювалися з автоматично обчисленими системою значеннями. Відхилень у розрахунках не зафіксовано.

Модуль керування та прийняття рішень перевірявся шляхом варіювання порогового значення ризику та аналізу зміни класифікації застосунків. Було підтверджено, що при перевищенні встановленого порогу система формує рішення про обмеження або блокування, тоді як при низькому значенні ризику застосунок класифікується як безпечний.

Модуль реагування перевірявся шляхом контролю формування звітів у HTML-форматі та відображення результатів у інтерфейсі (рис. 3.3). Для кожного

протестованого застосунку генерувався звіт, що містив перелік виявлених ознак, значення показника ризику та підсумкове рішення. Вміст звіту відповідав фактичним результатам аналізу.

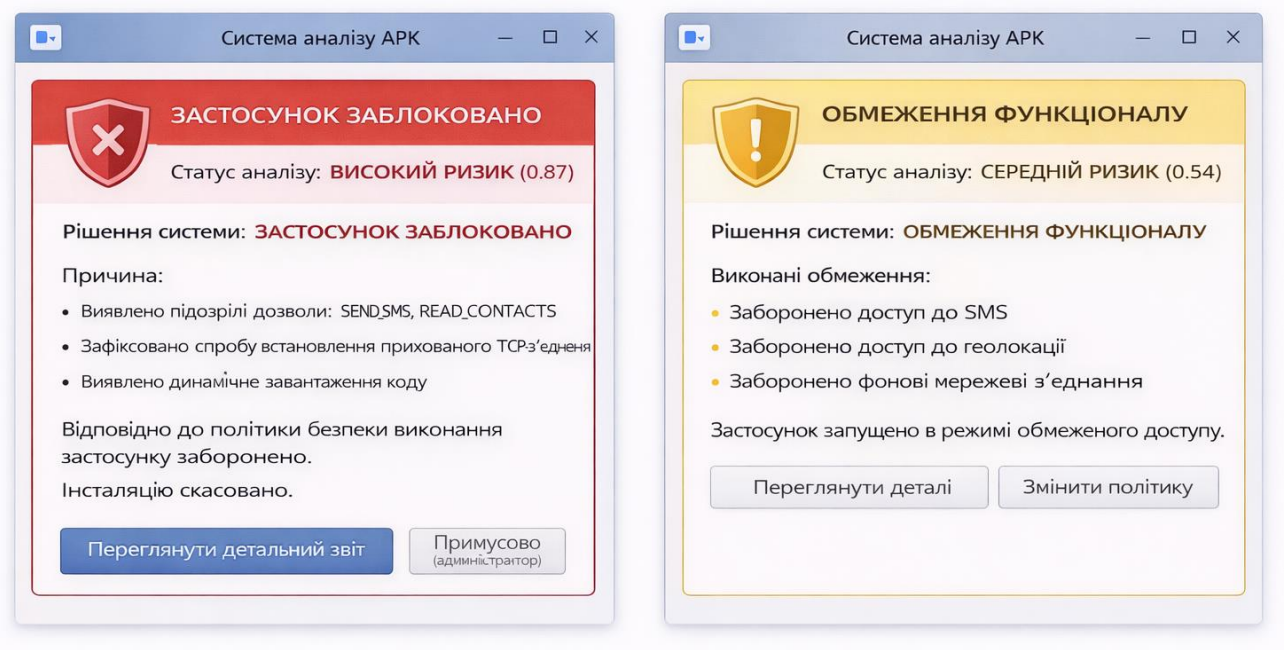


Рис. 3.3 – Приклад результатів роботи системи виявлення шкідливих програм

Таким чином, перевірка кожного функціонального модуля проводилася із застосуванням контрольних тестових випадків та порівнянням очікуваних і фактичних результатів. Отримані дані підтверджують коректність реалізації основних механізмів системи та узгодженість роботи всіх її компонентів.

3.4 Висновки до розділу

У третьому розділі здійснено розробку прототипу системи виявлення та блокування шкідливих програм у мобільній операційній системі, а також проведено її експериментальне дослідження.

На основі аналізу архітектури мобільної ОС та сучасних підходів до побудови систем виявлення загроз було сформовано сукупність функціональних, експлуатаційних та довірчих вимог до системи. Вимоги враховують особливості

моделі безпеки Android, необхідність інтеграції з механізмами sandbox та системою дозволів, а також положення міжнародних і національних стандартів у сфері інформаційної безпеки. Це дозволило визначити функціональну основу подальшого проєктування прототипу.

У межах проєктування було запропоновано модульну архітектуру системи, що включає модулі інсталяційної перевірки, статичного аналізу, поведінкового моніторингу, формування ознак та оцінювання ризику, керування і прийняття рішень, реагування, інтерфейсу та бази даних. Такий підхід забезпечує масштабованість, гнучкість і можливість подальшого вдосконалення окремих компонентів без зміни загальної структури системи. Центральним елементом архітектури визначено модуль оцінювання ризику, який агрегує результати статичного та динамічного аналізу та формує інтегральний показник ризику застосунку.

Реалізацію прототипу здійснено із використанням сучасних інструментів аналізу Android-застосунків та програмних засобів Python, що дозволило забезпечити автоматизацію процесів збору, обробки та збереження даних. Проведене тестування у середовищі емуляції підтвердило працездатність усіх функціональних модулів та коректність їх взаємодії.

За результатами експериментального дослідження встановлено, що розроблений прототип продемонстрував вищу ефективність виявлення шкідливих застосунків порівняно з окремими сигнатурними рішеннями, зокрема за рахунок поєднання статичного та поведінкового підходів. Отримані показники свідчать про доцільність використання комбінованого методу аналізу для підвищення рівня виявлення загроз у мобільному середовищі.

Таким чином, у розділі обґрунтовано вимоги до системи, розроблено її архітектуру, реалізовано програмний прототип та експериментально підтверджено його ефективність. Отримані результати створюють основу для подальшого вдосконалення алгоритмів оцінювання ризику, розширення бази даних та інтеграції системи у реальне мобільне середовище.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну задачу підвищення рівня захисту мобільних операційних систем від шкідливого програмного забезпечення шляхом розроблення системи виявлення на основі поєднання статичного та поведінкового аналізу.

У вступі обґрунтовано актуальність теми дослідження, зумовлену стрімким зростанням кількості шкідливих програм для мобільних пристроїв та обмеженістю можливостей традиційних сигнатурних методів виявлення. Визначено мету роботи – розроблення системи виявлення шкідливих програм, здатної функціонувати незалежно від наявності сигнатур у базі даних, із використанням поведінкових підходів.

У першому розділі проаналізовано сучасний стан захищеності мобільних операційних систем, зокрема архітектуру та механізми безпеки ОС Android, включаючи модель sandbox, систему дозволів, ізоляцію процесів і багаторівневу структуру програмного стеку. Показано, що, незважаючи на наявність вбудованих механізмів захисту, мобільні ОС залишаються вразливими до нових і модифікованих шкідливих програм.

У другому розділі досліджено множину шкідливих програм, визначено їх характерні поведінкові та структурні ознаки, а також сформовано експериментальну вибірку для подальшого аналізу. На основі отриманих результатів розроблено архітектуру системи виявлення, що передбачає поетапний аналіз застосунку на різних етапах його життєвого циклу та інтеграцію з механізмами безпеки мобільної ОС. Сформовано функційну модель системи, яка охоплює процеси первинного аналізу, статичного дослідження структури, поведінкового моніторингу, оцінювання ризику та прийняття рішення.

У третьому розділі розроблено прототип системи виявлення шкідливих програм, реалізовано його модульну архітектуру та проведено експериментальне тестування в контрольованому середовищі емуляції. За результатами тестування встановлено, що запропонований прототип продемонстрував вищу ефективність

виявлення шкідливих застосунків порівняно з окремими сигнатурними рішеннями, що підтверджує доцільність комбінованого підходу.

У процесі виконання роботи:

– проаналізовано архітектуру та механізми захисту мобільних операційних систем;

– досліджено структуру і поведінку шкідливих програм та визначено їх характерні ознаки;

– розроблено архітектуру системи виявлення шкідливих програм;

– реалізовано програмний прототип системи;

– проведено експериментальну перевірку ефективності розробленого рішення.

Отримані результати свідчать про те, що поєднання статичного аналізу структури застосунку з поведінковим моніторингом під час виконання дозволяє підвищити ефективність виявлення нових і модифікованих шкідливих програм у мобільних операційних системах.

Практична цінність роботи полягає у можливості використання розробленої архітектури та прототипу системи як основи для створення інтегрованих засобів захисту мобільних пристроїв або на рівні системних сервісів ОС Android, або на рівні гібридної архітектури: мобільний клієнт + серверна «пісочниця». Можливі подальші напрямки роботи полягають у вдосконаленні алгоритмів оцінювання ризику, розширенні бази даних щодо шкідливої поведінки застосунків і впровадженні інтелектуальних методів аналізу в реальні мобільні середовища.

ПЕРЕЛІК ДЖЕРЕЛ

1. Синявіна Ю. В. Використання алгоритмів машинного навчання для виявлення шкідливих програм в Android-системах на основі аномалій// Матеріали МНПК «Інформаційні технології в сучасному світі». Харків, ДБТУ, 2025. С.369-371.

2. Ковальова Т. І., Скоморохов В. А. Сучасні загрози інформаційній безпеці мобільних операційних систем Android та iOS// Національні інтереси України : наук.-практ. журнал. 2025. №12(17). С. 306-315.

3. Гоголь Б. В., Наталенко М. М. Спосіб протидії вразливостям додатків на базі операційної системи Android через управління дозволами// Сучасний захист інформації. 2022. №2(50). С.45-51.

4. Lookout. Mobile Threat Landscape Report: Q3 2024 [Електронний ресурс]. Режим доступу: https://www.lookout.com/threat-intelligence/report/q3-2024-mobile-landscape-threat-report-copy

5. Zimperium. Mobile Phishing Attacks Targeting Enterprises Surge, Zimperium Researchers Find [Електронний ресурс]. Режим доступу: https://s24.q4cdn.com/538403808/files/doc_news/Mobile-Phishing-Attacks-Targeting-Enterprises-Surge-Zimperium-Researchers-Find-2024.pdf

6. Сосновий, В., & Лащевська, Н. (2024). Виявлення шкідливої діяльності з використанням нейронної мережі для безперервної роботи. Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 3(23), 213-224. https://doi.org/10.28925/2663-4023.2024.23.213224

7. Android [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Android

8. Понад 200 шкідливих застосунків у Google Play завантажили 8 млн разів: які найпоширеніші загрози (з посиланням на дані Zscaler) [Електронний ресурс]. Режим доступу: https://dev.ua/news/u-2024-rotsi-ponad-200-shkidlyvykh-zastosunkiv-u-google-play-zavantazhyly-8-mln-raziv-iak-naiposhyrenishi-zahrozy-1729065957

9. Google. Android Security Paper 2024 [Електронний ресурс]. Режим доступу: https://services.google.com/fh/files/misc/android-security-paper-2024.pdf

10. Google Security Blog. How we kept the Google Play & Android app ecosystems safe in 2024 [Електронний ресурс]. – 29.01.2025. – Режим доступу: https://security.googleblog.com/2025/01/how-we-kept-google-play-android-app-ecosystem-safe-2024.html

11. Zscaler. Zscaler Identifies More Than 200 Malicious Apps in the Google Play Store with Over 8 Million Installs [Електронний ресурс]. Режим доступу: https://www.zscaler.com/press/zscaler-identifies-more-200-malicious-apps-google-play-store-over-8-million-installs

12. ESET. ESET Threat Report H2 2024 [Електронний ресурс]. Режим доступу: https://web-assets.esetstatic.com/wls/en/papers/threat-reports/eset-threat-report-h22024.pdf

13. Державна служба спеціального зв'язку та захисту інформації України, Державний центр кіберзахисту. Системи виявлення вразливостей і реагування на кіберінциденти та кібератаки. 2024. Річний звіт [Електронний ресурс]. Режим доступу: https://scpc.gov.ua/api/files/72e13298-4d02-40bf-b436-46d927c88006

14. Avast. Avast Q1/2024 threat report [Електронний ресурс]. Режим доступу: https://www.gendigital.com/blog/insights/reports/avast-q1-2024-threat-report

15. Розроблення мобільних застосунків: конспект лекцій. Навчальний посібник/ Київ: КПІ ім. Ігоря Сікорського, 2023. 546 с.

16. Google. Android Architecture Overview [[Електронний ресурс]. Режим доступу: https://source.android.com/docs/core/architecture

17. Тема 2. Огляд сучасних ОС для мобільних пристроїв (Android на ядрі Linux; Dalvik/DEX): навчальні матеріали [Електронний ресурс]. Луцьк: ЛНТУ. Режим доступу: https://e-tk.lntu.edu.ua/mod/resource/view.php?id=10868

18. Проєктування додатків для мобільних пристроїв: навчальний посібник/ О. І. Пушкар, Є. М. Грабовський. Харків : ХНЕУ ім. С. Кузнеця, 2023. 167 с.

19. Основи операційних систем: навч. посібн./ О. С. Головня. Житомир: «Житомирська політехніка», 2023. 126 с.

20. Enck W., Ongtang M., McDaniel P. Understanding Android Security// IEEE Security & Privacy. 2009. Vol. 7, No. 1. P. 50–57. DOI: https://doi.org/10.1109/MSP.2009.26

21. Yajin Z., Xuxian J. Dissecting Android Malware: Characterization and Evolution // IEEE Symposium on Security and Privacy. 2012. DOI: https://doi.org/10.1109/SP.2012.16

22. Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. 140 с.

23. Програмування мобільних пристроїв та систем: Методичні вказівки до виконання самостійних робіт/ Укл.: В.В. Смірнов, Н.В. Смірнова. Кропивницький : ЦНТУ, 2019 р. 49 с.

24. Методичні вказівки до виконання лабораторних робіт з дисципліни «Програмування мобільних пристроїв» / Укл.: Заставний О.М. Тернопіль: ТНЕУ, 2019. 89 с.

25. Android Developers. App manifest overview [Електронний ресурс]. Режим доступу: https://developer.android.com/guide/topics/manifest/manifest-intro

26. Android Developers. <manifest> (manifest-element) [Електронний ресурс]. Режим доступу: https://developer.android.com/guide/topics/manifest/manifest-element

27. Vanderbilt University mirror (Android docs). The AndroidManifest.xml File [Електронний ресурс]. Режим доступу: https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/guide/topics/manifest/manifest-intro.html

28. Android Developers. Sign your app [Електронний ресурс]. Режим доступу: https://developer.android.com/studio/publish/app-signing

29. Android Open Source Project. App signing [Електронний ресурс]. Режим доступу: https://source.android.com/docs/security/features/apksigning

30. Програмування мобільних пристроїв: навчальний посібник для дистанційного навчання/ К.Т. Кузьма. Миколаїв: СПД Румянцева Г. В., 2021. 128 с.

31. Android Developers. Android Debug Bridge (adb) [Електронний ресурс]. Режим доступу: https://developer.android.com/tools/adb

32. Zimperium. Global Mobile Threat Report 2024 [Електронний ресурс]. Режим доступу: https://www.zimperium.com/global-mobile-threat-report/

33. Felt A. P., Egelman S., Wagner D. Android Permissions: User Attention, Comprehension, and Behavior// SOUPS 2012. DOI: https://doi.org/10.1145/2335356.2335360

34. Android Open Source Project. Verified Boot & APK Signing [Електронний ресурс]. Режим доступу: https://source.android.com/docs/security/features/verifiedboot

35. Google. Android Security & Privacy Year in Review 2023/2024 [Електронний ресурс]. Режим доступу: https://security.googleblog.com

36. ENISA. Threat Landscape 2024 [Електронний ресурс]. Режим доступу: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024

37. Zhou Y., Jiang X. Dissecting Android Malware: Characterization and Evolution // IEEE Symposium on Security and Privacy. 2012. DOI: https://doi.org/10.1109/SP.2012.16

38. Symantec (Broadcom). Internet Security Threat Report 2024 [Електронний ресурс]. Режим доступу: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence

39. Android Open Source Project. Application Sandbox [Електронний ресурс]. Режим доступу: https://source.android.com/docs/security/app-sandbox

40. Android Open Source Project. Binder overview [Електронний ресурс]. – Режим доступу: https://source.android.com/docs/core/architecture/ipc/binder-overview

41. Android Open Source Project. Security-Enhanced Linux in Android [Електронний ресурс]. Режим доступу: https://source.android.com/docs/security/features/selinux

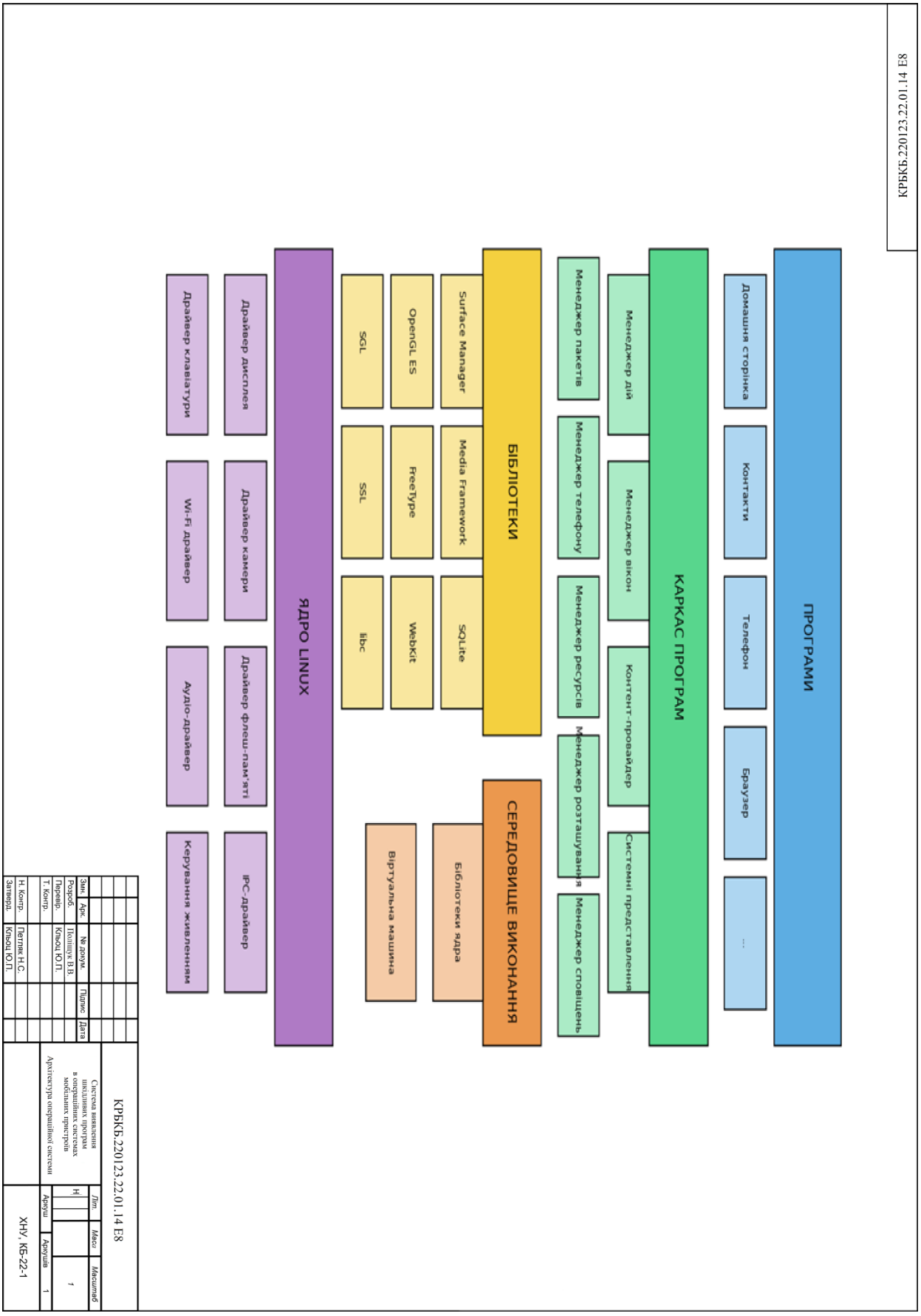
42. Android Developers. Manifest.permission | API reference [Електронний ресурс]. Режим доступу: https://developer.android.com/ reference/android/Manifest.permission

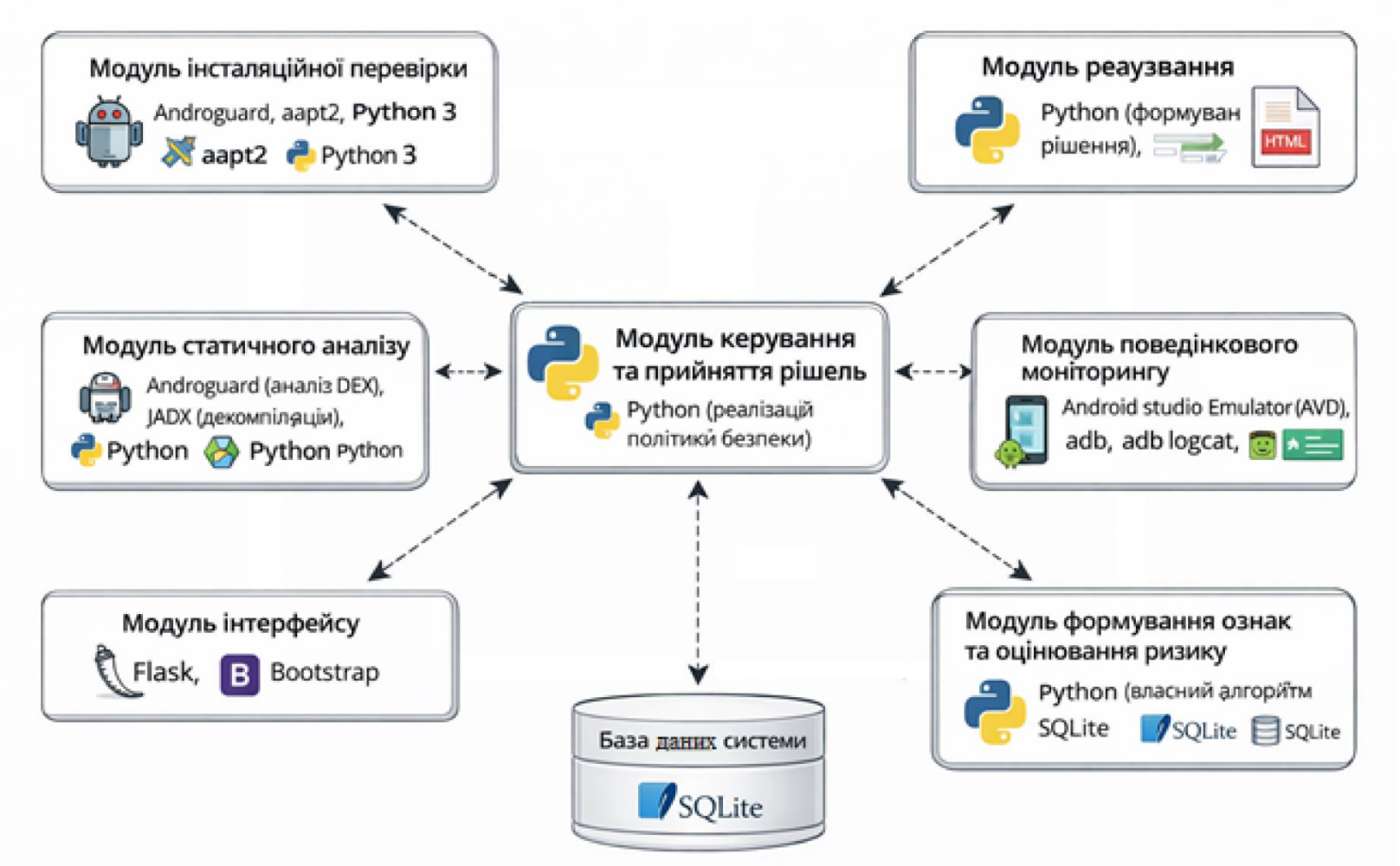
43. Tam K., Feizollah A., Anuar N. B., Salleh R., Cavallaro L. The Evolution of Android Malware and Android Analysis Techniques// ACM Computing Surveys. 2017. Vol. 49, No. 4. Article 76. DOI: https://doi.org/10.1145/3017427

44. Faruki P., Bharmal A., Laxmi V., Ganmoor V., Gaur M., Conti M., Rajarajan M. Android Security: A Survey of Issues, Malware Penetration, and Defenses// IEEE Communications Surveys & Tutorials. 2015. Vol. 17, No. 2. P. 998-1022. DOI: https://doi.org/10.1109/COMST.2014.2386139

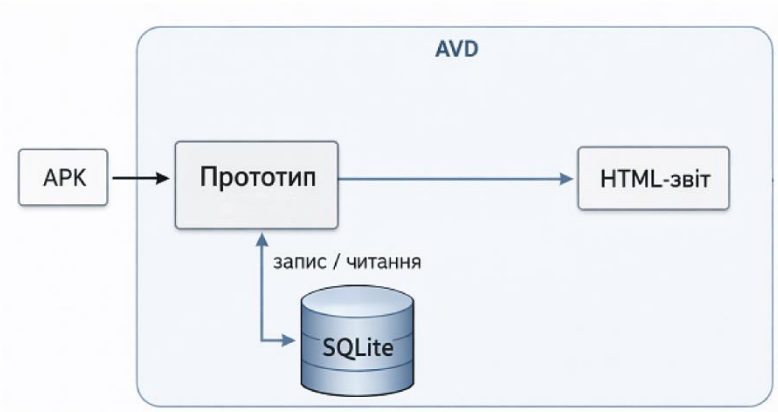
ДОДАТОК А

Копії графічної частини

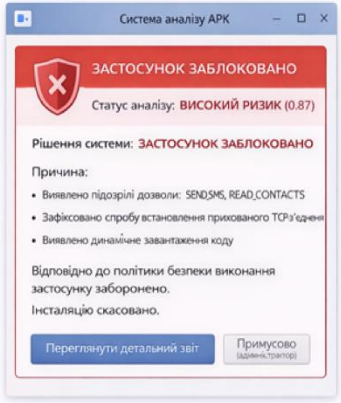


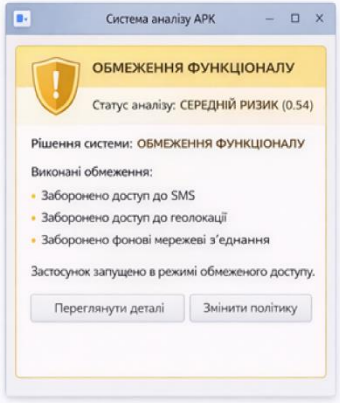


					КРБКБ.220123.22.01.14 Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливих програм в операційних системах мобільних пристроїв. Архітектура прототипу системи виявлення шкідливих програм	Літ. / Маси / Масштаб: Н / 1
Розроб.		Поліщук В.В.				Аркуш 1 / Аркушів 1
Перевір.		Кльоц Ю.П.				
Т. Контр.						ХНУ, КБ-22-1
Н. Контр.		Петляк Н.С.				
Затверд.		Кльоц Ю.П.				



	Кількість коректно виявлених шкідливих програм	Кількість хибних спрацювань	Кількість не виявлених
Google Play Protect	15	-	5
ESET Mobile Security	17	1	3
Розроблений прототип	18	1	2





					КРБКБ.220123.22.01.14 Е8	
Змн.	Арк.	№ докум.	Підпис	Дата	Система виявлення шкідливих програм в операційних системах мобільних пристроїв Результати тестування системи	Літ. / Маса / Масштаб
Розроб.		Поліщук В.В.				Н / 1
Перевір.		Кльоц Ю.П.				Аркуш / Аркушів 1
Т. Контр.						
Н. Контр.		Петляк Н.С.				ХНУ, КБ-22-1
Затверд.		Кльоц Ю.П.				

Додаток Б
Реалізація прототипу для тестування
(фрагменти програмного коду)

1) Структура проєкту

```
malware_detector/
  app.py
  wsgi.py
  config.py
  requirements.txt

  detector/
    __init__.py
    pipeline.py
    errors.py

    install_check/
      __init__.py
      install_check.py
      aapt.py

    static_analysis/
      __init__.py
      static_analyzer.py
      androguard_analyzer.py
      jadx.py
      apktool.py

    dynamic_analysis/
      __init__.py
      monitor.py
      adb.py
      emulator.py
      collectors.py
    features/
      __init__.py
      feature_builder.py
      risk_scoring.py

    policy/
      __init__.py
      policy_engine.py

    response/
      __init__.py
      responder.py
      report.py
      templates/
        report.html.j2

    db/
```

```
    __init__.py
    sqlite.py
    schema.sql
    repo.py

  web/
    __init__.py
    routes.py
    templates/
      index.html
      result.html
    static/
      css/
        main.css
```

2) requirements.txt

```
Flask==3.0.3
Werkzeug==3.0.3
Jinja2==3.1.4
androguard==4.1.2
```

3) config.py

```
from dataclasses import dataclass
from pathlib import Path

@dataclass(frozen=True)
class Settings:
    BASE_DIR: Path = Path(__file__).resolve().parent
    DATA_DIR: Path = BASE_DIR / "data"
    TOOLS_DIR: Path = BASE_DIR / "tools"

    SQLITE_PATH: Path = DATA_DIR / "detector.db"

    AAPT2: str = "aapt2"
    APKTOOL: str = "apktool"
    JADX: str = "jadx"
    ADB: str = "adb"
    EMULATOR: str = "emulator"

    AVD_NAME: str = "MalwareLab_AVD"
    EMULATOR_BOOT_TIMEOUT_SEC: int = 180
    DYNAMIC_RUN_SEC: int = 45

    REPORT_DIR: Path = DATA_DIR / "reports"
    ARTIFACTS_DIR: Path = DATA_DIR / "artifacts"

settings = Settings()
settings.DATA_DIR.mkdir(parents=True, exist_ok=True)
settings.REPORT_DIR.mkdir(parents=True, exist_ok=True)
settings.ARTIFACTS_DIR.mkdir(parents=True, exist_ok=True)
```

4) DB: schema.sql + sqlite.py + repo.py

```
PRAGMA foreign_keys = ON;

CREATE TABLE IF NOT EXISTS samples (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sha256 TEXT NOT NULL UNIQUE,
  filename TEXT NOT NULL,
  created_at TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS analyses (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sample_id INTEGER NOT NULL,
  status TEXT NOT NULL,          -- OK/FAILED
  risk_score REAL NOT NULL,
  risk_level TEXT NOT NULL,       -- LOW/MEDIUM/HIGH
  decision TEXT NOT NULL,         -- ALLOW/BLOCK/QUARANTINE
  summary TEXT,
  report_path TEXT,
  created_at TEXT NOT NULL,
  FOREIGN KEY(sample_id) REFERENCES samples(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS features (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  analysis_id INTEGER NOT NULL,
  key TEXT NOT NULL,
  value TEXT NOT NULL,
  FOREIGN KEY(analysis_id) REFERENCES analyses(id) ON DELETE CASCADE
);
CREATE INDEX IF NOT EXISTS idx_features_analysis_id ON features(analysis_id);

import sqlite3
from pathlib import Path

def connect(db_path: Path) -> sqlite3.Connection:
    conn = sqlite3.connect(str(db_path))
    conn.row_factory = sqlite3.Row
    conn.execute("PRAGMA foreign_keys = ON;")
    return conn

def init_db(conn: sqlite3.Connection, schema_path: Path) -> None:
    sql = schema_path.read_text(encoding="utf-8")
    conn.executescript(sql)
    conn.commit()

from __future__ import annotations
from dataclasses import dataclass
from datetime import datetime, timezone
import sqlite3
from typing import Any, Dict, Iterable, Optional

def _now() -> str:
```

```
    return datetime.now(timezone.utc).isoformat()

@dataclass
class SampleRow:
    id: int
    sha256: str
    filename: str
    created_at: str

@dataclass
class AnalysisRow:
    id: int
    sample_id: int
    status: str
    risk_score: float
    risk_level: str
    decision: str
    summary: str | None
    report_path: str | None
    created_at: str

class Repo:
    def __init__(self, conn: sqlite3.Connection):
        self.conn = conn

    def upsert_sample(self, sha256: str, filename: str) -> SampleRow:
        cur = self.conn.cursor()
        cur.execute(
            "INSERT OR IGNORE INTO samples(sha256, filename, created_at) VALUES(?,?,?)",
            (sha256, filename, _now())
        )
        self.conn.commit()
        row = cur.execute("SELECT * FROM samples WHERE sha256=?", (sha256,)).fetchone()
        return SampleRow(**dict(row))

    def create_analysis(self, sample_id: int, status: str, risk_score: float,
                        risk_level: str, decision: str, summary: str | None,
                        report_path: str | None) -> AnalysisRow:
        cur = self.conn.cursor()
        cur.execute(
            """INSERT INTO analyses(sample_id, status, risk_score, risk_level, decision,
summary, report_path, created_at)
               VALUES(?,?,?,?,?,?,?,?)""",
            (sample_id, status, float(risk_score), risk_level, decision, summary, report_path,
_now())
        )
        self.conn.commit()
        analysis_id = cur.lastrowid
        row = cur.execute("SELECT * FROM analyses WHERE id=?", (analysis_id,)).fetchone()
        return AnalysisRow(**dict(row))

    def add_features(self, analysis_id: int, features: Dict[str, Any]) -> None:
```

```
        cur = self.conn.cursor()
        cur.executemany(
            "INSERT INTO features(analysis_id, key, value) VALUES(?,?,?)",
            [(analysis_id, k, str(v)) for k, v in features.items()]
        )
        self.conn.commit()
```

5) Загальні помилки + утиліти

```
class DetectorError(Exception):
    pass

class ToolExecError(DetectorError):
    def __init__(self, tool: str, message: str):
        super().__init__(f"{tool}: {message}")
        self.tool = tool

class ValidationError(DetectorError):
    pass
```

6) Модуль інсталяційної перевірки (Androguard + aapt2)

```
from __future__ import annotations
import subprocess
from dataclasses import dataclass
from typing import List
from detector.errors import ToolExecError

@dataclass(frozen=True)
class AaptBadging:
    package_name: str
    version_name: str | None
    version_code: str | None
    sdk_min: str | None
    sdk_target: str | None

def run_aapt2_badging(aapt2_bin: str, apk_path: str) -> str:
    try:
        cp = subprocess.run(
            [aapt2_bin, "dump", "badging", apk_path],
            text=True, capture_output=True, check=False
        )
    except FileNotFoundError as e:
        raise ToolExecError("aapt2", f"not found: {e}") from e

    if cp.returncode != 0:
        raise ToolExecError("aapt2", cp.stderr.strip() or "badging failed")

    return cp.stdout

def parse_badging(text: str) -> AaptBadging:
    pkg = None
    vname = None
    vcode = None
```

```
    sdk_min = None
    sdk_target = None

    for line in text.splitlines():
        line = line.strip()
        if line.startswith("package:"):
            # package: name='com.x' versionCode='12' versionName='1.2'
            def _extract(key: str) -> str | None:
                token = f"{key}='"
                if token not in line:
                    return None
                return line.split(token, 1)[1].split("'", 1)[0]
            pkg = _extract("name")
            vcode = _extract("versionCode")
            vname = _extract("versionName")

        if line.startswith("sdkVersion:"):
            sdk_min = line.split(":", 1)[1].strip().strip("'")
        if line.startswith("targetSdkVersion:"):
            sdk_target = line.split(":", 1)[1].strip().strip("'")

    if not pkg:
        raise ValueError("package name not found in badging")

    return AaptBadging(pkg, vname, vcode, sdk_min, sdk_target)

from __future__ import annotations
from dataclasses import dataclass
from hashlib import sha256
from pathlib import Path
from typing import Dict, Any

from androguard.core.apk import APK
from detector.errors import ValidationError
from .aapt import run_aapt2_badging, parse_badging

@dataclass(frozen=True)
class InstallCheckResult:
    sha256: str
    package_name: str
    is_signed: bool
    cert_subject: str | None
    manifest_permissions: list[str]
    meta: Dict[str, Any]

def hash_file(path: Path) -> str:
    h = sha256()
    with path.open("rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024), b""):
            h.update(chunk)
    return h.hexdigest()
```

```
def installation_check(apk_path: Path, aapt2_bin: str) -> InstallCheckResult:
    if not apk_path.exists() or apk_path.suffix.lower() != ".apk":
        raise ValidationError("Provided file is not an .apk or does not exist")

    digest = hash_file(apk_path)

    apk = APK(str(apk_path))
    if not apk.is_valid_APK():
        raise ValidationError("APK structure looks invalid/corrupted")

    perms = sorted(set(apk.get_permissions() or []))

    # signature/cert (simplified)
    is_signed = bool(apk.get_signature_names())
    cert_subject = None
    try:
        certs = apk.get_certificates()
        if certs:
            cert_subject = getattr(certs[0], "subject", None)
            cert_subject = str(cert_subject) if cert_subject else None
    except Exception:
        cert_subject = None

    badging_txt = run_aapt2_badging(aapt2_bin, str(apk_path))
    badging = parse_badging(badging_txt)

    meta = {
        "version_name": badging.version_name,
        "version_code": badging.version_code,
        "sdk_min": badging.sdk_min,
        "sdk_target": badging.sdk_target,
        "app_name": apk.get_app_name(),
    }

    return InstallCheckResult(
        sha256=digest,
        package_name=badging.package_name,
        is_signed=is_signed,
        cert_subject=cert_subject,
        manifest_permissions=perms,
        meta=meta
    )
```

7) Модуль статичного аналізу (Androguard DEX + JADX + apktool)

```
import subprocess
from pathlib import Path
from detector.errors import ToolExecError

def decompile_with_jadx(jadx_bin: str, apk_path: Path, out_dir: Path) -> Path:
    out_dir.mkdir(parents=True, exist_ok=True)
    cp = subprocess.run(
        [jadx_bin, "-d", str(out_dir), str(apk_path)],
```

```
        text=True, capture_output=True, check=False
    )
    if cp.returncode != 0:
        raise ToolExecError("jadx", cp.stderr.strip() or "decompile failed")
    return out_dir

from __future__ import annotations
from dataclasses import dataclass
from typing import List, Set

from androguard.misc import AnalyzeAPK

SUSPICIOUS_API_KEYWORDS = [
    "getDeviceId", "getSubscriberId", "SmsManager", "sendTextMessage",
    "Runtime.exec", "DexClassLoader", "loadClass", "setComponentEnabledSetting",
]

@dataclass(frozen=True)
class DexInsights:
    suspicious_api_hits: List[str]
    declared_components: dict

def analyze_dex(apk_path: str) -> DexInsights:
    a, d, dx = AnalyzeAPK(apk_path)

    hits: Set[str] = set()
    for m in dx.get_methods():
        name = m.name
        if any(k.lower() in name.lower() for k in SUSPICIOUS_API_KEYWORDS):
            hits.add(name)

    comps = {
        "activities": a.get_activities() or [],
        "services": a.get_services() or [],
        "receivers": a.get_receivers() or [],
        "providers": a.get_providers() or [],
    }
    return DexInsights(suspicious_api_hits=sorted(hits), declared_components=comps)

from __future__ import annotations
from dataclasses import dataclass
from pathlib import Path
from typing import Any, Dict

from .androguard_analyzer import analyze_dex
from .jadx import decompile_with_jadx
from .apktool import decode_resources

@dataclass(frozen=True)
class StaticAnalysisResult:
    dex: Dict[str, Any]
    jadx_dir: Path
```

```
    apktool_dir: Path

def static_analysis(apk_path: Path, jadx_bin: str, apktool_bin: str, artifacts_dir: Path) ->
StaticAnalysisResult:
    jadx_out = artifacts_dir / "jadx"
    apktool_out = artifacts_dir / "apktool"

    dex_insights = analyze_dex(str(apk_path))
    decompile_with_jadx(jadx_bin, apk_path, jadx_out)
    decode_resources(apktool_bin, apk_path, apktool_out)

    dex_payload = {
        "suspicious_api_hits": dex_insights.suspicious_api_hits,
        "components": dex_insights.declared_components,
        "suspicious_api_hit_count": len(dex_insights.suspicious_api_hits),
        "service_count": len(dex_insights.declared_components.get("services", [])),
        "receiver_count": len(dex_insights.declared_components.get("receivers", [])),
    }

    return StaticAnalysisResult(
        dex=dex_payload,
        jadx_dir=jadx_out,
        apktool_dir=apktool_out
    )
```

8) Модуль поведінкового моніторингу (AVD + adb + logcat + dumpsys)

```
import subprocess
from detector.errors import ToolExecError

def adb(adb_bin: str, *args: str, timeout: int = 30) -> str:
    cp = subprocess.run([adb_bin, *args], text=True, capture_output=True, timeout=timeout,
check=False)
    if cp.returncode != 0:
        raise ToolExecError("adb", cp.stderr.strip() or f"command failed: {' '.join(args)}")
    return cp.stdout

def adb_shell(adb_bin: str, cmd: str, timeout: int = 30) -> str:
    return adb(adb_bin, "shell", cmd, timeout=timeout)

import subprocess, time
from detector.errors import ToolExecError
from .adb import adb_shell, adb

def start_emulator(emulator_bin: str, avd_name: str) -> subprocess.Popen:
    try:
        # -no-snapshot -wipe-data можна вмикати в лабораторії
        return subprocess.Popen([emulator_bin, "-avd", avd_name], stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)
    except FileNotFoundError as e:
        raise ToolExecError("emulator", f"not found: {e}") from e

def wait_boot(adb_bin: str, timeout_sec: int) -> None:
```

```
    deadline = time.time() + timeout_sec
    while time.time() < deadline:
        try:
            out = adb_shell(adb_bin, "getprop sys.boot_completed", timeout=10).strip()
            if out == "1":
                return
        except Exception:
            pass
        time.sleep(2)
    raise ToolExecError("emulator/adb", "device did not boot in time")

def stop_emulator(adb_bin: str) -> None:
    try:
        adb(adb_bin, "emu", "kill", timeout=10)
    except Exception:
        # best effort
        pass
```

```
from __future__ import annotations
from dataclasses import dataclass
from detector.dynamic_analysis.adb import adb_shell, adb

@dataclass(frozen=True)
class DynamicObservations:
    logcat: str
    dumpsys_activity: str
    dumpsys_network: str

def collect(adb_bin: str) -> DynamicObservations:
    # logcat (one-shot)
    logcat = adb(adb_bin, "logcat", "-d", timeout=30)

    dumpsys_activity = adb_shell(adb_bin, "dumpsys activity activities", timeout=30)
    dumpsys_network = adb_shell(adb_bin, "dumpsys connectivity", timeout=30)

    return DynamicObservations(
        logcat=logcat,
        dumpsys_activity=dumpsys_activity,
        dumpsys_network=dumpsys_network
    )
```

```
from __future__ import annotations
import time
from dataclasses import dataclass
from pathlib import Path

from detector.dynamic_analysis.emulator import start_emulator, wait_boot, stop_emulator
from detector.dynamic_analysis.adb import adb, adb_shell
from detector.dynamic_analysis.collectors import collect, DynamicObservations

@dataclass(frozen=True)
class DynamicAnalysisResult:
```

```
    observations: DynamicObservations
    indicators: dict

def dynamic_analysis(apk_path: Path, package_name: str, adb_bin: str, emulator_bin: str,
                     avd_name: str, run_sec: int, boot_timeout_sec: int) -> DynamicAnalysisResult:
    # start emulator
    proc = start_emulator(emulator_bin, avd_name)
    try:
        adb(adb_bin, "wait-for-device", timeout=boot_timeout_sec)
        wait_boot(adb_bin, boot_timeout_sec)

        # install & launch
        adb(adb_bin, "install", "-r", str(apk_path), timeout=120)
        adb_shell(adb_bin, f"monkey -p {package_name} -c android.intent.category.LAUNCHER
1", timeout=30)

        # let it run
        time.sleep(run_sec)

        obs = collect(adb_bin)

        # crude indicators extraction (real system would parse)
        indicators = {
            "logcat_contains_sms": ("SmsManager" in obs.logcat) or ("SEND_SMS" in obs.logcat),
            "logcat_contains_dexload": ("DexClassLoader" in obs.logcat) or ("ClassLoader" in
obs.logcat),
            "network_mentions": ("Network" in obs.dumpsys_network) or ("CONNECTED" in
obs.dumpsys_network),
        }

        return DynamicAnalysisResult(observations=obs, indicators=indicators)

    finally:
        stop_emulator(adb_bin)
        try:
            proc.terminate()
        except Exception:
            pass
```

9) Ознаки + ризик-скоринг (Python + SQLite)

```
from __future__ import annotations
from typing import Any, Dict
from detector.install_check.install_check import InstallCheckResult
from detector.static_analysis.static_analyzer import StaticAnalysisResult
from detector.dynamic_analysis.monitor import DynamicAnalysisResult

HIGH_RISK_PERMS = {
    "android.permission.SEND_SMS",
    "android.permission.RECEIVE_SMS",
    "android.permission.READ_SMS",
    "android.permission.READ_CONTACTS",
    "android.permission.WRITE_CONTACTS",
```

```
    "android.permission.RECORD_AUDIO",
    "android.permission.CAMERA",
    "android.permission.READ_CALL_LOG",
    "android.permission.WRITE_CALL_LOG",
    "android.permission.REQUEST_INSTALL_PACKAGES",
}

def build_features(install: InstallCheckResult,
                   static: StaticAnalysisResult,
                   dynamic: DynamicAnalysisResult) -> Dict[str, Any]:
    perms = set(install.manifest_permissions)

    feats: Dict[str, Any] = {}
    feats["is_signed"] = install.is_signed
    feats["high_risk_perm_count"] = len(perms.intersection(HIGH_RISK_PERMS))
    feats["perm_requests_install_packages"] =
int("android.permission.REQUEST_INSTALL_PACKAGES" in perms)

    feats["suspicious_api_hit_count"] = static.dex.get("suspicious_api_hit_count", 0)
    feats["receiver_count"] = static.dex.get("receiver_count", 0)
    feats["service_count"] = static.dex.get("service_count", 0)

    feats["dyn_logcat_contains_sms"] = int(dynamic.indicators.get("logcat_contains_sms",
False))
    feats["dyn_logcat_contains_dexload"] =
int(dynamic.indicators.get("logcat_contains_dexload", False))
    feats["dyn_network_mentions"] = int(dynamic.indicators.get("network_mentions", False))

    return feats

from __future__ import annotations
from dataclasses import dataclass
from typing import Dict, Any

@dataclass(frozen=True)
class RiskScore:
    score: float
    level: str
    reasons: list[str]

def score_risk(features: Dict[str, Any]) -> RiskScore:
    score = 0.0
    reasons: list[str] = []

    # Signature
    if not bool(int(features.get("is_signed", 0)) if isinstance(features.get("is_signed"), str) else
features.get("is_signed")):
        score += 1.2
        reasons.append("APK без валідного підпису/сертифіката (підвищений ризик).")

    # Permissions
    hr = int(features.get("high_risk_perm_count", 0))
```

```
    score += min(2.0, hr * 0.35)
    if hr >= 4:
        reasons.append(f"Запитано багато чутливих дозволів: {hr}.")

    if int(features.get("perm_requests_install_packages", 0)) == 1:
        score += 1.4
        reasons.append("Запит REQUEST_INSTALL_PACKAGES (можлива інсталяція
сторонніх APK).")

    # Static API hits
    api_hits = int(features.get("suspicious_api_hit_count", 0))
    score += min(2.5, api_hits * 0.25)
    if api_hits >= 5:
        reasons.append(f"Підозрілі API-виклики у DEX: {api_hits}.")

    # Components
    receivers = int(features.get("receiver_count", 0))
    services = int(features.get("service_count", 0))
    if receivers >= 3:
        score += 0.7
        reasons.append("Багато BroadcastReceiver (можливий автозапуск/перехоплення
подій).")
    if services >= 2:
        score += 0.6
        reasons.append("Кілька Service (можлива фонова активність).")

    # Dynamic indicators
    if int(features.get("dyn_logcat_contains_sms", 0)) == 1:
        score += 1.8
        reasons.append("У logcat є ознаки SMS-активності.")
    if int(features.get("dyn_logcat_contains_dexload", 0)) == 1:
        score += 1.6
        reasons.append("Ознаки динамічного завантаження коду
(DexClassLoader/ClassLoader).")
    if int(features.get("dyn_network_mentions", 0)) == 1:
        score += 0.5
        reasons.append("Ознаки мережевої активності під час тесту.")

    # Normalize-ish
    score = round(min(score, 10.0), 2)

    if score >= 6.0:
        level = "HIGH"
    elif score >= 3.0:
        level = "MEDIUM"
    else:
        level = "LOW"

    return RiskScore(score=score, level=level, reasons=reasons)
```

10) Модуль керування та прийняття рішень (політика безпеки)

```
from __future__ import annotations
```

```
from dataclasses import dataclass
from detector.features.risk_scoring import RiskScore

@dataclass(frozen=True)
class Decision:
    decision: str
    summary: str

def decide(risk: RiskScore) -> Decision:
    if risk.level == "HIGH":
        return Decision("BLOCK", "Високий ризик: застосунок заблоковано")
    if risk.level == "MEDIUM":
        return Decision("QUARANTINE", "Середній ризик: обмеження функціоналу")
    return Decision("ALLOW", "Низький ризик: дозволити")
```

11) Модуль реагування + HTML-звіт

```
from __future__ import annotations
from dataclasses import dataclass
from pathlib import Path
from jinja2 import Environment, FileSystemLoader, select_autoescape

@dataclass(frozen=True)
class ReportContext:
    package_name: str
    sha256: str
    risk_score: float
    risk_level: str
    decision: str
    reasons: list[str]
    features: dict

def render_report(template_dir: Path, out_path: Path, ctx: ReportContext) -> Path:
    env = Environment(
        loader=FileSystemLoader(str(template_dir)),
        autoescape=select_autoescape(["html", "xml"])
    )
    tpl = env.get_template("report.html.j2")
    html = tpl.render(**ctx.__dict__)
    out_path.parent.mkdir(parents=True, exist_ok=True)
    out_path.write_text(html, encoding="utf-8")
    return out_path

<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <title>Malware Detector Report</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 24px; }
    .badge { display:inline-block; padding:6px 10px; border-radius: 10px; background:#eee; }
    .k { color:#555; width: 240px; display:inline-block; }
    .reason { margin: 6px 0; }
```

```
    pre { background:#f7f7f7; padding:12px; border-radius:8px; overflow:auto; }
  </style>
</head>
<body>
  <h1>Звіт аналізу APK</h1>

  <p><span class="k">Package:</span> <b>{{ package_name }}</b></p>
  <p><span class="k">SHA-256:</span> <code>{{ sha256 }}</code></p>

  <p>
    <span class="k">Risk:</span>
    <span class="badge">{{ risk_level }}</span>
    <span class="badge">score={{ risk_score }}</span>
    <span class="badge">decision={{ decision }}</span>
  </p>

  <h2>Причини</h2>
  {% if reasons and reasons|length > 0 %}
    <ul>
      {% for r in reasons %}
        <li class="reason">{{ r }}</li>
      {% endfor %}
    </ul>
  {% else %}
    <p>Явних причин не виявлено.</p>
  {% endif %}

  <h2>Ознаки</h2>
  <pre>{{ features | tojson(indent=2) }}</pre>

</body>
</html>
```

12) Pipeline orchestrator

```
from __future__ import annotations
from dataclasses import dataclass
from pathlib import Path

from detector.install_check.install_check import installation_check
from detector.static_analysis.static_analyzer import static_analysis
from detector.dynamic_analysis.monitor import dynamic_analysis
from detector.features.feature_builder import build_features
from detector.features.risk_scoring import score_risk
from detector.policy.policy_engine import decide
from detector.response.report import render_report, ReportContext

@dataclass(frozen=True)
class PipelineResult:
    sha256: str
    package_name: str
    risk_score: float
    risk_level: str
```

```
    decision: str
    summary: str
    report_path: Path
    features: dict

def run_pipeline(apk_path: Path, settings) -> PipelineResult:
    artifacts_dir = settings.ARTIFACTS_DIR / apk_path.stem
    artifacts_dir.mkdir(parents=True, exist_ok=True)

    install = installation_check(apk_path, settings.AAPT2)

    stat = static_analysis(
        apk_path=apk_path,
        jadx_bin=settings.JADX,
        apktool_bin=settings.APKTOOL,
        artifacts_dir=artifacts_dir / "static"
    )

    dyn = dynamic_analysis(
        apk_path=apk_path,
        package_name=install.package_name,
        adb_bin=settings.ADB,
        emulator_bin=settings.EMULATOR,
        avd_name=settings.AVD_NAME,
        run_sec=settings.DYNAMIC_RUN_SEC,
        boot_timeout_sec=settings.EMULATOR_BOOT_TIMEOUT_SEC
    )

    feats = build_features(install, stat, dyn)
    risk = score_risk(feats)
    decision = decide(risk)

    report_path = settings.REPORT_DIR / f"{install.sha256}.html"
    render_report(
        template_dir=Path(__file__).resolve().parent / "response" / "templates",
        out_path=report_path,
        ctx=ReportContext(
            package_name=install.package_name,
            sha256=install.sha256,
            risk_score=risk.score,
            risk_level=risk.level,
            decision=decision.decision,
            reasons=risk.reasons,
            features=feats,
        )
    )

    return PipelineResult(
        sha256=install.sha256,
        package_name=install.package_name,
        risk_score=risk.score,
        risk_level=risk.level,
```

```
            decision=decision.decision,
            summary=decision.summary,
            report_path=report_path,
            features=feats
        )
```

13) Flask UI (інтерфейс)

```
from pathlib import Path
from flask import Flask
from config import settings
from detector.db.sqlite import connect, init_db
from detector.db.repo import Repo
from web.routes import bp as web_bp

def create_app() -> Flask:
    app = Flask(__name__)
    app.config["MAX_CONTENT_LENGTH"] = 50 * 1024 * 1024  # 50MB

    conn = connect(settings.SQLITE_PATH)
    init_db(conn, schema_path=Path(__file__).resolve().parent / "detector" / "db" /
"schema.sql")
    app.repo = Repo(conn)  # type: ignore[attr-defined]
    app.settings = settings  # type: ignore[attr-defined]

    app.register_blueprint(web_bp)
    return app

app = create_app()

from __future__ import annotations
from pathlib import Path
from flask import Blueprint, current_app, render_template, request, redirect, url_for, send_file

from detector.pipeline import run_pipeline
from detector.errors import DetectorError

bp = Blueprint("web", __name__)

@bp.get("/")
def index():
    return render_template("index.html")

@bp.post("/scan")
def scan():
    f = request.files.get("apk")
    if not f or not f.filename:
        return render_template("index.html", error="Не вибрано файл APK"), 400

    settings = current_app.settings
    upload_path = settings.DATA_DIR / f.filename
    f.save(str(upload_path))
```

```
    try:
        result = run_pipeline(Path(upload_path), settings)
        repo = current_app.repo

        sample = repo.upsert_sample(result.sha256, f.filename)
        analysis = repo.create_analysis(
            sample_id=sample.id,
            status="OK",
            risk_score=result.risk_score,
            risk_level=result.risk_level,
            decision=result.decision,
            summary=result.summary,
            report_path=str(result.report_path)
        )
        repo.add_features(analysis.id, result.features)

        return render_template(
            "result.html",
            package_name=result.package_name,
            sha256=result.sha256,
            risk_score=result.risk_score,
            risk_level=result.risk_level,
            decision=result.decision,
            summary=result.summary,
            report_url=url_for("web.report", sha256=result.sha256),
        )

    except DetectorError as e:
        return render_template("index.html", error=str(e)), 500

@bp.get("/report/<sha256>")
def report(sha256: str):
    settings = current_app.settings
    path = settings.REPORT_DIR / f"{sha256}.html"
    if not path.exists():
        return "Report not found", 404
    return send_file(str(path), mimetype="text/html")
```

```
<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <title>APK Malware Detector</title>
</head>
<body>
  <h1>APK Malware Detector</h1>

  {% if error %}
    <p style="color:red;"><b>{{ error }}</b></p>
  {% endif %}

  <form method="post" action="/scan" enctype="multipart/form-data">
```

```
    <input type="file" name="apk" accept=".apk" required />
    <button type="submit">Сканувати</button>
  </form>
</body>
</html>

<!doctype html>
<html lang="uk">
<head>
  <meta charset="utf-8">
  <title>Результат</title>
</head>
<body>
  <h1>Результат аналізу</h1>
  <p><b>Package:</b> {{ package_name }}</p>
  <p><b>SHA-256:</b> {{ sha256 }}</p>
  <p><b>Risk:</b> {{ risk_level }} ({{ risk_score }})</p>
  <p><b>Decision:</b> {{ decision }}</p>
  <p>{{ summary }}</p>

  <p><a href="{{ report_url }}">Відкрити HTML-звіт</a></p>
  <p><a href="/">Назад</a></p>
</body>
</html>
```

Додаток В
Реалізація як Android-застосунку
(фрагменти програмного коду)

1. Архітектура додатку

```
com.example.mobiledetector
│
├── MainActivity.kt
├── analyzer/
│   PermissionAnalyzer.kt
│   RiskScorer.kt
├── model/
│   AppRisk.kt
├── ui/
│   AppAdapter.kt
```

2. AndroidManifest.xml

```
<manifest package="com.example.mobiledetector"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <uses-permission android:name="android.permission.QUERY_ALL_PACKAGES"/>

   <application
      android:label="Mobile Malware Detector"
      android:theme="@style/Theme.Material3.DayNight.NoActionBar">

      <activity
         android:name=".MainActivity"
         android:exported="true">
         <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter>
      </activity>

   </application>
</manifest>
```

3. Модель ризику (model/AppRisk.kt)

```
package com.example.mobiledetector.model

data class AppRisk(
   val packageName: String,
   val appName: String,
   val permissions: List<String>,
   val riskScore: Int,
   val verdict: String
)
```

4. Аналіз дозволів (analyzer/PermissionAnalyzer.kt)

```
package com.example.mobiledetector.analyzer
```

```
object PermissionAnalyzer {

    private val dangerousPermissions = mapOf(
        "android.permission.SEND_SMS" to 20,
        "android.permission.RECEIVE_SMS" to 15,
        "android.permission.READ_SMS" to 15,
        "android.permission.REQUEST_INSTALL_PACKAGES" to 25,
        "android.permission.SYSTEM_ALERT_WINDOW" to 18,
        "android.permission.READ_CONTACTS" to 10,
        "android.permission.RECORD_AUDIO" to 12,
        "android.permission.ACCESS_FINE_LOCATION" to 8,
        "android.permission.CAMERA" to 8,
        "android.permission.INTERNET" to 5
    )

    fun calculatePermissionScore(permissions: Array<String>?): Int {
        if (permissions == null) return 0

        var score = 0

        for (perm in permissions) {
            dangerousPermissions[perm]?.let {
                score += it
            }
        }

        return score
    }
}
```

5. Обчислення ризику (analyzer/RiskScorer.kt)

```
package com.example.mobiledetector.analyzer

object RiskScorer {

    fun determineVerdict(score: Int): String {
        return when {
            score >= 60 -> "ВИСОКИЙ РИЗИК"
            score >= 30 -> "СЕРЕДНІЙ РИЗИК"
            else -> "НИЗЬКИЙ РИЗИК"
        }
    }
}
```

6. Основна програма (MainActivity.kt)

```
package com.example.mobiledetector

import android.content.pm.PackageManager
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
```

```
import com.example.mobiledetector.analyzer.PermissionAnalyzer
import com.example.mobiledetector.analyzer.RiskScorer
import com.example.mobiledetector.model.AppRisk
import com.example.mobiledetector.ui.AppAdapter

class MainActivity : ComponentActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var adapter: AppAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        recyclerView = RecyclerView(this)
        recyclerView.layoutManager = LinearLayoutManager(this)
        setContentView(recyclerView)

        val results = analyzeInstalledApps()
        adapter = AppAdapter(results)
        recyclerView.adapter = adapter
    }

    private fun analyzeInstalledApps(): List<AppRisk> {

        val pm = packageManager
        val packages = pm.getInstalledPackages(PackageManager.GET_PERMISSIONS)

        val resultList = mutableListOf<AppRisk>()

        for (pkg in packages) {

            val permissions = pkg.requestedPermissions
            val score = PermissionAnalyzer.calculatePermissionScore(permissions)
            val verdict = RiskScorer.determineVerdict(score)

            val appName = pkg.applicationInfo.loadLabel(pm).toString()

            resultList.add(
                AppRisk(
                    packageName = pkg.packageName,
                    appName = appName,
                    permissions = permissions?.toList() ?: emptyList(),
                    riskScore = score,
                    verdict = verdict
                )
            )
        }

        return resultList.sortedByDescending { it.riskScore }
    }
}
```

7. Відображення звіту (ui/AppAdapter.kt)

```
package com.example.mobiledetector.ui

import android.graphics.Color
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.mobiledetector.model.AppRisk

class AppAdapter(private val apps: List<AppRisk>) :
    RecyclerView.Adapter<AppAdapter.ViewHolder>() {

    class ViewHolder(val textView: TextView) : RecyclerView.ViewHolder(textView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val tv = TextView(parent.context)
        tv.setPadding(20, 20, 20, 20)
        return ViewHolder(tv)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val app = apps[position]

        holder.textView.text =
            "${app.appName}\nScore: ${app.riskScore}\n${app.verdict}"

        holder.textView.setTextColor(
            when (app.verdict) {
                "ВИСОКИЙ РИЗИК" -> Color.RED
                "СЕРЕДНІЙ РИЗИК" -> Color.YELLOW
                else -> Color.GREEN
            }
        )
    }

    override fun getItemCount() = apps.size
}
```