

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

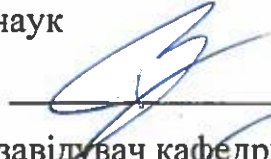
ПОЯСНЮВАЛЬНА ЗАПИСКА

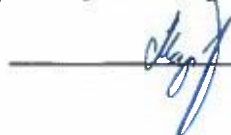
до випускної кваліфікаційної роботи магістра
на тему: «Методи та технології безпеки мобільних
застосунків»

Виконав студент 2-го курсу групи КБм-22-1
спеціальності 125 «Кібербезпека»

 В.О. Кобилинський

Керівник, доцент кафедри комп'ютерної
інженерії та кібербезпеки, кандидат
технічних наук

 О.В. Пірог
Рецензент, завідувач кафедри КН, PhD

 М.С. Граф

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ
Завідувач
кафедри комп'ютерної
інженерії та кібербезпеки
 Андрій Єфіменко
13 жовтня 2023 р.

ЗАВДАННЯ
на випускню кваліфікаційну роботу магістра

Студент: Кобилинський Вадим Олександрович

Тема роботи: «Методи та технології безпеки мобільних застосунків»
затверджена Наказом університету від 13 жовтня 2023 р. № 572/с

Термін здачі студентом закінченої роботи 27 грудня 2023 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

1. – Мобільний додаток

Обладнання та програмне забезпечення:

1. – Android Studio (IDE)
2. – Інструменти для пентестингу

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	М.С Граф	 1.09.2023	15.09.2023 
2	М.С Граф	 16.10.2023	30.10.2023 
3	М.С Граф	 1.12.2023	18.12.2023 

Студент  В.О Кобилинський
Керівник  О.В Пірог

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз отриманого завдання та уточнення поставлених завдань	05.05.2023 р.	<i>KB</i>
2	Аналіз літературних джерел	10.06.2023 р.	<i>KB</i>
3	Аналіз об'єкта дослідження	16.07.2023 р.	<i>KB</i>
4	Постановка задачі дослідження	20.08.2023 р.	<i>KB</i>
5	Розробка	10.09.2023 р.	<i>KB</i>
6	Тестування	15.10.2023 р.	<i>KB</i>
8	Перевірка працездатності	26.11.2023 р.	<i>KB</i>
9	Оформлення пояснювальної записки	20.12.2023 р.	<i>KB</i>
	Захист	Згідно розкладу	<i>KB</i>

Студент *KB* Кобилинський В.О

Керівник *KB* Пірог О. В.

АНОТАЦІЯ

Кваліфікаційна робота складається з методів та технологій безпеки мобільних застосунків та пояснювальної записки до нього. Пояснювальна записка містить у собі 58 сторінок і 25 зображень.

Метою роботи є розгляди та методи безпеки мобільних застосунків

В роботі визначено основні завдання на методи захисту, яка забезпечить коректну роботу без витоків інформації. Також в роботі проаналізовані та описані можливі варіанти для отримання несанкціонованого доступу. Обґрунтована актуальність даної тематики та описана сфера її використання. Проведено тестування та наведені методи безпеки та аналогічні йому.

Ключові слова: АНДРОЇД, БЕЗПЕКА, БЕЗПЕЧНА ПАМ'ЯТЬ, ПЕНТЕСТИНГ, РЕВЕРСИНГ, МОБІЛЬНА БЕЗПЕКА

ANNOTATION

The qualification work consists of methods and technologies for mobile application security and an explanatory note to it. The explanatory note contains 58 pages and 20 images.

The purpose of the work is to review the security considerations and methods for mobile applications

The paper defines the main tasks for security methods that will ensure the application's integrity and operation without information leaks. The paper also analyzes and describes possible options for gaining unauthorized access. The relevance of this topic is substantiated and the scope of its use is described. Testing is carried out and security methods and similar ones are presented.

Keywords: ANDROID, SECURITY, MOBILE, SECURITY MEMORY, OS

Зміст

ВСТУП	9
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ	11
1.1 ШИФРУВАННЯ ДАНИХ	11
1.1.1 FILE-BASED ENCRYPTION (FBE):	12
1.1.2 ANDROID KEYSTORE:	15
1.1.3 NETWORK SECURITY CONFIGURATION:	18
1.2 БІОМЕТРИЧНІ ПОРІВНЯННЯ	21
1.2.1 FACE RECOGNITION	21
1.2.2 FINGERPRINT RECOGNITION	23
1.2.3 VOICE RECOGNITION	28
1.2.4 IRIS RECOGNITION	30
1.3 OAUTH2 SERVICES	32
1.4 ВИСНОВОК ДО ПЕРШОГО РОЗДІЛУ	36
РОЗДІЛ 2 РОЗРОБКА СИСТЕМ ЗАХИСТУ ДЛЯ МОБІЛЬНОГО ДОДАТКУ	37
2.1 ІНТЕГРАЦІЯ БЕЗПЕЧНОЇ ВЗАЄМОДІЇ З СЕРВЕРОМ НА ANDROID	37
2.2 ЗАШИФРОВАНА БАЗА ДАНИХ	39
2.3 ЗАПЛУТАННЯ КОДУ ДЛЯ УСКЛАДНЕННЯ РЕВЕРСУ ДОДАТКУ (BINARY PROTECTIONS)	40
2.4 ЗАХИСТ ВІД SSL PINNING ATTACK	41
2.5 ЗАХИСТ ВІД SSL ATTACK НА ДЕВАЙСАХ ЗІ СТАРИМИ СЕРВІСАМИ	42
2.6 ROOTING HACK	43
2.7 ВІДКЛЮЧЕННЯ ВІДЛАГОДЖУВАЛЬНИХ МОЖЛИВОСТЕЙ	44
2.8 CODE INTEGRITY CHECKS (SAFETYNET)	45
2.9 TAMPER-EVIDENT LOGGING	45
2.10 DYNAMIC CODE LOADING RESTRICTIONS	46
2.11 MANIFEST SECURITY	47
2.12 ВИСНОВОК ДО ДРУГОГО РОЗДІЛУ	48
РОЗДІЛ 3 ПЕРЕВІРКА СИСТЕМ ЗАХИСТУ ДЛЯ МОБІЛЬНОГО ДОДАТКУ	49
3.1 ПЕНТЕСТИНГ MANIFEST ФАЙЛУ	49
3.2 BINARY PROTECTIONS – РЕВЕРС ІНЖИНИРИНГ ДОДАТКУ	51
3.3 ТЕСТУВАННЯ НА BACKDOOR	53
3.4 ТЕСТУВАННЯ З ВІДКРИТИМ ЛОГУВАННЯМ В ДОДАТКУ	54
3.5 НЕЗАШИФРОВАНЕ З’ЄДНАННЯ З СЕРВЕРОМ	56
3.6 НЕЗАХИЩЕНЕ ЗБЕРІГАННЯ ДАНИХ	57
3.7 ПРОБЛЕМИ З АВТОРИЗАЦІЄЮ	58
3.8 ВИСНОВОК ДО ТРЕТЬОГО РОЗДІЛУ	59
ВИСНОВКИ	60
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

SECURITY MEMORY – спеціальна пам'ять в андроїда до якої потрібний ключ

ANDROID KEYSTORE – спеціалізоване місце для збереження конфедіційних даних

PENTESTING – метод оцінювання захищеності комп'ютерної системи чи мережі шляхом часткового моделювання дій зовнішніх зловмисників з проникнення у неї

FBE – шифрує дані на рівня файла

FBE – необроблений ключ шифрування

AOSP – Android Open Source Project

TEE – Trusted Execution Environment

JDK (англ. java development kit) – комплект розробки додатків на мові Java, що включає в себе компілятор, стандартні бібліотеки класів, приклади, документацію та виконуючу систему

SDK (англ. software development kit) – набір засобів розробки, який дозволяє спеціалістам з програмного забезпечення створювати додатки для певного пакета програм, апаратної платформи, комп'ютерної системи, ігрових консолей, операційних систем

ADB (англ. android debug bridge) – засіб відладки, виявлення помилок в додатках, передачі даних між програмою та девайсом APK (англ. android package) – формат архівних виконуваних файлів додатків, який включає в себе весь код додатку, ресурси, активи, файл маніфесту та нативні бібліотеки

API (англ. Application Programming Interface) – набір готових класів, процедур, функцій, структур і констант, що надаються додатком (бібліотекою, сервісом) або операційною системою для використання у зовнішніх програмних продуктах

ВСТУП

Актуальність теми. Вражає актуальністю в контексті сучасного цифрового віку. Зростаючий інтерес до мобільних додатків створює ситуацію, коли загрози кібербезпеки збільшуються. Це пов'язано з різноманітністю дій, які користувачі виконують за допомогою мобільних застосунків та з постійним розвитком методів атак. Зростаюча кількість обміну конфіденційною інформацією через мобільні додатки підвищує їхню привабливість для кіберзлочинців. Додатковий тиск з боку регуляторів та технологічний прогрес створюють нові виклики у сфері безпеки мобільних застосунків. Таким чином, розробники і користувачі стикаються з важливістю застосування ефективних заходів безпеки для забезпечення захисту від потенційних загроз та збереження конфіденційності.

Метою кваліфікаційної роботи є розробка тестування безпеки мобільних додатків.

Визначена мета обумовлює наступні завдання:

- Розглянути методи захисту мобільного додатку, спрямовані на забезпечення конфіденційності, цілісності та доступності інформації.
- Шляхом впровадження сучасних систем виявлення ін'єкцій та механізмів автоматичного оновлення, забезпечити надійний захист інформації, що обробляється в додатку.
- Запропонувати практичний підхід, що дозволяє забезпечити ефективний захист мобільного додатку.
- Застосувати попередні кейси для тестування системи.

Об'єктом дослідження є мобільні додатки, які мають потенційні проблеми з безпекою.

Предметом дослідження є особливості технологій безпеки і методів покращення їх роботи.

Практичне значення результатів роботи. Одержані висновки та рекомендації нададуть основу для подальших покращень у сфері захисту від кіберзагроз, допомагаючи розробникам програмного забезпечення та іншим

зацікавленим сторонам створити більш безпечне та надійне середовище для користувачів Android-пристроїв.

Особистий внесок здобувача. Усі результати роботи отримані автором самостійно.

Структура роботи. Робота складається із трьох розділів, розподілених на структурні підрозділи. Загальний обсяг – 60 сторінок, 25 рисунків

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ

1.1 Шифрування даних

Як і більшість програмного забезпечення для шифрування дисків і файлів, шифрування пам'яті Android традиційно покладається на наявність необроблених ключів шифрування в системній пам'яті, щоб можна було виконати шифрування. Навіть якщо шифрування виконується спеціальним апаратним забезпеченням, а не програмним забезпеченням, програмне забезпечення зазвичай потребує керування необробленими ключами шифрування.

Традиційно це не вважається проблемою, оскільки ключі не будуть присутні під час офлайн-атаки, яка є основним типом атаки, від якої покликано захистити шифрування сховища. Однак існує бажання забезпечити посилений захист від інших типів атак, таких як атаки холодного завантаження та онлайн-атаки, коли зломисник може отримати витік системної пам'яті без повної шкоди для пристрою.

Щоб вирішити цю проблему, Android 11 запровадив підтримку *апаратних* ключів, де присутня апаратна підтримка. Ключі в апаратній упаковці – це ключі зберігання, які відомі спеціальному апаратному забезпеченню лише в необробленому вигляді; програмне забезпечення бачить і працює лише з цими ключами в загорнутій (зашифрованій) формі. Це апаратне забезпечення має бути здатне генерувати та імпортувати ключі зберігання, загортати ключі зберігання в ефемерні та довгострокові форми, отримувати підключі, напрямую програмувати один підключ у вбудований криптографічний механізм і повертати окремий підключ до програмного забезпечення.

Android надає різноманітні технології для шифрування даних, спрямовані на захист інформації, яка зберігається на мобільних пристроях. Ось декілька ключових технологій шифрування даних для Android

1.1.1 File-Based Encryption (FBE):

FBE шифрує дані на рівні файла, що надає вищий рівень безпеки. Кожен файл і каталог може бути шифрований окремо за допомогою унікального ключа.

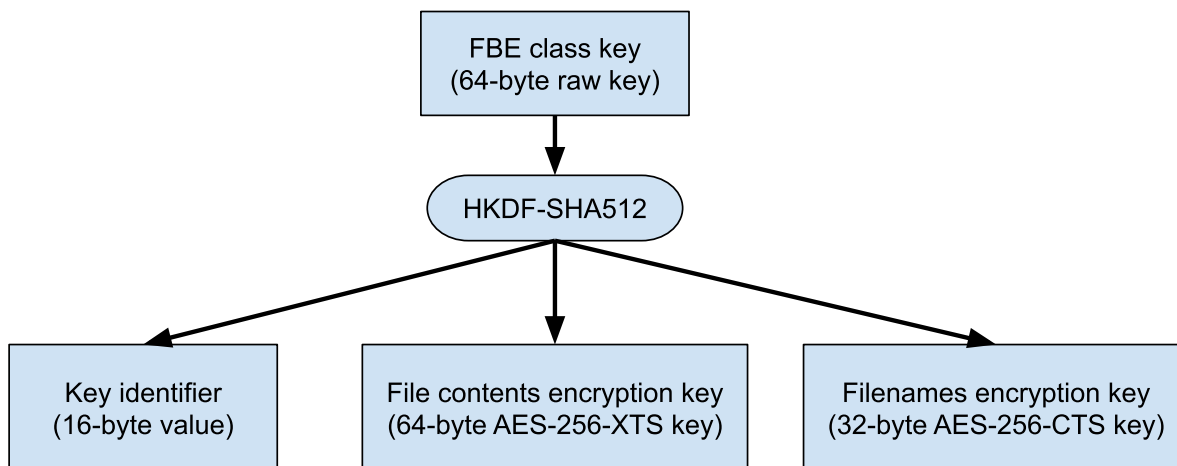


Рисунок 1.1 – Структура File-Based Encryption

Ключі можуть бути отримані з інших ключів за допомогою KDF (функція виведення ключів), наприклад HKDF, у результаті чого створюється ієрархія ключів

Ключ класу FBE — це необроблений ключ шифрування, який Android передає ядру Linux, щоб розблокувати певний набір зашифрованих каталогів, наприклад сховище, зашифроване обліковими даними для певного користувача Android. (У ядрі цей ключ називається головним ключем `fsencrypt`.) З цього ключа ядро отримує такі підключі:

- Ідентифікатор ключа. Це значення не використовується для шифрування, а для визначення ключа, яким захищено певний файл або каталог.
- Ключ шифрування вмісту файлу
- Ключ шифрування імен файлів

Для досягнення цілей безпеки ключів в апаратній упаковці визначено два типи упаковки ключів:

- Ефемерне обгортання: апаратне забезпечення шифрує необроблений ключ за допомогою ключа, який випадковим чином генерується під час кожного

завантаження та не розкривається безпосередньо за межами апаратного забезпечення.

- Довгострокова упаковка: апаратне забезпечення шифрує необроблений ключ за допомогою унікального постійного ключа, вбудованого в апаратне забезпечення, яке безпосередньо не розкривається за межами апаратного забезпечення.

Усі ключі, передані до ядра Linux для розблокування сховища, тимчасово запаковані. Це гарантує, що якщо зломиснику вдасться витягти використовуваний ключ із системної пам'яті, цей ключ стане непридатним для використання не лише поза пристроєм, а й на пристрої після перезавантаження.

У той же час Android все ще має мати можливість зберігати зашифровану версію ключів на диску, щоб їх можна було розблокувати в першу чергу. Необроблені ключі підійдуть для цієї мети. Однак бажано, щоб необроблені ключі взагалі ніколи не були присутні в системній пам'яті, щоб їх ніколи не можна було витягнути для використання поза пристроєм, навіть якщо видобуто під час завантаження. З цієї причини визначено поняття тривалого обгортання.

Щоб підтримувати керування ключами, обгорнутими цими двома різними способами, апаратне забезпечення має реалізовувати такі інтерфейси:

- Інтерфейси для створення та імпорту ключів зберігання, повертаючи їх у довгостроковій упаковці. Доступ до цих інтерфейсів здійснюється непрямо через KeyMint, і вони відповідають TAG_STORAGE_KEY тегу KeyMint. Можливість «генерувати» використовується для voldстворення нових ключів пам'яті для використання в Android, тоді як здатність «імпортувати» використовується для vts_kernel_encryption_test імпорту тестових ключів.
- Інтерфейс для перетворення ключа зберігання в довгостроковій упаковці на ключ зберігання в тимчасовій упаковці. Це відповідає convertStorageKeyToEphemeralметоду KeyMint. Цей метод використовується як voldдля vts_kernel_encryption_testтого, щоб розблокувати сховище.

Алгоритм упаковки ключів є деталлю реалізації, але він повинен використовувати потужний AEAD, такий як AES-256-GCM з випадковими IV. FDE шифрує всі дані на рівні блоків, включаючи системний розділ. Ключ шифрування зазвичай зберігається на засекреченій області пристрою.

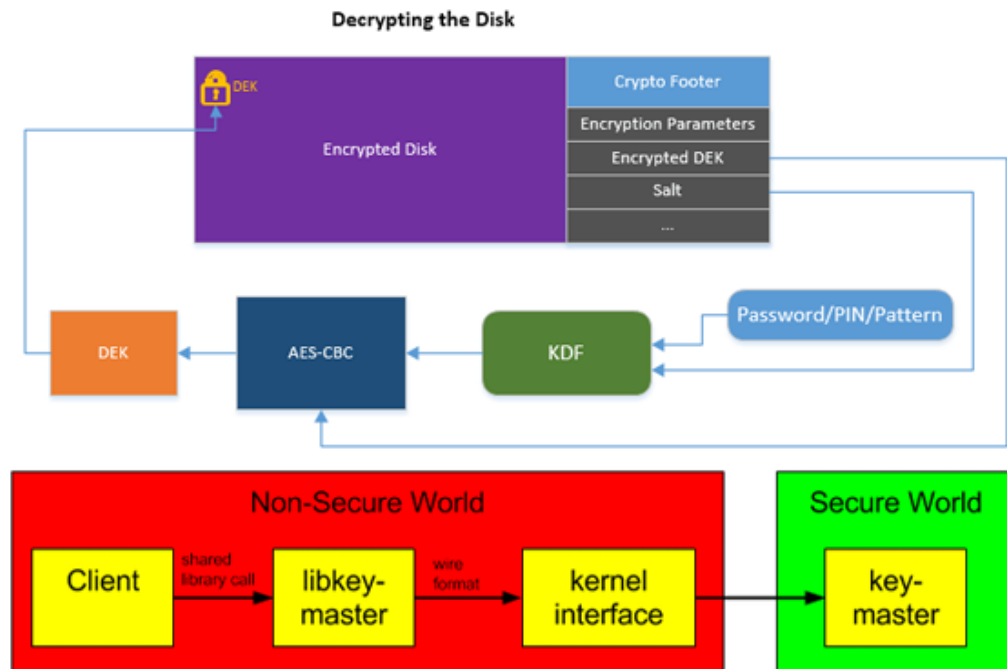


Рисунок 1.2 – Структура шифрування диску

Для пристроїв, які запускаються з Android 7.0 або новішої версії, розділ даних користувача зашифровано за замовчуванням. Щоб відповідати галузевим і урядовим вимогам безпеки, Samsung Knox будує FDE, щоб покращити впровадження Android Open Source Project (AOSP), використовуючи переваги апаратних механізмів безпеки та Trusted Execution Environment (TEE) на пристроях Samsung Galaxy.

На пристроях Android на основі FDE усі дані користувача шифруються за допомогою AES-256-XTS або AES-256-CBC (залежно від пристрою) за допомогою випадково згенерованого ключа шифрування, також відомого як первинний ключ. Після шифрування пристрою всі дані, створені користувачем, автоматично шифруються перед записом на диск і розшифровуються під час процесу читання.

Коли користувач завантажує пристрій уперше (наприклад, після скидання до заводських налаштувань), 256-бітний первинний ключ випадковим чином генерується та шифрується в пам'яті компонентом Keystore на основі TEE. Цей компонент використовує ключі, отримані з пароля за замовчуванням, і апаратного ключа, унікального для пристрою, для захисту від офлайн-атак. Якщо користувач встановлює PIN-код, пароль або шаблон на пристрої та налаштовує безпечний запуск, первинний ключ повторно шифрується Keystore за допомогою облікових даних користувача та зберігається. Якщо безпечний запуск не ввімкнено після налаштування, первинний ключ повторно шифрується за допомогою пароля за замовчуванням замість облікових даних користувача.

Під час наступних перезавантажень пристрою первинний ключ відновлюється або за допомогою пароля за замовчуванням, або за обліковими даними користувача, залежно від того, чи було налаштовано безпечний запуск, що дозволяє системі монтувати розділ даних користувача та розшифровувати його дані під час зчитування з флеш-пам'яті.

1.1.2 Android Keystore:

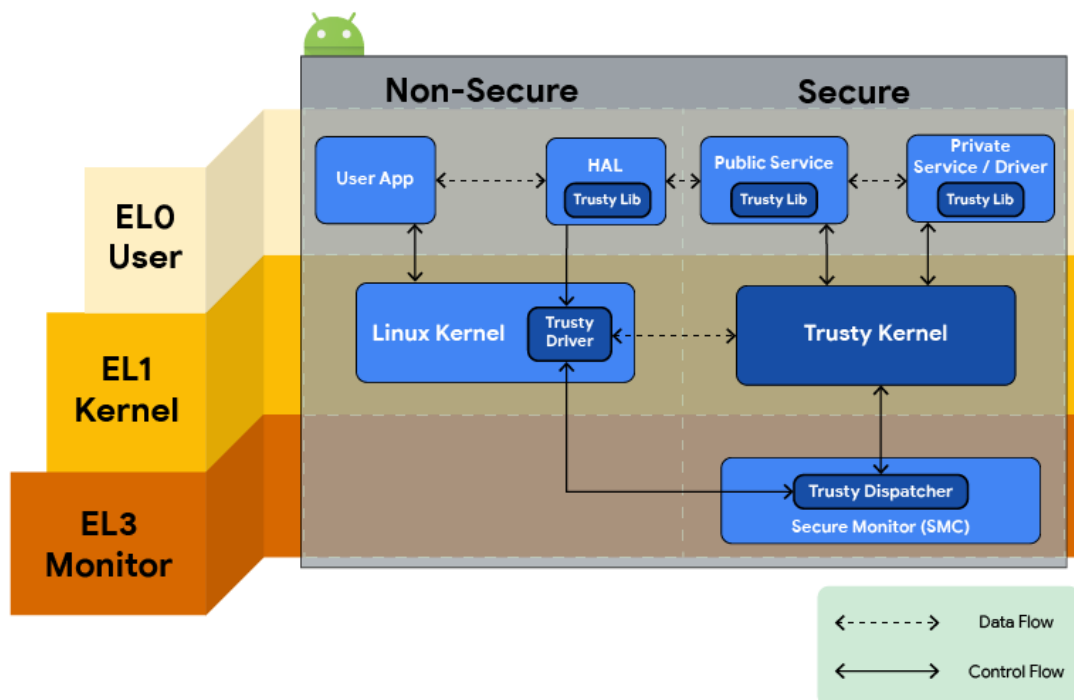


Рисунок 1.3 – Структура роботи Android Keystore

Система Android Keystore дозволяє зберігати криптографічні ключі в контейнері, щоб ускладнити їх вилучення з пристрою. Коли ключі знаходяться в сховищі ключів, ви можете використовувати їх для криптографічних операцій, при цьому ключовий матеріал залишається неекспортованим. Крім того, система сховища ключів дозволяє обмежувати, коли та як можна використовувати ключі, наприклад вимагати автентифікації користувача для використання ключа або обмежувати використання ключів лише в певних криптографічних режимах

Система сховища ключів використовується KeyChain API, доданим в Android 4.0 (API, рівень 14), а також функцією Android Keystore provider, представленою в Android 4.3 (API, рівень 18)

Система Android Keystore захищає ключові матеріали від несанкціонованого використання двома способами. По-перше, це зменшує ризик несанкціонованого використання матеріалу ключа ззовні *пристрою* Android, запобігаючи вилучення матеріалу ключа з процесів програми та з пристрою Android загалом. По-друге, система сховища ключів знижує ризик несанкціонованого використання матеріалу ключа *на* пристрої Android, змушуючи додатки вказувати авторизоване використання своїх ключів, а потім накладаючи ці обмеження поза процесами додатків.

Ключовий матеріал ключів Android Keystore захищено від вилучення за допомогою двох заходів безпеки:

- Ключовий матеріал ніколи не потрапляє в процес подання заявки. Коли програма виконує криптографічні операції за допомогою ключа Android Keystore, відкритий текст, зашифрований текст і повідомлення, які потрібно підписати або перевірити, передаються в системний процес, який виконує криптографічні операції. Якщо процес програми скомпрометовано, зломисник може використовувати ключі програми, але не зможе витягнути їхній ключ (наприклад, для використання поза пристроєм Android).
- Ключовий матеріал може бути прив'язаний до захищеного апаратного забезпечення пристрою Android, наприклад Trusted Execution Environment

(TEE) або Secure Element (SE). Якщо цю функцію ввімкнено для ключа, його матеріал ключа ніколи не буде відкритим за межами безпечного обладнання. Якщо ОС Android зламано або зловмисник може прочитати внутрішню пам'ять пристрою, зловмисник може використовувати ключі Android Keystore будь-якої програми на пристрої Android, але не зможе витягти їх із пристрою. Ця функція вмикається, лише якщо захищене апаратне забезпечення пристрою підтримує певну комбінацію алгоритму ключа, режимів блокування, схем заповнення та дайджестів, з якими авторизовано використовувати ключ.

Щоб перевірити, чи ввімкнено функцію для ключа, треба отримати KeyInfo для ключа.

- Якщо програма націлена на Android 10 (рівень API 29) або вище, варто перевірити значення, що повертається `getSecurityLevel()`. `STRONGBOX` вказують на те, що ключ знаходиться в захищеному обладнанні.

Підтримувані пристрої з ОС Android 9 (рівень API 28) або новішої версії можуть мати Strong Box Keymaster, реалізацію Keymaster або Keymint HAL, яка міститься в захищеному елементі, подібному до апаратного модуля безпеки. У той час як апаратні модулі безпеки можуть стосуватися багатьох різних реалізацій зберігання ключів, де компрометація ядра Linux не може виявити їх, наприклад TEE, StrongBox явно стосується таких пристроїв, як вбудовані елементи безпеки (eSE) або захищені процесорні блоки на SoC (iSE).

Модуль містить наступне:

- Власний процесор
- Безпечне зберігання
- Справжній генератор випадкових чисел
- Додаткові механізми для захисту від підробки пакетів і несанкціонованого стороннього завантаження програм
- Безпечний таймер

- PIN-код сповіщення про перезавантаження (або еквівалент), як-от введення/виведення загального призначення (GPIO)

Для підтримки реалізацій StrongBox з низьким енергоспоживанням підтримується підмножина алгоритмів і розмірів ключів:

- RSA 2048
- AES 128 і 256
- ECDSA, ECDH P-256
- HMAC-SHA256 (підтримує розміри ключів від 8 до 64 байтів включно)
- Потрійний DES
- APDU збільшеної довжини
- Атестація ключа
- Підтримка поправки N для оновлення

Під час генерації або імпорту ключів за допомогою KeyStore класу ви вказуєте перевагу для зберігання ключа в StrongBox Keymaster, встановлюючи true до setIsStrongBoxBacked() методу.

1.1.3 Network Security Configuration:

Для захисту комунікацій між мобільним застосунком і сервером можна використовувати мережеві конфігурації для вимагання безпечного з'єднання через HTTPS.

o/u/a/network_security_config.xml

```
1 <?xml version="1.0"?>
2 <network-security-config>
3     <base-config>
4         <trust-anchors>
5             <!-- Trust preinstalled CAs -->
6             <certificates src="system"/>
7             <!-- Additionally trust user added CAs -->
8             <certificates src="user"/>
9         </trust-anchors>
10    </base-config>
11 </network-security-config>
```

Рисунок 1.4 – Демонстрація налаштування файлу network security

Функція конфігурації мережевої безпеки дає змогу налаштувати параметри мережевої безпеки програми в безпечному декларативному файлі конфігурації, не змінюючи код програми. Ці параметри можна налаштувати для певних доменів і для певної програми. Основні можливості цієї функції:

- **Спеціальні прив'язки довіри:** налаштуйте, яким центрам сертифікації (СА) можна довіряти для безпечних з'єднань програми. Наприклад, довіряти певним самопідписаним сертифікатам або обмежувати набір загальнодоступних ЦС, яким програма довіряє.
- **Перевизначення лише для налагодження:** безпечно налагоджуйте безпечні з'єднання в програмі без додаткового ризику для встановленої бази.
- **Відмова від трафіку відкритого тексту:** захистіть програми від випадкового використання трафіку відкритого тексту (незашифрованого).
- **Закріплення сертифіката:** обмежте безпечне з'єднання програми певними сертифікатами.

Можна зробити, щоб ваша програма довіряла спеціальному набору ЦС замість стандартної платформи. Найпоширеніші причини цього:

- Підключення до хосту за допомогою спеціального ЦС, наприклад, самопідписаного ЦС або виданого всередині компанії.
- Обмеження набору ЦС лише тими ЦС, яким ви довіряєте, замість кожного попередньо встановленого ЦС.
- Довіра додаткових ЦС, не включених до системи.

За замовчуванням безпечні з'єднання (з використанням таких протоколів, як TLS і HTTPS) від усіх додатків довіряють попередньо встановленим системним центрам сертифікації, а програми, націлені на Android 6.0 (рівень API 23) і нижчі, також за умовчанням довіряють доданому користувачем ЦС. Можна налаштувати підключення програми за допомогою `base-config` (для налаштування всієї програми) або `domain-config` (для налаштування для кожного домену).

```

1
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <domain includeSubdomains="true">example.com</domain>
    <trust-anchors>
      <certificates src="@raw/my_ca"/>
    </trust-anchors>
  </domain-config>
</network-security-config>

```

Рисунок 1.4.1 – Демонстрація налаштування всієї програми

Якщо ви не хочете, щоб ваша програма довіряла всім ЦС, яким довіряє система, замість цього можна вказати зменшений набір ЦС, яким довіряють. Це захищає програму від шахрайських сертифікатів, виданих будь-яким іншим ЦС.

Конфігурація для обмеження набору довірених ЦС подібна до довіри спеціальному ЦС для певного домену, за винятком того, що в ресурсі надається кілька ЦС. Наведений нижче уривок коду демонструє, як обмежити набір довірених центрів сертифікації програми

```

<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <domain includeSubdomains="true">secure.example.com</domain>
    <domain includeSubdomains="true">cdn.example.com</domain>
    <trust-anchors>
      <certificates src="@raw/trusted_roots"/>
    </trust-anchors>
  </domain-config>
</network-security-config>

```

Рисунок 1.4.2 – Демонстрація налаштування кожного домену

В цьому підрозділі були проаналізовані методи шифрування дисків та файлів різними технологіями і наведені їх кращі сторони

1.2 Біометричні порівняння

1.2.1 Face Recognition

Аутентифікація обличчям дозволяє користувачам розблокувати свій пристрій, просто подивившись на передню панель пристрою. Android 10 додає підтримку для нового стеку автентифікації обличчя, який може безпечно обробляти кадри камери, зберігаючи безпеку та конфіденційність під час автентифікації обличчя на підтримуваному обладнанні. Android 10 також забезпечує простий спосіб впровадження сумісних із безпекою реалізацій, щоб увімкнути інтеграцію додатків для транзакцій, таких як онлайн-банкінг або інші служби.

Стек автентифікації обличчя Android — це нова реалізація в Android 10. Нова реалізація представляє інтерфейси `IBiometricsFace.hal`

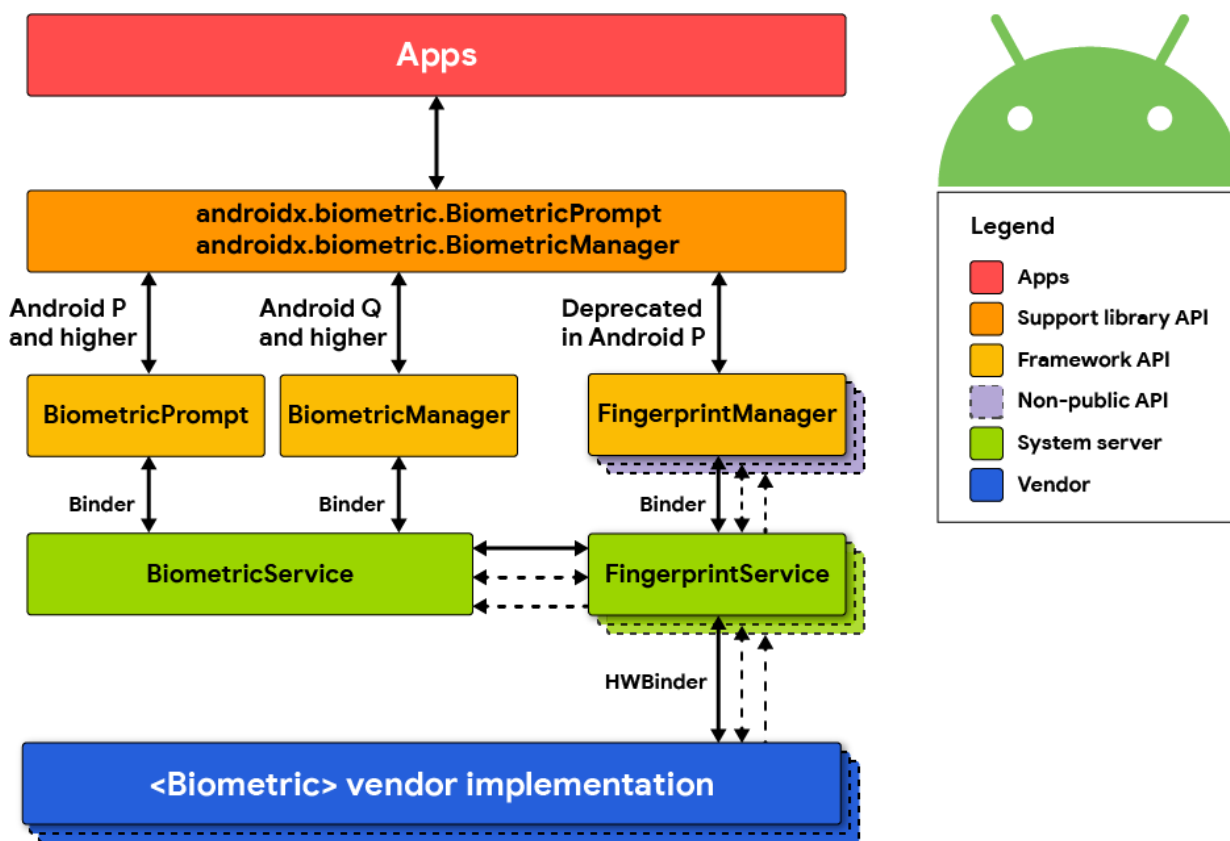


Рисунок 1.5 – Структура роботи Face Recognition в Android

Щоб пристрої відповідали суворим біометричним вимогам для Android 10, вони повинні мати безпечне обладнання для забезпечення цілісності даних про обличчя та остаточного порівняння автентифікації. Документ із визначенням сумісності Android (CDD) визначає необхідний рівень безпеки та необхідний прийнятний коефіцієнт підробки (SAR). Для безпечної обробки та розпізнавання потрібне довірене середовище виконання (TEE). Крім того, потрібне захищене апаратне забезпечення камери для запобігання ін'єкційним атакам на автентифікацію обличчя. Наприклад, пов'язані сторінки пам'яті для даних зображення можуть бути привілейованими та позначеними як лише для читання, щоб лише апаратне забезпечення камери могло оновлювати їх. В ідеалі жоден процес не повинен мати доступу, крім TEE та обладнання.

Оскільки апаратне забезпечення автентифікації за обличчям значно відрізняється, необхідно розробити драйвери для апаратного забезпечення, щоб увімкнути автентифікацію за обличчям, залежно від конкретної архітектури

пристрою. Таким чином, для faced очікується, що структура faced буде відповідати наведеній нижче діаграмі стану.

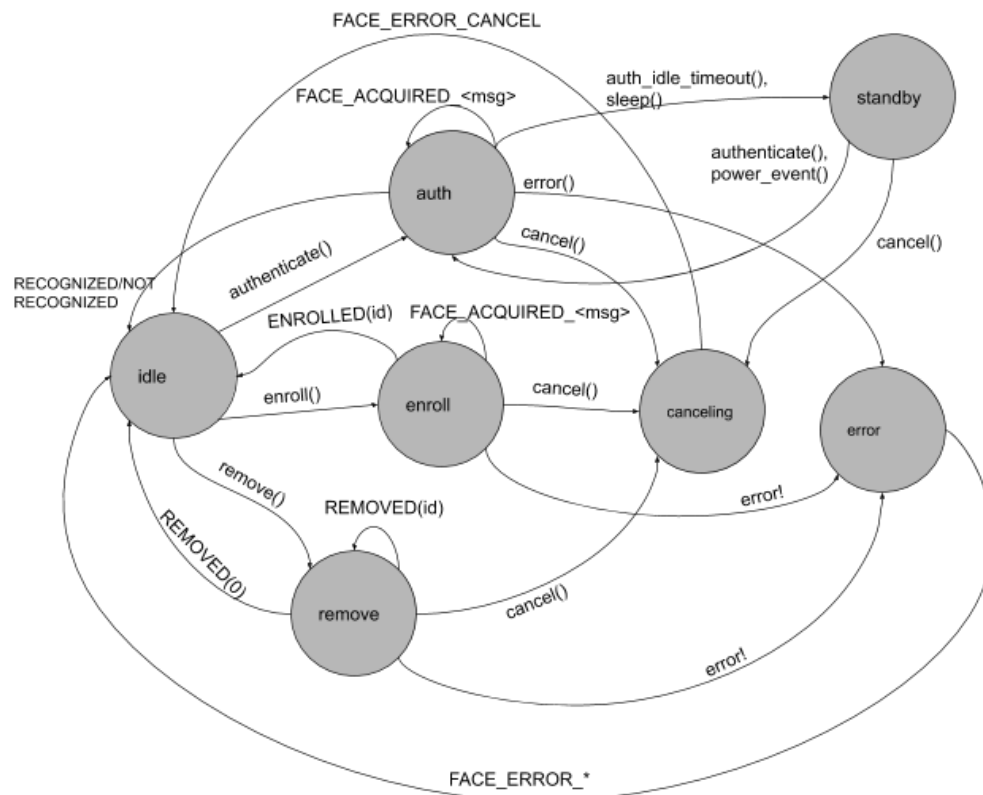


Рисунок 1.6 – Діаграма стану Face Recognition в Android

1.2.2 Fingerprint Recognition

На пристроях із датчиком відбитків пальців користувачі можуть зареєструвати один або кілька відбитки пальців і використовувати ці відбитки пальців, щоб розблокувати пристрій і виконувати інші дії завдання. Android використовує Fingerprint Hardware Interface Definition Language (HIDL) для підключення до спеціальної бібліотеки постачальника та обладнання відбитків пальців (для наприклад, датчик відбитків пальців).

Одним із методів захисту конфіденційної інформації або преміум-контенту у вашій програмі є запит на біометричну автентифікацію, наприклад за допомогою розпізнавання обличчя або відбитка пальця.

Щоб почати використовувати автентифікатор, користувач повинен створити PIN-код, шаблон або пароль. Якщо у користувача його ще немає, процес біометричної реєстрації запропонує створити його.

Щоб визначити типи біометричної автентифікації, які приймає ваша програма, передайте в метод тип автентифікації або побітову комбінацію типів `setAllowedAuthenticators()`. У наведеному нижче фрагменті коду показано, як підтримувати автентифікацію за допомогою біометричних даних класу 3 або облікових даних блокування екрана.

Якщо ваша програма використовує секретний ключ, для розблокування якого потрібні біометричні облікові дані, користувач повинен автентифікувати свої біометричні облікові дані кожного разу, перш ніж ваша програма отримає доступ до ключа.

Щоб зашифрувати конфіденційну інформацію лише після автентифікації користувача за допомогою біометричних облікових даних, виконайте такі дії:

1. Згенеруйте ключ, який використовує таку `KeyGenParameterSpec` конфігурацію:

```
generateSecretKey(KeyGenParameterSpec.Builder(  
    KEY_NAME,  
    KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT)  
    .setBlockModes(KeyProperties.BLOCK_MODE_CBC)  
    .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)  
    .setUserAuthenticationRequired(true)  
    // Invalidate the keys if the user has registered a new biometric  
    // credential, such as a new fingerprint. Can call this method only  
    // on Android 7.0 (API level 24) or higher. The variable  
    // "invalidatedByBiometricEnrollment" is true by default.  
    .setInvalidatedByBiometricEnrollment(true)  
    .build()))
```

Ви можете використовувати секретний ключ, який дозволяє автентифікацію за допомогою біометричних облікових даних або облікових даних для блокування екрана (PIN-код, ключ або пароль). Під час налаштування цього ключа вкажіть період дії. Протягом цього періоду часу ваша програма може виконувати кілька криптографічних операцій без повторної автентифікації користувача.

За замовчуванням система вимагає від користувачів виконати певну дію, наприклад натиснути кнопку, після того, як їхні біометричні облікові дані прийняті. Ця конфігурація є кращою, якщо ваша програма показує діалогове вікно для підтвердження конфіденційної або високоризикованої дії, як-от здійснення покупки.

Однак якщо ваша програма показує діалогове вікно біометричної автентифікації для дії з меншим ризиком, ви можете надати системі підказку про те, що користувачеві не потрібно підтверджувати автентифікацію. Ця підказка може дозволити користувачеві швидше переглядати вміст у вашій програмі після повторної автентифікації за допомогою пасивної модальності, як-от розпізнавання обличчя чи райдужної оболонки ока. Щоб надати цю підказку, перейдіть `false` до `setConfirmationRequired()` методу.

Наведені нижче вказівки HAL щодо відбитків пальців розроблено для того, щоб це гарантувати дані відбитків пальців **не витікають** та **видаляються** коли користувача видалено з пристрою:

- Необроблені дані відбитків пальців або похідні (наприклад, шаблони) ніколи не повинні бути доступним поза драйвером датчика або TEE. Якщо обладнання підтримує TEE, доступ до апаратного забезпечення має бути обмежений TEE та захищений SELinux політики. Канал послідовного периферійного інтерфейсу (SPI) має бути доступний лише для TEE, і для всіх файлів пристрою має бути чітка політика SELinux.
- Отримання, реєстрація та розпізнавання відбитків пальців мають відбуватися всередині TEE.
- У файлі можна зберігати лише зашифровану форму даних відбитків пальців системи, навіть якщо сама файлова система зашифрована.
- Шаблони відбитків пальців мають бути підписані приватним ключем для конкретного пристрою. Для розширеного стандарту шифрування (AES)

потрібно принаймні підписати шаблон з абсолютним шляхом до файлової системи, групою та ідентифікатором пальця, таким як шаблон файли не працюють на іншому пристрої або для будь-кого, крім користувача зареєстрував їх на одному пристрої. Наприклад, копіювання даних відбитків пальців із а інший користувач на тому самому пристрої або з іншого пристрою не повинен працювати.

- Реалізації повинні або використовувати шлях до файлової системи, наданий у `setActiveGroup()` функціонують або забезпечують спосіб видалення всіх користувачів дані шаблону, коли користувача буде видалено. Це настійно рекомендується файли шаблонів відбитків пальців зберігатимуться в зашифрованому вигляді та зберігатимуться за вказаним шляхом. Якщо це неможливо через вимоги до зберігання TEE, розробник повинен додати гаки забезпечити видалення даних після видалення користувача.

Датчик відбитків пальців пристрою зазвичай неактивний. Однак у відповідь на виклик `authenticate` або `enroll`, датчик відбитків пальців розпізнає дотик (екран також може прокинутися, коли користувач торкається датчика відбитків пальців). Високорівневий потік зіставлення відбитків пальців містить такі кроки:

1. Користувач прикладає палець до датчика відбитків пальців.
2. Спеціальна бібліотека постачальника визначає, чи є збіг відбитків пальців поточний набір зареєстрованих шаблонів відбитків пальців.
3. Відповідні результати передаються до `FingerprintService`.

Архітектура

Fingerprint HAL взаємодіє з такими компонентами.

- `BiometricManager` взаємодіє безпосередньо з програмою в процесі програми. Кожен додаток має екземпляр `IBiometricsFingerprint.hal`

- FingerprintService діє в системний процес, який обробляє зв'язок із відбитком пальця HAL.
- **Fingerprint HAL** — це реалізація C/C++ Інтерфейс IBiometricsFingerprint HIDL. Він містить бібліотеку постачальника який взаємодіє з апаратним забезпеченням пристрою.
- **Компоненти API сховища ключів і Keymaster** надають криптографія з апаратною підтримкою для безпечного зберігання ключів у безпечному середовищі, як-от довірене середовище виконання (TEE).

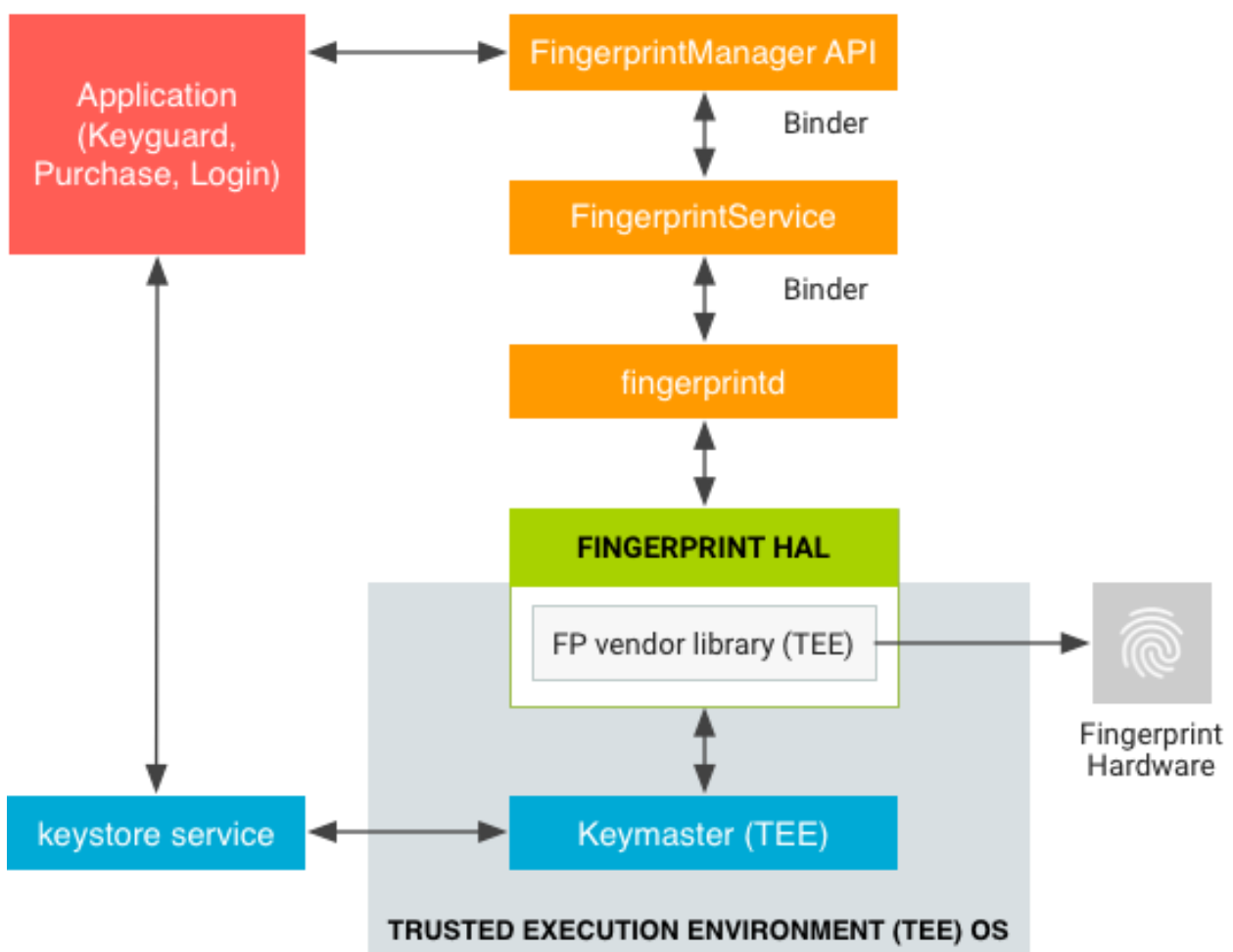


Рисунок 1.7 - Високорівневий потік даних для відбитків пальців автентифікація

Реалізація HAL від постачальника має використовувати протокол зв'язку вимагається TEE. Необроблені зображення та оброблені відбитки пальців не повинні передаватися в ненадійну пам'ять. Усі такі біометричні дані потрібно

зберігати у безпечному обладнанні, такому як TEE. Укорінення **не можна** мати можливість скомпрометувати біометричні дані.

FingerprintService та fingerprintd здійснюють дзвінки через HAL відбитків пальців до бібліотеку постачальника для реєстрації відбитків пальців та виконання інших операцій.

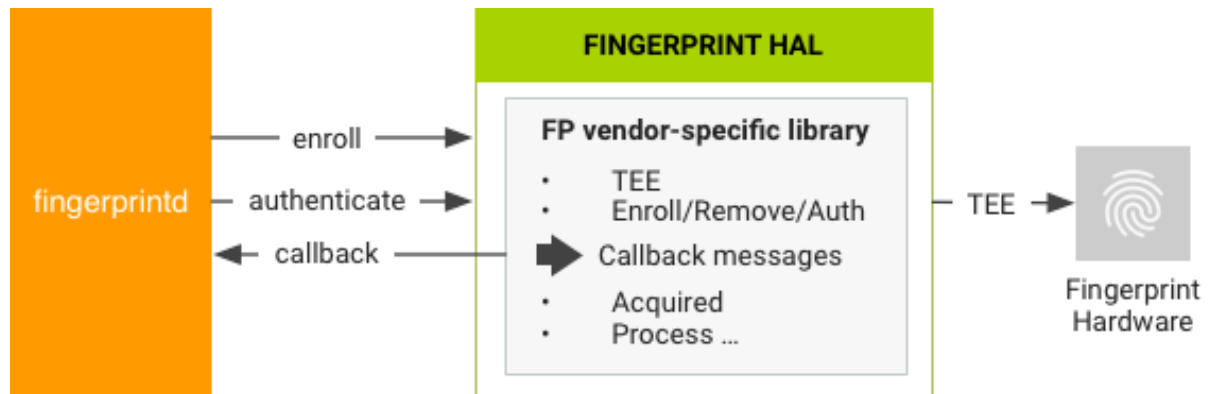


Рисунок 1.8 – Взаємодія відбитків пальців із спеціальною бібліотекою постачальника відбитків пальців

1.2.3 Voice Recognition

API служби голосової взаємодії забезпечує абстракцію щодо різних потенційних програм голосового керування. Реалізації можна розробляти відповідно до вказівок.

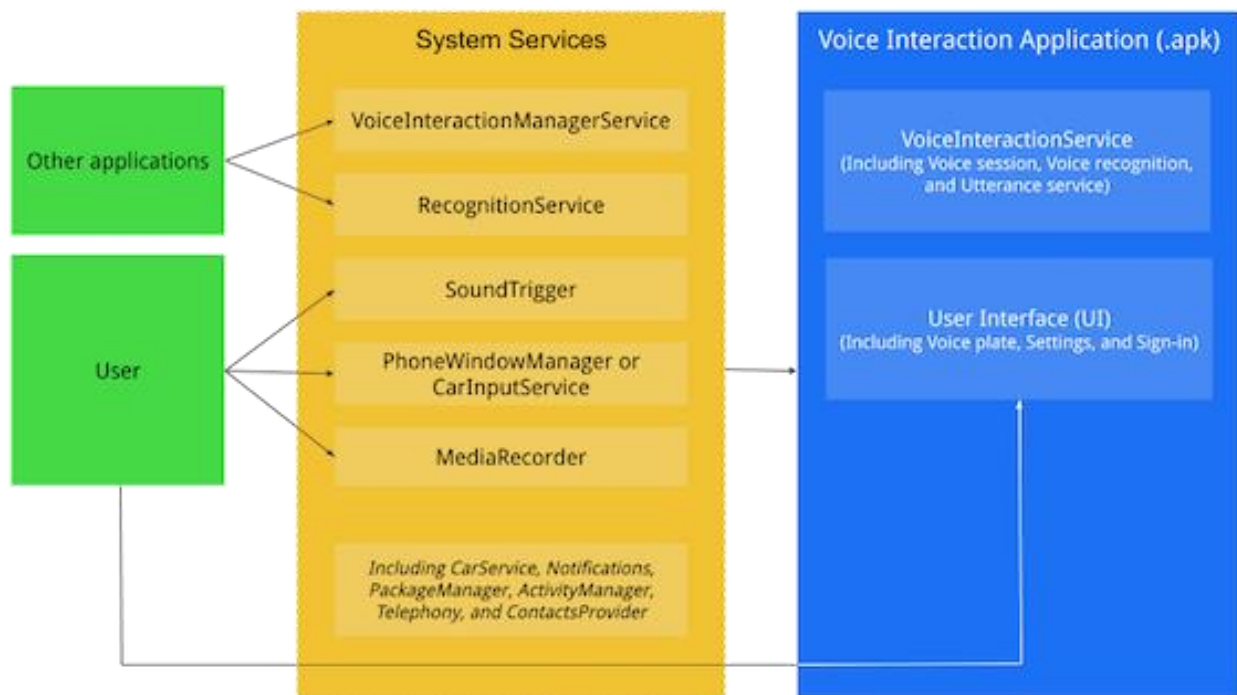


Рисунок 1.9 – API Voice Recognition

Використовуючи екземпляр `SpeechRecognizer`, ми зможемо прослуховувати команди користувача та виконувати дії всередині нашої програми. Додайте глобальний екземпляр `SpeechRecognizer` під назвою `mSpeechRecognizer` до `MainActivity` та додайте функцію для його ініціалізації. Ми також додамо `RecognitionListener` до `mSpeechRecognizer`, що дозволить нам отримувати сповіщення на кожному етапі аудіосеансу. Два зворотних виклики, які нас цікавлять, це `onPartialResults()` та `onResults()`. Ми заповнимо ці зворотні виклики пізніше. Ви можете ігнорувати інші зворотні виклики — вони нам не знадобляться для цього експерименту.

```

class MainActivity : AppCompatActivity() {
    private var mSpeechRecognizer: SpeechRecognizer? = null
    private var mIsListening = false // this will be needed later
    private var mUserInfoText: TextView? = null
    private var mUserUtteranceOutput: TextView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        // ...
        mUserUtteranceOutput = findViewById(R.id.user_utterance_output)
        mUserInfoText = findViewById(R.id.user_info_text)

        createSpeechRecognizer()
        // ...
    }

    private fun createSpeechRecognizer() {
        mSpeechRecognizer = SpeechRecognizer.createSpeechRecognizer(this)
    }
}

```

```

mSpeechRecognizer?.setRecognitionListener(object : RecognitionListener {
    override fun onReadyForSpeech(params: Bundle) {}
    override fun onBeginningOfSpeech() {}
    override fun onRmsChanged(rmsdB: Float) {}
    override fun onBufferReceived(buffer: ByteArray) {}
    override fun onEndOfSpeech() {}

    override fun onError(error: Int) {}

    override fun onResults(results: Bundle) {
        // Called when recognition results are ready. This callback will
        be called when the
        // audio session has been completed and user utterance has been
        parsed.
    }

    override fun onPartialResults(partialResults: Bundle) {
        // Called when partial recognition results are available, this
        callback will be
        // called each time a partial text result is ready while the
        user is speaking.
    }

    override fun onEvent(eventType: Int, params: Bundle) {}
})
}

```

Перед початком аудіосеансу нам також потрібно буде визначити намір розпізнавання аудіо, який буде передано до *mSpeechRecognizer*

```

private fun createIntent(): Intent {
    val i = Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
    i.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
    i.putExtra(RecognizerIntent.EXTRA_PARTIAL_RESULTS, true)
    i.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "en-IN")
    return i
}

```

EXTRA_LANGUAGE_MODEL є обов'язковим додатком для ACTION_RECOGNIZE_SPEECH. Ми також встановлюємо для параметра EXTRA_PARTIAL_RESULTS значення true, щоб отримувати сповіщення про часткові результати мовлення, коли користувач починає говорити через onPartialResults зворотний виклик.

EXTRA_LANGUAGE є необов'язковим і встановлює мову для розпізнавання мовлення. Крім того, є кілька інших додаткових функцій, які ви можете перевірити тут.

1.2.4 Iris Recognition

Розпізнавання ірисів (Iris Recognition) - це біометричний метод ідентифікації особи, який використовує унікальні характеристики структури очей. Ірис - це

кільцева частина очного яблука, і він має складну та унікальну текстуру, яка може слугувати як унікальна "біометрична карта" для ідентифікації особи.

Основні характеристики ірисів, які використовуються для розпізнавання:

1. Унікальність структури:

- Кожен ірис має свою власну унікальну структуру, що визначається розташуванням вен і пігментів.

2. Текстурні особливості:

- Різноманітність та складність текстури ірисів, така як відстань між венами та їхнім розташуванням, використовуються для унікального визначення особливостей кожного ірису.

3. Висока стабільність

- Іриси залишаються стабільними протягом життя і мають невелику вразливість до впливу факторів зовнішнього середовища.

Процес розпізнавання ірисів зазвичай включає такі етапи:

1. Збір даних:

- Використання спеціалізованого обладнання (ірис-сканера або камери високої роздільної здатності), щоб отримати зображення ірисів з очей користувача.

2. Створення шаблону:

- Обробка отриманих зображень для виділення унікальних особливостей ірисів та створення біометричного шаблону.

3. Збереження шаблону:

- Зберігання біометричного шаблону в безпечному місці або його перетворення у хеш-код для забезпечення конфіденційності.

4. Порівняння шаблону:

- Під час аутентифікації або ідентифікації, зібраний шаблон порівнюється з збереженим еталоном для визначення ідентичності особи.

Розпізнавання ірисів використовується в системах безпеки, контролю доступу та аутентифікації, де високий рівень точності та надійності є критичними.

В цьому розділі були проаналізовані API методи для авторизації юзера за біометричними даними

1.3 OAuth2 services

OAuth 2 являє собою фреймворк для авторизації, що дозволяє програмам здійснювати обмежений доступ до облікових записів користувачів на HTTP сервісах, наприклад, на Facebook, GitHub і DigitalOcean. Він працює за принципом делегування аутентифікації користувача сервісу, на якому знаходиться обліковий запис користувача, дозволяючи сторонньому додатку отримувати доступ до облікового запису користувача. OAuth 2 працює в Інтернеті, на десктопних та мобільних додатках

Абстрактное описание протокола

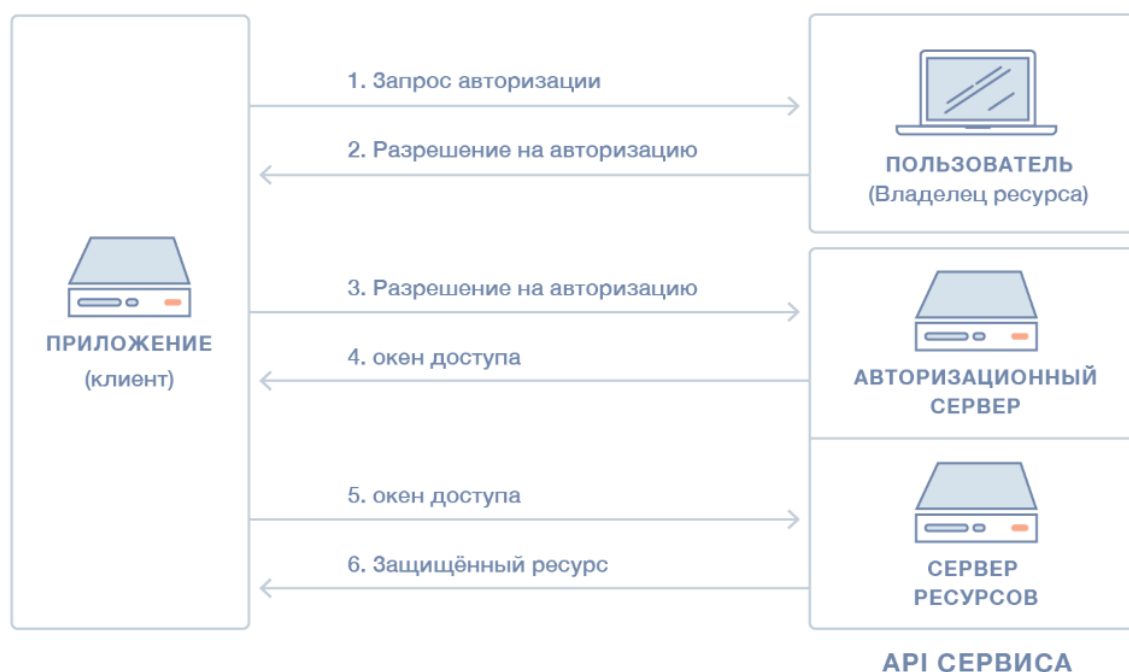


Рисунок 1.10 – Послідовність роботи протокола

Розглянемо опис послідовності кроків на цій діаграмі:

1. Додаток запитує у користувача авторизацію на доступ до сервера ресурсів.
2. Якщо користувач авторизує запит, додаток отримує дозвіл на авторизацію (authorization grant).

3. Додаток запитує авторизаційний токен у сервера авторизації (API) шляхом надання інформації про себе і дозвіл на авторизацію від користувача.
4. Якщо автентифікація програми підтверджена і авторизація дійсною, сервер авторизації (API) створює токен доступу для програми. Процес авторизації завершено.
5. Додаток запитує ресурс у сервера ресурсів (API), надаючи при цьому токен доступу для аутентифікації.
6. Якщо токен дійсний, сервер ресурсів (API) надає запитуваний ресурс додатку^{a i=4}.

Фактичний порядок кроків описаного процесу може відрізнятися в залежності від типу дозволу на авторизацію, але в цілому процес буде виглядати описаним чином. Далі ми розглянемо різні типи дозволів на авторизацію.

У абстрактному описі протоколу вище перші чотири кроки стосуються питань створення дозволу на авторизацію та токена доступу. Тип дозволу на авторизацію залежить від методу запиту авторизації, що використовується програмою, а також від того, які типи дозволу підтримуються з боку API. OAuth 2 визначає чотири різні типи, кожен з яких корисний у певних ситуаціях:

- **Код авторизації (Authorization Code):** використовується з серверними програмами (server-side applications).
- **Неявний (Implicit):** використовується мобільними або веб-додатками (програми, що працюють на пристрої користувача).
- **Облікові дані власника ресурсу (Resource Owner Password Credentials):** використовуються довіреними додатками, наприклад, додатками, які є частиною самого сервісу.
- **Облікові дані клієнта (Client Credentials):** використовуються при доступі програми до API.

Далі ми розглянемо ці типи дозволу авторизацію, приклади їх використання.

Код авторизації є одним з найпоширеніших типів дозволу на авторизацію, оскільки він добре підходить для серверних додатків, де вихідний код програми та секрет клієнта не доступні стороннім. Процес в даному випадку будується на

перенаправленні (redirection), що означає, що додаток повинен бути в змозі взаємодіяти з користувачем агентом (user-agent), наприклад, веб-браузером, і отримувати коди авторизації API, що перенаправляються через користувацький агент.

Описание процесса авторизации

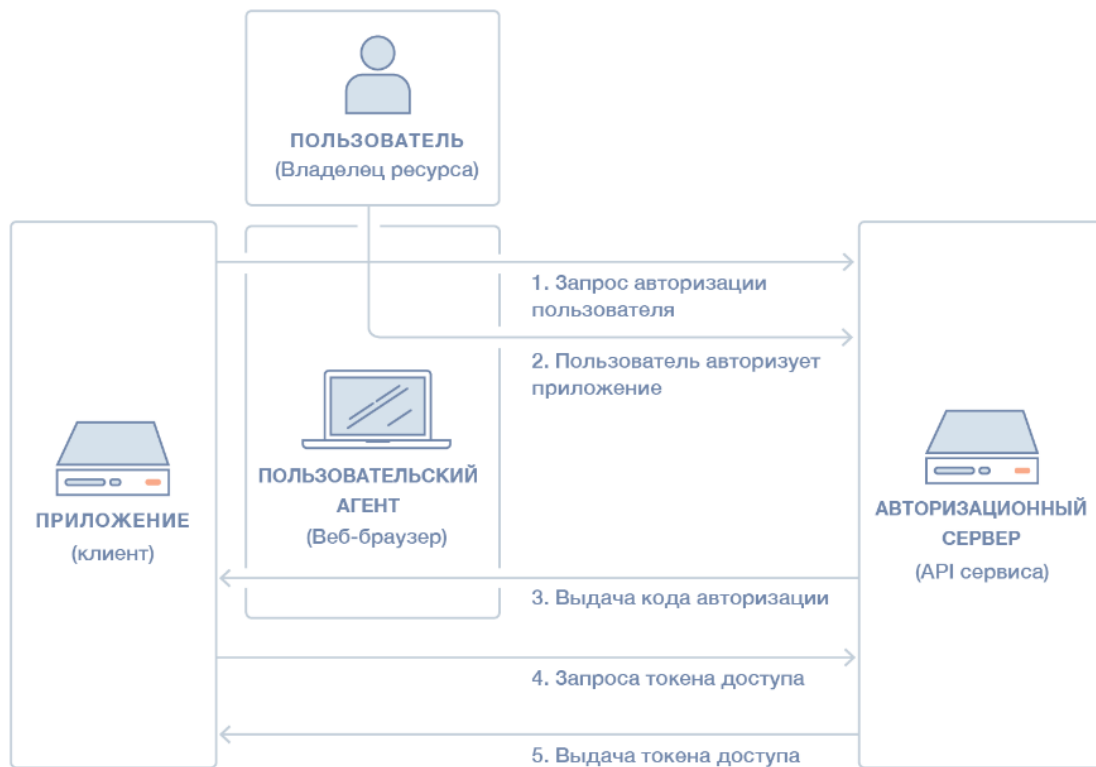


Рисунок 1.11 – Послідовність виконання процесу авторизації

Спочатку користувачеві надається посилання на наступний вид:

```
https://cloud.digitalocean.com/v1/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read
```

Розглянемо компоненти посилання:

- ****<https://cloud.digitalocean.com/v1/oauth/authorize>****: вхідна точка API авторизації (API authorization endpoint).
- **client_id=****CLIENT_ID**: ідентифікатор клієнта програми (за допомогою цього ідентифікатора API розуміє, яка програма запитує доступ).
- **redirect_uri=****CALLBACK_URL**: URL, на який сервіс перенаправить агент користувача (браузер) після видачі авторизаційного коду.

- **response_type=code**: вказує на те, що програма запитує доступ за допомогою коду авторизації.
- **scope=read**: задає рівень доступу програми (в даному випадку – доступ на читання).

Коли користувач натискає на посилання, він повинен спочатку здійснити вхід до системи для підтвердження своєї особистості (якщо він, звичайно, ще не залогінений). Після цього сервіс запропонує користувачеві авторизувати або відмовити в авторизації програмі для доступу до облікового запису користувача. Приклад такого діалогу наведено нижче:

Authorize Application

Thedropletbook App would like permission to access your account: **manicas@digitalocean.com**

Review Permissions

• Read

Authorize Application

Deny

Thedropletbook App

Drop the "the".

[Visit application website](#)

Рисунок 1.12 – Керування дозволами

Якщо користувач вибирає “Авторизувати програму”, сервіс перенаправляє користувальницький агент (браузер) по URL перенаправлення (redirect URL), який був заданий на етапі реєстрації клієнта. Посилання буде виглядати схожим чином (в даному прикладі програма називається “

https://dropletbook.com/callback?code=AUTHORIZATION_CODE

Додаток запитує токен доступу у API шляхом відправки авторизаційного коду та автентифікаційної інформації (включаючи *секрет клієнта*) сервісу. Нижче наведено приклад POST-запиту для отримання токена DigitalOcean:

https://cloud.digitalocean.com/v1/oauth/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=authorization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL

Якщо авторизація пройшла успішно, API повертає токен доступу (а також опціонально, токен для оновлення токена доступу - refresh token). Уся відповідь сервера може виглядати так:

```
{"access_token":"ACCESS_TOKEN","token_type":"bearer","expires_in":2592000,"refresh_token":"REFRESH_TOKEN","scope":"read","uid":100101,"info":{"name":"Mark E. Mark","email":"mark@thefunkybunch.com"}}
```

Тепер програма авторизована! Воно може використовувати токен для доступу до облікового запису користувача через API сервісу із заданими обмеженнями доступу до тих пір, поки не закінчиться термін дії токена або токен не буде відкликаний. Якщо було створено токен для оновлення токена доступу, він може бути використаний для отримання нових токенів доступу, коли закінчиться термін дії старого токена.

В цьому підрозділі, було розглянути OAuth2 протокол для безпечної авторизації юзера

1.4 Висновок до першого розділу

В цьому розділі були проаналізовані методи шифрування дисків та файлів різними технологіями і наведені їх кращі сторони, API методи для авторизації юзера за біометричними даними, розглянуто OAuth2 протокол для безпечної авторизації юзера

РОЗДІЛ 2 РОЗРОБКА СИСТЕМ ЗАХИСТУ ДЛЯ МОБІЛЬНОГО ДОДАТКУ

2.1 Інтеграція безпечної взаємодії з сервером на Android

Для приклада для безпечної взаємодії з сервером і передачі даних ми можемо згенерувати RSA ключі і підписувати повідомлення і розшифровувати їх. Всі дані будуть зберігатися в Keystore, тому доступ до них заборонений

```
companion object {  
  
    // settings crypto  
    private const val ALGORITHM = KeyProperties.KEY_ALGORITHM_RSA  
    private const val BLOCK_MODE = "None"  
    private const val PADDING = KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1  
    private const val TRANSFORMATION = "$ALGORITHM/$BLOCK_MODE/$PADDING"  
    const val KEY_ALIAS = "secret"  
  
    private const val ANDROID_KEYSTORE = "AndroidKeyStore"  
    private const val kPrivateKey = "privateKey"  
    private val recoveryPass = "recoveryPassword".toCharArray()  
  
    val keyStore: KeyStore = KeyStore.getInstance(ANDROID_KEYSTORE).apply {  
        load(null)  
    }  
  
    private val encryptCipher  
        get() = Cipher.getInstance(TRANSFORMATION).apply {  
            val publicKey = keyStore.getCertificate(KEY_ALIAS).publicKey  
            init(Cipher.ENCRYPT_MODE, publicKey)  
        }  
  
    private val decryptCipher  
        get() = Cipher.getInstance(TRANSFORMATION).apply {  
            val entry = keyStore.getEntry(KEY_ALIAS, null)  
            val privateKey = (entry as KeyStore.PrivateKeyEntry).privateKey  
            init(Cipher.DECRYPT_MODE, privateKey)  
        }  
  
    @Throws(Exception::class)  
    fun decryptMessage(encryptedText: String?): String {  
        return String(  
            decryptCipher.doFinal(  
                android.util.Base64.decode(  
                    encryptedText,  
                    android.util.Base64.DEFAULT  
                )  
            )  
        )  
    }  
  
    private fun createKey(): KeyPair? {  
        // if key is create, return func  
  
        if(PrefUtil.getIsKeyPair()) {
```

```

        Timber.d("SecurityUtils - success load")
        return null
    }

    Timber.d("SecurityUtils - createKey - creating")

    val kpg: KeyPairGenerator =
        KeyPairGenerator.getInstance(KeyProperties.KEY_ALGORITHM_RSA, ANDROID_KEYSTORE)
    val parameterSpec: KeyGenParameterSpec = KeyGenParameterSpec.Builder(
        KEY_ALIAS,
        KeyProperties.PURPOSE_SIGN
            or KeyProperties.PURPOSE_DECRYPT
            or KeyProperties.PURPOSE_ENCRYPT
            or KeyProperties.PURPOSE_VERIFY
    ).run {
        setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1)
        setRandomizedEncryptionRequired(true)
        build()
    }

    kpg.initialize(parameterSpec)

    PrefUtil.put(isKeyPair, true)

    return kpg.generateKeyPair()
}

fun initial() {
    createKey()
}
}

```

Для ходіння на сервер використовується бібліотека Retrofit, тому можна згенерувати сертифікат і удосконалити наш код, додавши схожу логіку, але яка автоматично буде підписувати будь який запит на сервер

```

private fun generateSecureOkHttpClient(context: Context): OkHttpClient {
    var httpClientBuilder = OkHttpClient.Builder()
        .readTimeout(60, TimeUnit.SECONDS)
        .connectTimeout(60, TimeUnit.SECONDS)

    // Get the file of our certificate
    var caFileInputStream = context.resources.openRawResource(R.raw.my_certificate)

    // We're going to put our certificates in a Keystore
    val keyStore = KeyStore.getInstance("PKCS12")
    keyStore.load(caFileInputStream, "my file password".toCharArray())

    // Create a KeyManagerFactory with our specific algorithm our our public keys
    // Most of the cases is gonna be "X509"
    val keyManagerFactory = KeyManagerFactory.getInstance("X509")
    keyManagerFactory.init(keyStore, "my file password".toCharArray())
}

```

```

        // Create a SSL context with the key managers of the KeyManagerFactory
        val sslContext = SSLContext.getInstance("TLS")
        sslContext.init(keyManagerFactory.keyManagers, null, SecureRandom())

        //Finally set the sslSocketFactory to our builder and build it
        return httpClientBuilder
            .sslSocketFactory(sslContext.socketFactory)
            .build()
    }

    fun generateService(context: Context): Retrofit {
        return Retrofit.Builder()
            .baseUrl("https://development.myapi.com/")
            .client(generateSecureOkHttpClient(context))
            .build()
    }
}

```

Таким чином, для безпечної взаємодія клієнта і сервера потрібно користуватися можливостями підпису сертифікатів та шифрування усіх повідомлень

2.2 Зашифрована база даних

Цей код визначає статичний метод **provideAppDatabase**, який надає шифровану базу даних SQLite використовуючи бібліотеку Room в Android-додатку. Використовується SupportFactory для налаштування шифрування з використанням паролю, щоб забезпечити додатковий рівень безпеки для збережених даних.

```

companion object {
    @Provides
    @Singleton
    fun provideAppDatabase(@ApplicationContext appContext: Context): AppDatabase {
        final byte[] passphrase = SQLiteDatabase.getBytes(userEnteredPassphrase);
        final SupportFactory factory = new SupportFactory(passphrase);
        return Room.databaseBuilder(activity, AppDatabase.class, DB_NAME)
            .openHelperFactory(factory)
            .build();
    }
}

```

Дамп різниці в базі даних зашифрованому вигляді і ні:

```
~ sjlombardo$ hexdump -C sqlite.db
00000000 53 51 4c 69 74 65 20 66 6f 72 6d 61 74 20 33 00 |SQLite format 3.|
...
000003c0 65 74 32 74 32 03 43 52 45 41 54 45 20 54 41 42 |et2t2.CREATE TAB|
000003d0 4c 45 20 74 32 28 61 2c 62 29 24 01 06 17 11 11 |LE t2(a,b)$...|
...
000007e0 20 74 68 65 20 73 68 6f 77 15 01 03 01 2f 01 6f | the show.../.o|
000007f0 6e 65 20 66 6f 72 20 74 68 65 20 6d 6f 6e 65 79 |ne for the money|

~ $ sqlite3 sqlcipher.db
sqlite> PRAGMA KEY='test123';
sqlite> CREATE TABLE t1(a,b);
sqlite> INSERT INTO t1(a,b) VALUES ('one for the money', 'two for the show');
sqlite> .quit

~ $ hexdump -C sqlcipher.db
00000000 84 d1 36 18 eb b5 82 90 c4 70 0d ee 43 cb 61 87 |.?6.?..?p.?C?a.|
00000010 91 42 3c cd 55 24 ab c6 c4 1d c6 67 b4 e3 96 bb |.B?..?|
00000bf0 8e 99 ee 28 23 43 ab a4 97 cd 63 42 8a 8e 7c c6 |..?{#C??.?cB..|?|

~ $ sqlite3 sqlcipher.db
sqlite> SELECT * FROM t1;
Error: file is encrypted or is not a database
```

Для пошуку з рівня JNI (C++), деякі класи повинні не бути заплутані і мати звичайну назву

```
-keep,includedescriptorclasses class net.sqlcipher.** { *; }
-keep,includedescriptorclasses interface net.sqlcipher.** { *; }
```

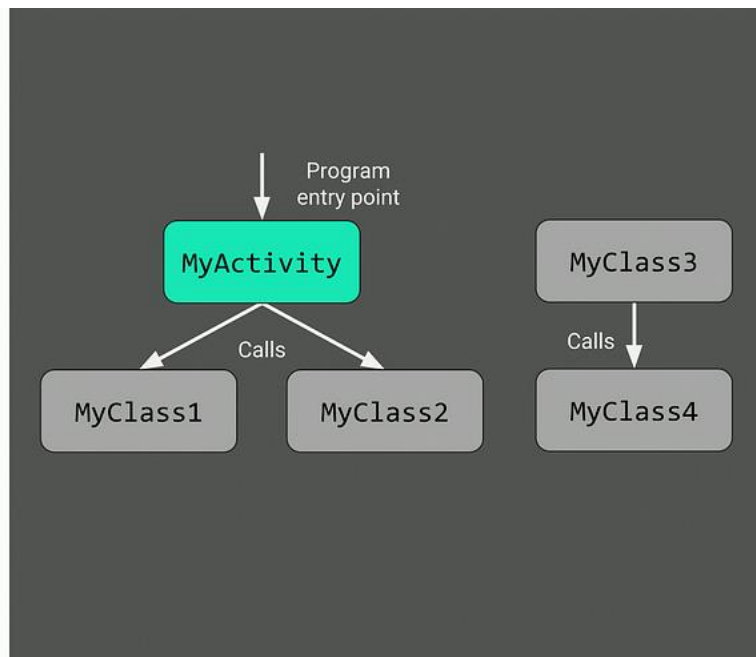
Таким чином, шифрування бд є важливою складовою для безпечного додатку

2.3 Заплутання коду для ускладнення реверсу додатку (Binary Protections)

ProGuard — це інструмент, який зменшує, оптимізує та заплутує код. ProGuard легко доступний як частина процесу складання Android Gradle і постачається разом із SDK.

Dead code elimination

 Live code



```
buildTypes{
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```

```
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
```

Таким чином, бібліотека ProGuard потребує певних знань з роботою з нею, але дає можливість уникнути або ускладнити реверс мобільного додатку

2.4 Захист від SSL Pinning Attack

Створення **CertificatePinner** з правилами перевірки сертифікатів для конкретних хостів та відповідних хешів сертифікатів. Це гарантує, що під час з'єднання з сервером будуть перевірятися лише очікувані сертифікати за їхніми хешами,

додаючи додатковий рівень безпеки до з'єднання та запобігаючи певним видам атак, таким як "Man-in-the-Middle".

```
fun getSafeOkHttpClient(): OkHttpClient {
    val httpClient = OkHttpClient.Builder()
    try {
        val certificatePinner = CertificatePinner.Builder()
            .add("api.example.com", "sha256/encoded_hash_of_certificate")
            .build()
        httpClient.certificatePinner(certificatePinner)
    } catch (e: Exception) {
        e.printStackTrace()
    }
    return httpClient.build()
}
```

Таким чином, додавання `CertificatePinner` покращує взаємодію клієнта і сервера і створює додатковий шар безпеки

2.5 Захист від SSL Attack на девайсах зі старими сервісами

Клас **PathWorker** відповідає за автоматичне оновлення чи встановлення безпечного провайдера на платформі Android через механізм `WorkManager`. Його основна мета - забезпечити, щоб додатки завжди користувалися актуальними і безпечними сервісами Google Play, що важливо для стабільної та безпечної роботи додатків на різних пристроях. Робота класу включає в себе обробку можливих винятків, спричинених станом Google Play Services, та повідомлення системи `WorkManager` про результати виконання завдання. Такий підхід допомагає забезпечити ефективне управління безпекою та сумісністю додатків на платформі Android.

```
class PatchWorker(appContext: Context, workerParams: WorkerParameters):
    Worker(appContext, workerParams) {

    override fun doWork(): Result {
        try {
            ProviderInstaller.installIfNeeded(context)
        } catch (e: GooglePlayServicesRepairableException) {
```

```

        // Indicates that Google Play services is out of date, disabled,
etc.

        // Prompt the user to install/update/enable Google Play services.
        GoogleApiAvailability.getInstance()
            .showErrorNotification(context, e.connectionStatusCode)

        // Notify the WorkManager that a soft error occurred.
        return Result.failure()

    } catch (e: GooglePlayServicesNotAvailableException) {
        // Indicates a non-recoverable error; the ProviderInstaller can't
        // install an up-to-date Provider.

        // Notify the WorkManager that a hard error occurred.
        return Result.failure()
    }

    // If this is reached, you know that the provider was already up to date
    // or was successfully updated.
    return Result.success()
}
}

```

Таким чином, перевірка оновлення GooglePlayServices є однієї із важливих методів захисту для потенційних вразливих місць в безпеці

2.6 Rooting hack

Rooting на Android означає отримання користувачем повних прав на управління пристроєм, включаючи системні файли та операційну систему. Основна проблема, пов'язана із застосунками для отримання root-доступу (різні rooting apps), полягає в безпеці та стабільності системи.

```

/**
 * Checks if the device is rooted.
 *
 * @return `true` if the device is rooted, `false` otherwise.
 */
fun isRooted(): Boolean {

    // get from build info
    val buildTags: String = android.os.Build.TAGS
    if (buildTags != null && buildTags.contains("test-keys")) {
        return true
    }

    // check if /system/app/Superuser.apk is present
    try {
        val file: java.io.File = java.io.File("/system/app/Superuser.apk")
        if (file.exists()) {
            return true
        }
    }
}

```

```

    } catch (e1: java.lang.Exception) {
        // ignore
    }

    // try executing commands
    return (canExecuteCommand("/system/xbin/which su")
        || canExecuteCommand("/system/bin/which su") ||
        canExecuteCommand("which su"))
}

// executes a command on the system
private fun canExecuteCommand(command: String): Boolean {
    val executedSuccessfully: Boolean
    executedSuccessfully = try {
        java.lang.Runtime.getRuntime().exec(command)
        true
    } catch (e: java.lang.Exception) {
        false
    }
    return executedSuccessfully
}

```

Таким чином, перевірка юзера на рутованість, не дає можливості для подальшого аналізу додатку і можливого його взлому.

2.7 Відключення відлагоджувальних можливостей

Відключення відлагоджувальних можливостей у Android важливо для забезпечення безпеки додатка в режимі реальної експлуатації. Це може запобігти неправомірному використанню додатка та зменшити ризик витоку конфіденційної інформації

```

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (BuildConfig.DEBUG) {
            Timber.plant(Timber.DebugTree())
        } else {
            Timber.uprootAll()
        }
    }
}

```

Таким чином, відключення від журналу логування дає можливість приховати важливі критичні моменти в додатку і унеможливорює повноцінне розуміння роботи програми

2.8 Code Integrity Checks (SafetyNet)

Code Integrity Checks визначається заходами, спрямованими на забезпечення цілісності вихідного коду додатка. Це важливий аспект забезпечення безпеки, який має на меті переконатися, що вихідний код додатка не був змінений або пошкоджений, що може стати на шляху до виконання шкідливого коду.

```
class CodeIntegrityChecker(private val listener: CodeIntegrityListener) {

    interface CodeIntegrityListener {
        fun onCodeIntegrityCheckResult(result: Boolean)
    }

    fun performCodeIntegrityCheck() {
        val nonce = "your_nonce_string"
        SafetyNet.getClient(/* context */)
            .attest(nonce.toByteArray(), /* your API key */)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    handleAttestationResult(task.result)
                } else {
                    if (task.exception is ApiException) {
                        val apiException = task.exception as ApiException
                        listener.onCodeIntegrityCheckResult(false)
                    }
                }
            }
    }

    private fun handleAttestationResult(result:
SafetyNetApi.AttestationResponse) {

        val resultIsValid = result.jwsResult.isNotEmpty()

        // Повідомлення слухача про результат
        listener.onCodeIntegrityCheckResult(resultIsValid)
    }
}
```

Таким чином, перевірка цілісності додатку є дуже важливою для подальшого доступу до ресурсу

2.9 Tamper-evident logging

Tamper-evident logging - це техніка, яка використовується для виявлення та показу того, чи були піддані втручанню журнали чи файли журналу. Це допомагає забезпечити цілісність та автентичність зареєстрованої інформації. У додатках для Android, написаних на Kotlin, можна реалізувати ведення журналу, яке стійке до втручання, використовуючи техніки, щоб журнали були стійкими до втручання. Ось простий приклад:

```

object TamperEvidentLogger {

    fun logMessage(message: String) {
        val timestamp = System.currentTimeMillis()
        val logEntry = "$timestamp: $message"

        // Обчислити хеш для запису в журнал
        val hash = calculateHash(logEntry)

        // Додати запис в журнал та його хеш у ваш файл журналу
        appendToLogFile("$logEntry - Хеш: $hash")
    }

    private fun calculateHash(data: String): String {
        val messageDigest = MessageDigest.getInstance("SHA-256")
        val hashBytes = messageDigest.digest(data.toByteArray())

        // Перетворити байти хешу в шістнадцятковий рядок
        val hexString = StringBuilder()
        for (byte in hashBytes) {
            val hex = Integer.toHexString(0xff and byte.toInt())
            if (hex.length == 1) hexString.append('0')
            hexString.append(hex)
        }

        return hexString.toString()
    }

    private fun appendToLogFile(logEntry: String) {
        ..
    }
}

```

Таким чином, логування у файли і передача на Firebase або іншу систему для зчитування логів користувача і покращення безпеки мобільного додатку

2.10 Dynamic Code Loading Restrictions

Ось простий приклад обмеження динамічного завантаження коду в Android за допомогою Kotlin. У цьому прикладі ми завантажимо динамічний модуль (DEX-файл) та виконаємо базові перевірки цілісності.

```

object DynamicCodeLoader {

    fun loadDynamicModule(context: Context, modulePath: String, expectedHash: String): Boolean {
        // Perform integrity check before loading
        val moduleFile = File(modulePath)
        if (!verifyCodeIntegrity(moduleFile.readBytes(), expectedHash)) {
            // Log or handle integrity check failure
            return false
        }

        // Use the default class loader for loading classes
        val classLoader = context.classLoader
    }
}

```

```

    try {
        // Load the dynamic module using DexClassLoader
        val dexClassLoader = DexClassLoader(
            modulePath,
            context.cacheDir.absolutePath,
            null,
            classLoader
        )

        // Load a class from the dynamic module
        val loadedClass =
dexClassLoader.loadClass("com.example.DynamicModuleClass")

        // Instantiate an object from the loaded class
        val instance = loadedClass.newInstance()
        return true
    } catch (e: Exception) {
        // Handle exceptions related to dynamic code loading
        return false
    }
}

private fun verifyCodeIntegrity(loadedCode: ByteArray, expectedHash:
String): Boolean {
    val actualHash = calculateHash(loadedCode)
    return actualHash == expectedHash
}

private fun calculateHash(data: ByteArray): String {
    val messageDigest = MessageDigest.getInstance("SHA-256")
    val hashBytes = messageDigest.digest(data)

    // Convert hash bytes to a hexadecimal string
    val hexString = StringBuilder()
    for (byte in hashBytes) {
        val hex = Integer.toHexString(0xff and byte.toInt())
        if (hex.length == 1) hexString.append('0')
        hexString.append(hex)
    }

    return hexString.toString()
}
}

```

Таким чином, для ще більш кращої безпеки додатку можна використовувати динамічне завантаження модулів і перевіряти їх цілісність

2.11 Manifest security

Найбільш часткові помилки при створенні маніфесту:

1. Режим резервного копіювання(`android:allowBackup="true"`). Якщо ввімкнено, зловмисник може зробити резервну копію або викрасти ваші конфіденційні дані програми

2. Режим налагодження коду (`android:debuggable="true"`). Зловмисник може налагодити програму та відкрити внутрішню архітектуру програми

2.12 Висновок до другого розділу

В другому розділі були проаналізовані проблеми в розробці сучасних мобільних додатків і використанні новітніх технологій для покращення якості роботи і безпеки мобільного додатку. Наведені приклади є сертифіковані від Google і тому мають вагомий вплив на створення якісного та безпечного додатку

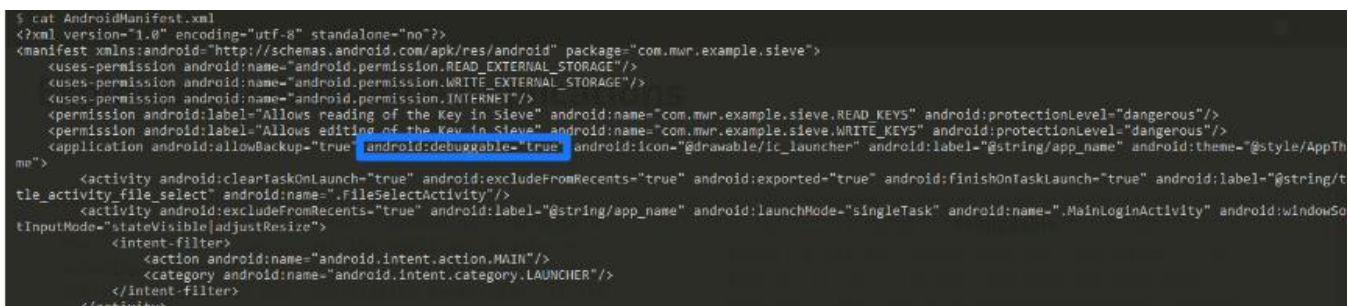
РОЗДІЛ 3 ПЕРЕВІРКА СИСТЕМ ЗАХИСТУ ДЛЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Пентестинг Manifest файлу

Якщо програму позначено як таку, що налагоджується, будь-який зловмисник може отримати доступ до даних програми, отримавши привілеї цієї програми, або може запустити довільний код із цим дозволом програми. У випадку програми, яка не налагоджується, зловмиснику спочатку потрібно отримати root-права на пристрій, щоб отримати будь-які дані.

Потрібно використати пакет Sieve, щоб продемонструвати цю проблему.

- Декомпілюйте файл .apk за допомогою apktool, а потім відкрийте файл AndroidManifest.xml.
- Потрібно знайти значення **android:debuggable** у файлі AndroidManifest.xml



```
$ cat AndroidManifest.xml
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.mwr.example.sieve">
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <permission android:label="Allows reading of the key in Sieve" android:name="com.mwr.example.sieve.READ_KEYS" android:protectionLevel="dangerous"/>
  <permission android:label="Allows editing of the key in Sieve" android:name="com.mwr.example.sieve.WRITE_KEYS" android:protectionLevel="dangerous"/>
  <application android:allowBackup="true" android:debuggable="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme">
    <activity android:clearTaskOnLaunch="true" android:excludeFromRecents="true" android:exported="true" android:finishOnTaskLaunch="true" android:label="@string/title_activity_file_select" android:name=".FileSelectActivity"/>
    <activity android:excludeFromRecents="true" android:label="@string/app_name" android:launchMode="singleTask" android:name=".MainLoginActivity" android:windowSoftInputMode="stateVisible|adjustResize">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Рисунок 3.1 – Демонстрація маніфеста

Тепер, щоб побачити, які програми підключені до налагоджувального сокера (@jdwp-control), потрібно ввести **adb jdwp** у консолі. У ньому буде перераховано PID (ідентифікатори процесу) програм, які можна налагодити.

```
$ adb jdwp
369
543
628
642
662
695
721
763
784
802
1012
1050
2192
3519
3877
```

Рисунок 3.2 – Демонстрація усіх процесів

Щоб визначити, який PID належить нашій програмі, потрібно ввести **adb jdwp** перед запуском програми, яка перевіряється.

```
$ adb jdwp
369
543
628
642
662
695
721
763
784
802
1012
1050
2192
3519
3877
```

Рисунок 3.3 – Знаходження правильного PID

Тепер знову вводиться команда **adb jdwp** після відкриття програми, і можна буде спостерігати, що до останнього результату додано PID, тобто серед цих PID є один PID, який належить нашій програмі. У випадку наведеному в прикладі додається лише один PID **3894**

```
$ adb shell ps | grep "3894"
u0_a39      3894  155   512484 29720 ffffffff b7538f37 S com.mwr.example.sieve
```

Рисунок 3.4 – Демонстрація виконання команди `adb shell ps`

Тепер потрібно перевірити, чи цей PID належить цій програмі, якщо більше одного PID. Він має вивести назву програми в кінці результату, наприклад **com.mwr.example.sieve**. Тепер, щоб просто продемонструвати, що може робити програма з можливістю налагодження, витягнемо дані з приватного каталогу програми. За допомогою `run-as binary` можна виконувати команди як додаток **com.mwr.example.sieve**

```
$ adb shell
shell@hwCHM-H:/ $ run-as com.mwr.example.sieve
run-as com.mwr.example.sieve
shell@hwCHM-H:/data/data/com.mwr.example.sieve $ |
```

Рисунок 3.5 – Демонстрація виконання команди `adb shell`

Тепер можна отримати дані або запустити довільний код, використовуючи дозвіл програми, як показано нижче.

```
$ adb shell run-as com.mwr.example.sieve ls -l
drwxrwx--x u0_a178 u0_a178      2015-10-08 03:22 cache
drwxrwx--x u0_a178 u0_a178      2015-10-08 03:22 databases
lrwxrwxrwx install install      2015-10-08 03:22 lib -> /data/app-lib/com.mwr.example.sieve-1
```

Рисунок 3.6 – Демонстрація виконання команди `adb shell`

Таким чином, було проведено покривний пентест маніфест файлу і показані можливості, які надає нам ця проблема в безпеці.

3.2 Binary protections – реверс інжиніринг додатку

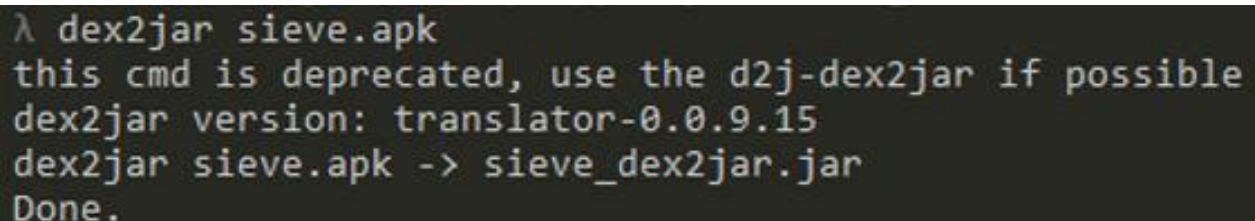
Відсутність бінарного захисту є останньою в OWASP Mobile Top 10 Risk

Додатки для Android постачаються у форматі файлу **.apk**, який зловмисник може перепроєктувати та бачити весь код, що міститься в ньому. Нижче наведено сценарії зворотного проектування програми:

- Зловмисник може проаналізувати та визначити, які захисні заходи реалізовано в додатку, а також знайти спосіб обійти ці механізми.
- Також зловмисник також може вставити шкідливий код, перекомпілювати його та доставити звичайним користувачам.
- Наприклад, ігрові додатки, у яких деякі функції розблоковані, молодь часто завантажує через незахищені джерела (іноді також через Google PlayStore). Більшість цих модифікованих додатків містять зловмисне програмне забезпечення, а деякі містять рекламу для отримання прибутку від цих користувачів.

Нижче наведено демонстрацію зворотного проектування програми.

- Я буду використовувати додаток Sieve для демонстрації.
- Спочатку використовую **dex2jar**, щоб перетворити файл **.apk** у файл **.jar**



```
λ dex2jar sieve.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar sieve.apk -> sieve_dex2jar.jar
Done.
```

Рисунок 3.7 – Демонстрація виконання команди dex2jar

- Потрібно відкрити файл jar у **jdgui**

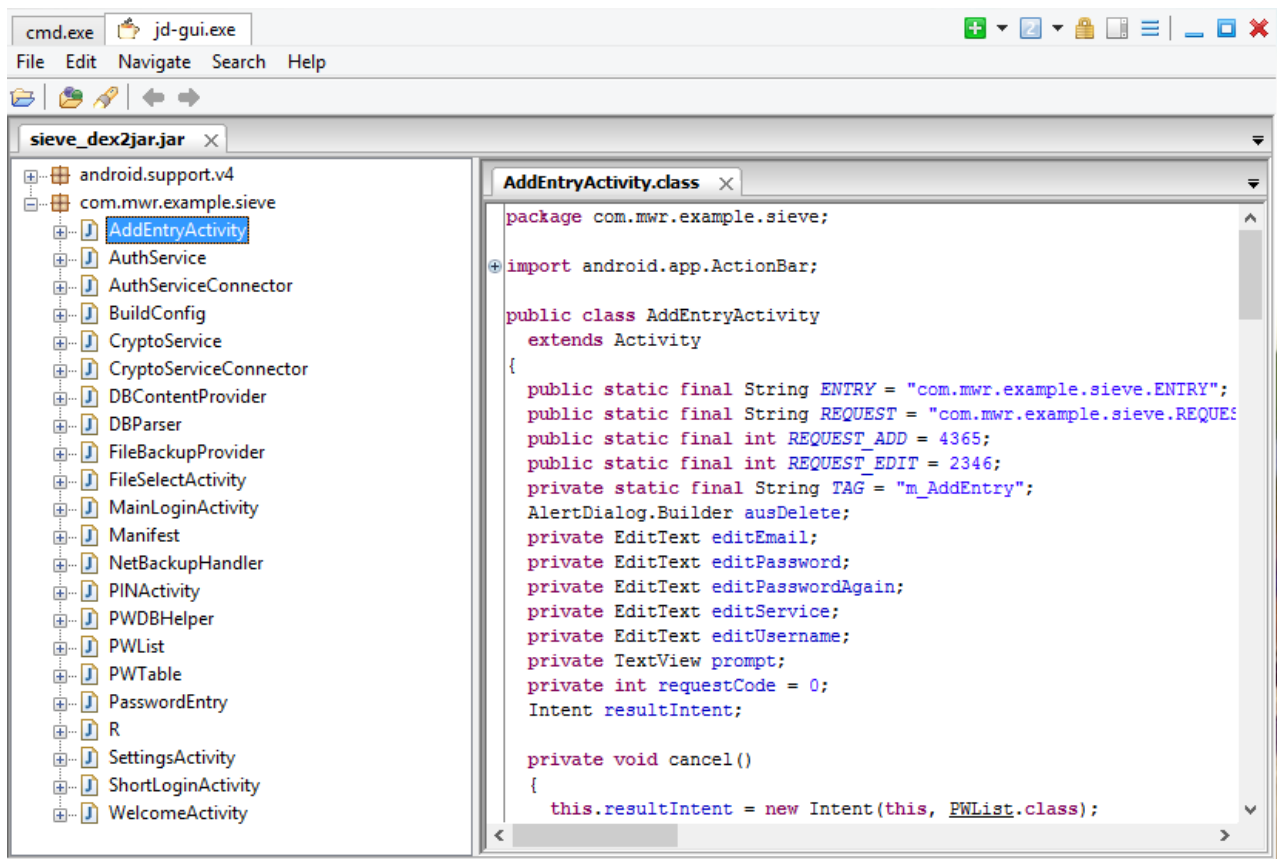


Рисунок 3.8 – Демонстрація відкриття файлу в jdgui

Таким чином, можна побачити на рисунку ми виконали наведені вище дії, тепер можемо побачити код і проаналізувати його

3.3 Тестування на Backdoor

Іноді розробник ставить бекдор до певної програми. Він/вона розміщує це, оскільки не хоче, щоб хтось інший мав доступ до цієї конфіденційної інформації, а іноді цей бекдор використовується для налагодження.

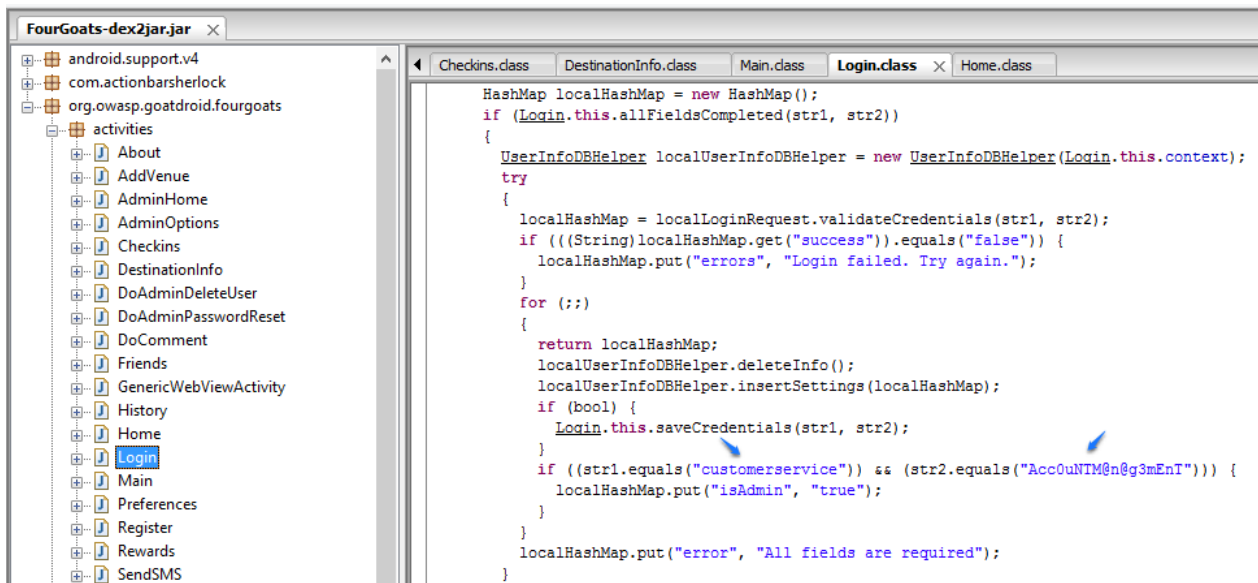


Рисунок 3.9 – Демонстрація backdoor

Таким чином, з наведеного вище зображення можна зрозуміти що пароль і логін юзера (було виділено на картинці) є таким і можна спробувати отримати додатковий доступ до додатку

3.4 Тестування з відкритим логуванням в додатку

Ненавмисний витік даних займає 4 позицію в рейтингу OWASP Mobile Top 10. Часто розробники залишають інформацію про налагодження публічно. Отже, будь-яка програма з дозволом READ_LOGS може отримати доступ до цих журналів і отримати конфіденційну інформацію через це.

- Pidcat — це модифікована версія logcat із кращим переглядом журналів.
- Тепер відкриваємо програму Sieve і вводимо свій пароль.

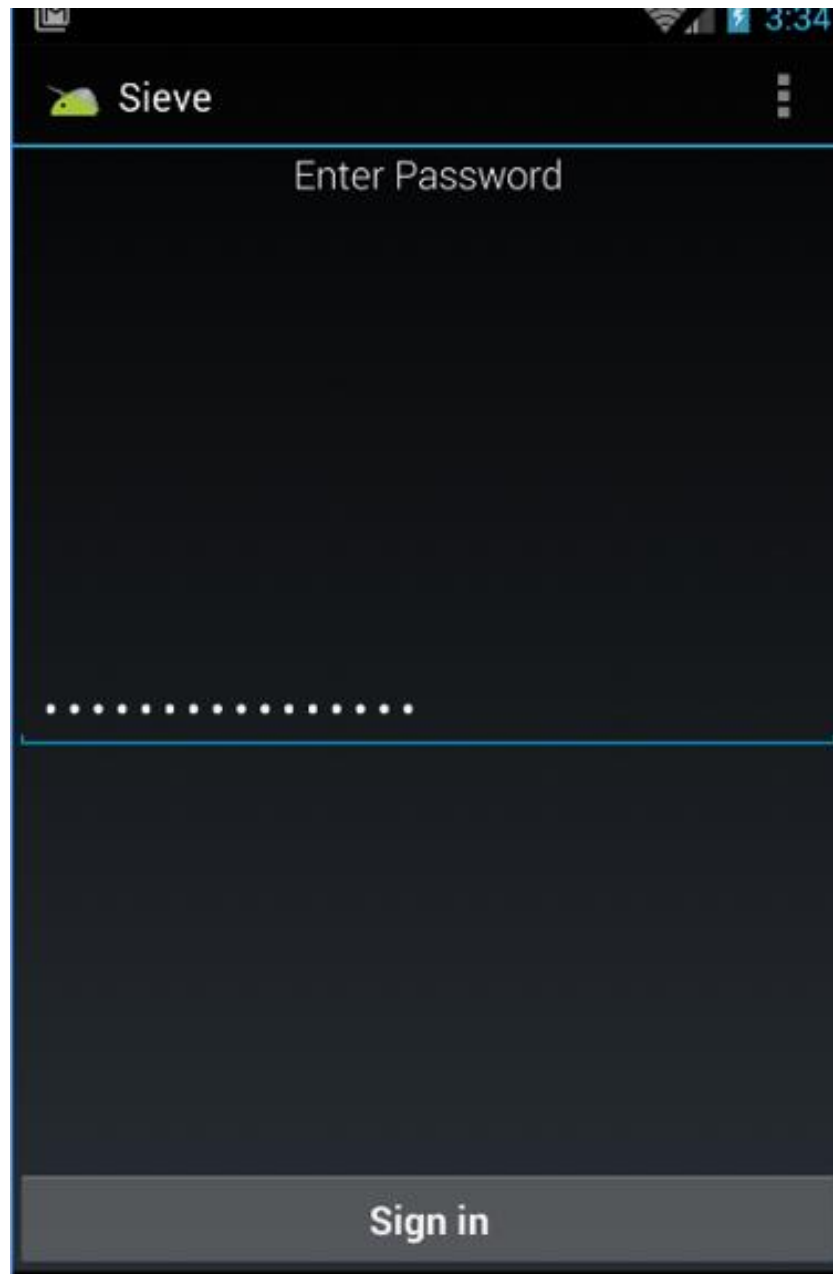


Рисунок 3.10 – Демонстрація вводу пароля

Тепер помітно, що пароль, який я ввів, відображається в підкат

```

^ pidcat com.mwr.example.sieve

Process com.mwr.example.sieve created for activity com.mwr.example.sieve/.MainLoginActivity
PID: 1868  UID: 10078  GIDs: {50078, 1028, 1015, 3003}

Process com.mwr.example.sieve:remote created for service com.mwr.example.sieve/.AuthService
PID: 1881  UID: 10078  GIDs: {50078, 1028, 1015, 3003}

libEGL: loaded /system/lib/egl/libEGL_genymotion.so
HostConnection::get() New Host Connection established 0xb92577e8, tid 1868
libGLESv1_CM: loaded /system/lib/egl/libGLESv1_CM_genymotion.so
libGLESv2: loaded /system/lib/egl/libGLESv2_genymotion.so
eglSurfaceAttrib not implemented
Enabling debug mode 0
Loading provider com.mwr.example.sieve.DBContentProvider: com.mwr.example.sieve.DBContentProvider
String entered: 1234567890123456
Trying to load lib /data/app-lib/com.mwr.example.sieve-1/libencrypt.so 0xa6a7ec58
[1881] Loading library(version: 3.2.1.43093 RELEASE)... successfully.

```

Рисунок 3.11 – Демонстрація вводу даних в підкат

Таким чином, на картинці 3.11 зображено пароль користувача, який було введено.

3.5 Незашифроване з'єднання з сервером

Недостатній захист транспортного рівня займає 3 місце в топ-10 OWASP Mobile

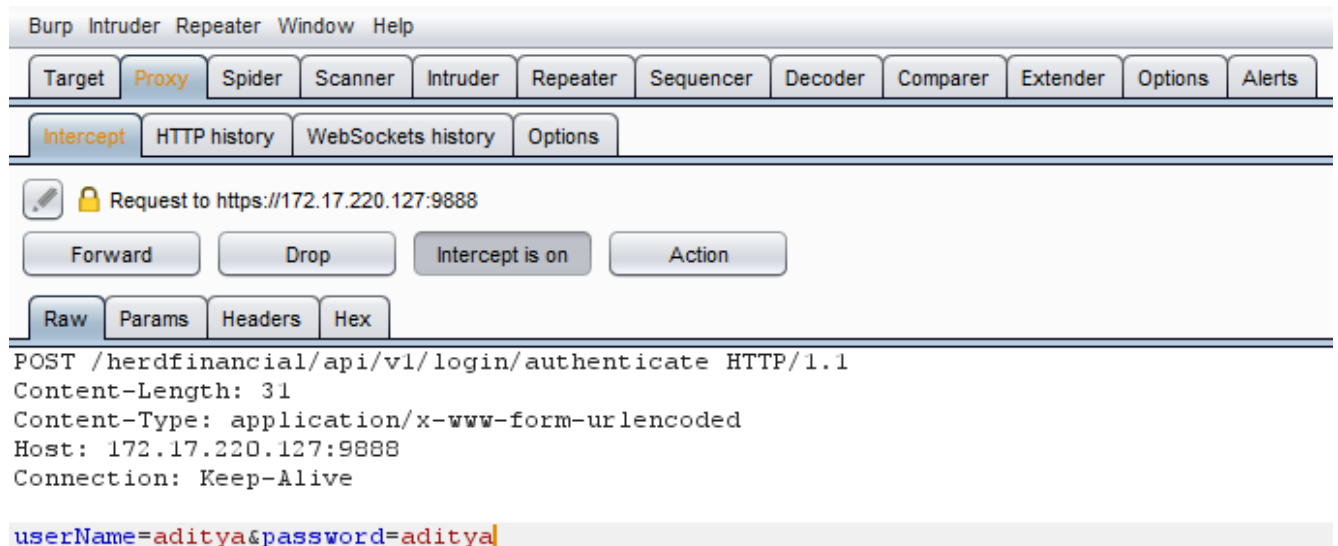


Рисунок 3.12 – Демонстрація перехвату відкритих даних

Якщо використати шифрування та наведені пункти рекомендації в розділі 2, можна уникнути такої проблеми



Рисунок 3.13 – Демонстрація перехвату закритих даних

Таким чином, при використанні методів шифрування можна отримати гарне шифрування, яке унеможливилює подальший OSINT додатку

3.6 Незахищене зберігання даних

Незахищене зберігання даних займає друге місце в рейтингу OWASP Mobile Top 10 .

Як я вже згадував в одній зі своїх попередніх розділах, за замовчуванням файли, які створюються у внутрішній пам'яті, доступні лише для вашої програми. Цей захист реалізований Android і його достатньо для більшості програм.

Але розробники часто використовують `MODE_WORLD_READABLE` & `MODE_WORLD_WRITABLE`, щоб надати ці файли деяким програмам, але це не обмежує іншим програмам (зловмисним) доступ до них.

Для демонстрації я використав програму FourGoats. Дані кожної програми зберігаються в `/data/data/` на пристрої Android. У кожній папці програми є `shared_prefs` папка і база даних і кілька інших папок, реалізованих програмою. Файли в цих папках належать до категорії внутрішнього сховища. У більшості програм ви побачите, що файли в папці `shared_prefs` доступні для читання, і навіть файли з конфіденційними даними є загальнодоступними

```
root@vbox86p:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs # ls -al
ls -al
-rw-rw-r-- u0_a99 u0_a99      209 2015-01-14 13:55 credentials.xml
-rw-rw-r-- u0_a99 u0_a99      153 2015-01-14 13:55 destination_info.xml
-rw-rw-r-- u0_a99 u0_a99      148 2015-01-14 13:55 proxy_info.xml
root@vbox86p:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs # cat credentials.xml
redentials.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="password">goatdroid</string>
  <boolean name="remember" value="true" />
  <string name="username">goatdroid</string>
</map>
```

Рисунок 3.14 – Демонстрація незахищених даних

На зображенні вище видно, що всі файли в папці `shared_prefs` програми FourGoats доступні для читання. Таким чином, будь-яка шкідлива програма також може отримати доступ до вмісту цих файлів на нерутованому пристрої. Основне занепокоєння тут полягає в тому, що іноді розробники також зберігають пари ім'я користувача та пароль у наступних файлах, що може призвести до компрометації облікового запису користувача.

Таким чином, зберігання чутливої інформації у SharedPreferences є не дуже безпечним варіантом

3.7 Проблеми з авторизацією

Погана авторизація та автентифікація посідають 5-е місце в топ-10 OWASP Mobile Security.

У більшості випадків після автентифікації в програмі Android вона переходить до нової діяльності, про яку користувачі в основному знають. Але розробники залишають ці дії експортованими навіть без спеціальних дозволів.

- Якщо ви побачите в програмі Financial, то побачите, що MainActivity експортується, а також не має жодних спеціальних дозволів
- Ви можете просто передати намір через Drozer, щоб почати цю конкретну дію

```
dz> run app.activity.start --component org.owasp.goatdroid.herdfinancial org.owasp.goatdroid.herdfinancial.activities.Main
```

Рисунок 3.14 – Демонстрація виконання Drozer

Після цього вас перекине на наступний екран, де завантажаться стандартні дані, які були виставлені розробником, якщо в тестуванні були помилки, то ви можете отримати доступ до цього екрану з додатковими можливостями



Рисунок 3.15 – Демонстрація перехвату пакету

Тепер можна змінити цей номер облікового запису на номер іншого облікового запису, як я змінив **7894561230** на **1234567890**

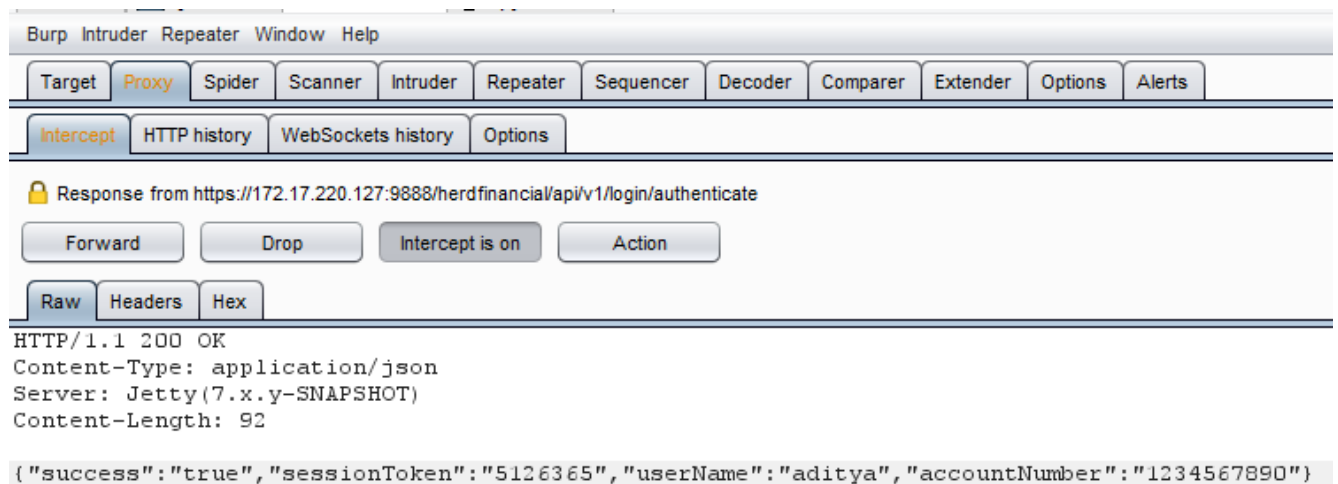


Рисунок 3.16 – Демонстрація зміни перехваченого пакету

Тепер, якщо ви скористаєтеся програмою Financial, ви побачите, що в ній є фактично всі реквізити рахунку **1234567890**, і ви також можете здійснити переказ від імені **1234567890**. Це означає, що тепер ви володієте банківським рахунком іншої особи, просто знаючи номер його/її рахунку

Таким чином, **android:exported="true"** для головних компонентів андроїд, які не взаємодіють з іншими додатками несуть потенційну проблему для подальшого взлому

3.8 Висновок до третього розділу

Згідно із виявленими недоліками і висновками в кожному підрозділі, прийнято використовувати методи захисту, які були наведені в розділі 2

ВИСНОВКИ

У першому розділі розглянуті методи шифрування даних в мобільних пристроях Android, яке відбувається за рахунок базового шифрування файлів, захищеного зберігання ключів та конфігурування мережевої безпеки. Авторизації користувача може відбуватися за біометричними даними (обличчя, відбитки пальців, розпізнавання голосу, очей).

У другому розділі запропоновано параметри налаштування системи захисту мобільного додатку за рахунок безпечної взаємодії з сервером, шифрування бази даних, заплутання коду додатку для ускладнення реверс-інженірингу додатку, захисту від SSL Pinning атак, SSL атак на пристроях зі старими сервісами, Rooting hack, відключення налагоджувальних можливостей, перевірки незмінності коду, захисту журналів додатку, обмеження динамічного завантаження коду, налагодження Manifest файлу для унеможливлення завантаження резервної копії додатку.

У третьому розділі демонстративним методом проведено пентестинг системи та запропоновано методи захисту.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bill Phillips, Chris Stewart & Kristin Marsicano — Android Programming: The Big Nerd Ranch Guide [Електронний ресурс]. — 2015. — Режим доступу: <https://www.amazon.com/Android-Programming-Nerd-Ranch-Guide/dp/0134171454>
2. Murat Y. Expert Android Studio / Y. Murat, D. Onur., 2016.
3. OWASP Mobile Security Testing Guide [Електронний ресурс] – Режим доступу до ресурсу: <https://owasp.org/www-project-mobile-security-testingguide/>
4. Talukder M. Analysis of Obfuscated Code with Program Slicing [Електронний ресурс] / M. Talukder, S. Islam, P. Falcarin – Режим доступу до ресурсу: https://syedislam.com/publications/Analysis_of_Obfuscated_code_with_Program_Slicing-final-a4.pdf
5. OWASP Mobile Top 10 [Електронний ресурс] – Режим доступу до ресурсу: <https://owasp.org/www-project-mobile-top-10/>
6. Advanced Binary Deobfuscation [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/malrev/ABD>
7. Mueller B. Hacking Soft Tokens. Advanced Reverse Engineering on Android [Електронний ресурс] / Bernhard Mueller // Vantage Point Security Pte. Ltd.. – 2016. – Режим доступу до ресурсу: <https://gsec.hitb.org/materials/sg2016/whitepapers/Hacking%20Soft%20Tokens%20-%20Bernhard%20Mueller.pdf>
8. Increase Code Complexity and Use Obfuscation [Електронний ресурс] – Режим доступу до ресурсу: <https://books.nowsecure.com/secure-mobiledevelopment/en/coding-practices/code-complexity-and-obfuscation.html>
9. Automated Deobfuscation of Android Native Binary Code [Електронний ресурс] / [Z. Kan, H. Wang, L. Wu та ін.] – Режим доступу до ресурсу: <https://arxiv.org/pdf/1907.06828.pdf>
10. Secure an Android Device [Електронний ресурс] // Source Android – Режим доступу до ресурсу: <https://source.android.com/security>.

11. Android Q privacy change: Scoped storage [Електронний ресурс] // Developers Android – Режим доступу до ресурсу: <https://developer.android.com/preview/privacy/scoped-storage>.
12. Man-in-the-Disk: Android Apps Exposed via External Storage [Електронний ресурс] // CHECK POINT RESEARCH. – 2018. – Режим доступу до ресурсу: <https://research.checkpoint.com/androids-man-in-the-disk/>
13. Android (operating system) [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: [https://en.wikipedia.org/wiki/Android_\(operating_system\)](https://en.wikipedia.org/wiki/Android_(operating_system))
14. Розробка для Android [Електронний ресурс]. – Режим доступу : <http://developer.android.com/>
15. C.J. DateAn Introduction to Database / C.J. DateAn // – Pearson – 2003. – 1024.
16. Burp and Mobile Devices — PortSwigger — Режим доступу: <https://portswigger.net/support/burp-and-mobile-devices>.
17. Drozer [Електронний ресурс] – Режим доступу до ресурсу: <https://pypi.org/project/drozer/>.
18. Гончаренко Д. А. Перехоплення мережевого трафіку у мобільних застосунках, що ігнорують системні налаштування проксі-серверу / Д. А. Гончаренко, М. В. Коломицев. // XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики». – 2021. – С. 214–216.
19. FRIDA – Dynamic instrumentation toolkit for developers, reverseengineers, and security researchers [Електронний ресурс] – Режим доступу до ресурсу: <https://frida.re>
20. CVE-2012-6636 [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: <https://nvd.nist.gov/vuln/detail/CVE-2012-6636>.