НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«До захисту допущено»
В.о. завідувача кафедри

_____ М.В.Грайворонський
(підпис)
“___”_____2019 р.

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

на тему: Розробка рекомендацій щодо унеможливлення атаки «Man in the Disk» для платформи Android

Виконав (-ла): студент (-ка) 4 курсу, групи ФБ-51
(шифр групи)

Чумаченко Роман Євгенович (підпис)
(прізвище, ім’я, по батькові)

Керівник: доцент к.ф.-м.н. Орєхов О.А. (підпис)
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

Консультант _____ _____ _____ (підпис)
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали)

Рецензент_____ _____ (підпис)
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.170101 «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)
«___»_____2019 р.

ЗАВДАННЯ
на дипломну роботу студенту

_____Чумаченко Роману Євгеновичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка рекомендацій щодо унеможливлення атаки «Man in the Disk» для платформи Android _____ ,
науковий керівник роботи доцент к.ф.-м.н. Орєхов О.А._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____ » 2019 р. № _____

2. Термін подання студентом роботи 10 червня 2019 р.

3. Вихідні дані до роботи _____
_____
_____

4. Зміст роботи_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) _____
_____
_____
_____

6. Дата видачі завдання _____

РЕФЕРАТ

Представлена дипломна робота складається з чотирьох розділів, загальний обсяг роботи – 88 сторінки. Містить 13 літературних посилань, 17 ілюстрацій, 4 таблиці.

Метою роботи є підвищення захищеності ОС Android, щоб унеможливити розробку застосунків, вразливих до різних видів атак. Для досягнення поставленої мети потрібно дослідити особливості роботи операційної системи Android, пов’язані з атакою «Man in the Disk» та алгоритм проведення атаки, проаналізувати підходи до вирішення подібних проблем, сформувати механізм захисту та на їх основі запропонувати рекомендації.

Об’єктом досліджень даної роботи є інформаційні процеси у сучасних мобільних операційних системах.

Предметом дослідження виступають атаки на застосунки ОС Android та механізми захисту від них.

Під час виконання даної роботи було проведено дослідження особливостей функціонування ОС Android, аналіз проведення атаки за допомогою опрацьованих джерел, вироблено механізми захисту, проведена їх часткова реалізація з метою перевірки концепції та запропоновані рекомендації по уникненню атаки.

Новизна полягає у тому, що в даній роботі вперше розглянуто механізми захисту від такого типу атак у вигляді рекомендацій по створенню бібліотеки, яка унеможливлює створення вразливих застосунків.

Практична значимість роботи полягає у можливості створення виробниками пристроїв тимчасового рішення даної вразливості на основі запропонованих рекомендацій.

Ключові слова: Man in the Disk, Android, вразливість, зовнішнє сховище, безпека, рекомендації.

РЕФЕРАТ

Представленная дипломная работа состоит из четырех разделов, общий объем работы - 88 страницы. Содержит 13 литературных ссылок, 17 иллюстраций, 4 таблицы.

Целью работы является повышение защищенности ОС Android, чтобы сделать невозможным разработку приложений, уязвимых к различным видам атак. Для достижения поставленной цели необходимо исследовать особенности работы операционной системы Android, связанные с атакой «Man in the Disk» и алгоритм проведения атаки, проанализировать подходы к решению подобных проблем, сформировать механизм защиты и на их основе предложить рекомендации.

Объектом исследований данной работы являются процессы в современных мобильных операционных системах.

Предметом исследования выступают атаки на приложения ОС Android и механизмы защиты от них.

Во время выполнения данной работы было проведено исследование особенностей функционирования ОС Android, анализ проведения атаки с помощью обработанных источников, выработаны механизмы защиты, проведена их частичная реализация с целью проверки концепции и предложены рекомендации по избежанию атаки.

Новизна заключается в том, что в данной работе впервые рассмотрены механизмы защиты от таких атак в виде рекомендаций по созданию библиотеки, которая делает невозможным создание уязвимых приложений.

Практическая значимость работы заключается в возможности создания производителями устройств временного решения данной уязвимости на основе предложенных рекомендаций.

Ключевые слова: Man in the Disk, Android, уязвимость, внешнее хранилище, безопасность, рекомендации.

ABSTRACT

The presented thesis consists of four sections, the total amount of work is 88 pages. It contains 13 literary references, 17 illustrations, 4 tables.

The aim of the work is to increase the security of the Android OS in order to make it impossible to develop applications that are vulnerable to various types of attacks. To achieve this goal, it is necessary to investigate the features of the Android operating system associated with the Man in the Disk attack and the algorithm of the attack, analyze the approaches of solving such problems, form a defense mechanism and offer recommendations based on them.

The object of research in this work is the processes in modern mobile operating systems.

The subject of the research is attacks on Android OS applications and defense mechanisms against them.

During this work was conducted a study about the features of Android OS. To analyze the attack different sources were processed. Protection mechanisms were developed and partially implemented to verify the concept. Based on this were given recommendations about attack evasion.

The novelty of the work is in the fact that for the first time in this work, the mechanisms of protection against such attacks were considered in the form of recommendations for creating a library that makes it impossible to create vulnerable applications.

The practical significance of the work lies in the possibility that manufacturers can create a temporary solution to this vulnerability based on the proposed recommendations.

Keywords: Man in the Disk, Android, vulnerability, external storage, security, recommendations.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ОС – операційна система

MitD – Man in the Disk

HAL – Hardware Abstraction Layer

JVM – Java Virtual Machine, віртуальна машина Java

Dalvik VM – Dalvik Virtual Machine, віртуальна машина Dalvik

Sandbox – «пісочниця», захисний механізм для контрольованого безпечного виконання застосунків

ВСТУП

Швидкий розвиток технологій, що спостерігається сьогодні, тягне за собою розвиток та мініатюризацію комплектуючих для обчислювальної техніки, що сприяє підвищенню доступних потужностей. Це не призводить не тільки до створення більш якісних рішень вже існуючих питань, але й до розробки рішень ряду нових завдань, вирішення яких раніше було не під силу. Також розвиток технологій сприяє спрощенню процесу виробництва комплектуючих, що призводить до зниження їх вартості та, відповідно, зростанню доступності. Сукупність цих факторів обумовлює зростаючу популярність мобільних пристроїв, адже з року в рік вони отримують все більше нових можливостей, їх потужність зростає, внаслідок чого покращується і досвід користувача, а зниження вартості їх комплектуючих сприяє створенню якісних, але доступних пристроїв. Результатом цього є впевнено зростаюча популярність мобільних пристроїв протягом останніх 10 років.

Сучасні мобільні пристрої оброблюють та зберігають велику кількість різноманітних даних, велика частина яких становить чутливу інформацію про користувача або ж інформацію, що користувачу належить. Також особливістю таких пристроїв є наявність нестандартних засобів введення – камера, мікрофон, GPS. Сукупність цих факторів робить мобільні пристрої бажаною ціллю для зловмисників.

На даний момент на ринку домінують пристрої під керуванням операційних систем Android та iOS. Згідно статистики [1], станом на квітень 2019 75.22% всіх мобільних пристроїв функціонують на базі Android. Не зважаючи на покращення безпеки, що вносилися в дану систему з кожним оновленням, спеціалісти продовжують знаходити нові типи вразливостей. Так, відкрита в серпні 2018 року вразливість, що надає можливість проведення нового для даної платформи типу атака «Man in the Disk», призводить до створення нової поверхні атак. Спеціалісти

виявили серед великої кількості вразливих застосунків навіть продукти таких лідерів індустрії, як Google, Xiaomi, Yandex, LG та Epic Games.

Наведене вище визначило актуальність даної роботи – розробка рекомендацій щодо унеможливлення атаки «Man in the Disk» для платформи Android.

Метою дипломної роботи є підвищення захищеності ОС Android з метою унеможливити розробку застосунків, вразливих до різних видів атак.

Для досягнення поставленої мети необхідно виконати такі завдання:

1) дослідити особливості роботи операційної системи Android, пов’язані з атакою «Man in the Disk» та алгоритм проведення атаки
2) проаналізувати підходи до вирішення подібних проблем
3) сформувати механізм захисту
4) запропонувати рекомендації

Об’єктом досліджень даної роботи є інформаційні процеси у сучасних мобільних операційних системах.

Предметом дослідження виступають атаки на застосунки ОС Android та механізми захисту від них.

Новизна полягає у тому, що в даній роботі вперше розглянуто механізми захисту від такого типу атак у вигляді рекомендацій по створенню бібліотеки, яка унеможливлює створення вразливих застосунків.

Практична значимість роботи полягає у можливості створення виробниками пристроїв тимчасового рішення даної вразливості на основі запропонованих рекомендацій.

1 ОСОБЛИВОСТІ ФУНКЦІОНУВАННЯ ANDROID ТА ОГЛЯД АТАКИ «MITD»

1.1 Особливості функціонування операційної системи, пов’язані з атакою «Man in the Disk»

1.1.1 Загальні відомості

Операційна система Android розробляється компанією Google, вихідний код потрапляє у відкритий доступ після випуску нової версії операційної системи. Більшість вихідного коду операційної системи – включаючи мережевий стек та телефонію – публікується під ліцензією Apache License v2, яка дозволяє внесення модифікацій та подальше поширення [2]. Такий спосіб поширення надає можливість виробникам мобільних пристроїв видозмінювати та налаштовувати операційну систему з урахуванням особливостей архітектури пристроїв власного виробництва, а також з метою покращення користувацького досвіду.

Слід зауважити, що тільки базова версія операційної системи та деяке програмне забезпечення до неї поширюється відкрито, тоді як більшість пристроїв поставляються разом з фірмовим програмним забезпеченням, таким як Google Mobile Services (надалі GMS), що включає в себе як ряд застосунків (Gmail, Google Play Store та інші), так і Google Play Services – прикладний програмний інтерфейс необхідний для взаємодії прикладного програмного забезпечення з сервісами, що надає Google. Для використання такого програмного забезпечення виробники повинні пройти ліцензування від Google, головною умовою якого є відповідність стандартам сумісності. Таким чином, на абсолютній більшості мобільних пристроїв, що функціонують на базі Android, насправді встановлено певним чином модифіковану версію Android. В залежності від виробника, зміни бувають як відносно незначні (зміни у дизайні графічного інтерфейсу користувача, зміни в набір попередньо встановленого програмного

забезпечення), так і значними (виклики методів системних бібліотек приводять до інших результатів, різна кількість дозволених одночасно користувачів та інше). Це також стосується й деяких налаштувань безпеки.

1.1.2 Механізми безпеки Android

1.1.2.1 Механізми безпеки рівня користувача

На користувацькому рівні Android впроваджує різні типи аутентифікації для розблокування пристрою, починаючи з PIN-кодів, паролів та візерунків (pattern) і закінчуючи скануванням відбитків пальців та системи аутентифікації за обличчям, використання яких стало можливим завдяки розвитку технологій та зниженню ціни на комплектуючі частини.

Деякі з найбільш технологічних мір захисту – сканування обличчя, наприклад – кожний виробник реалізує у свій спосіб, тож їх ефективність, на момент написання роботи, різниться. Так, згідно експерименту журналісту Forbes [3], обійти механізм аутентифікації за обличчям можливо завдяки використанню 3D-моделі обличчя. В експерименті було протестовано наступні флагманські пристрої під управлінням Android: LG G7 ThinQ, Samsung Note 8 та One Plus 6. За результатами, всіх з протестованих пристроїв розпізнали модель, як справжнє коректне обличчя і авторизували «зловмисника». І справді, деякі з виробників робили попередження через діалогове вікно, що даний механізм сьогодні не є стовідсотково надійним і його слід використовувати в якості додаткової міри аутентифікації.

Також операційна система надає прикладний програмний інтерфейс, що надає змогу виробникам застосунків впроваджувати запит на додаткову аутентифікацію перед виконанням певних дій у застосунку або ж перед запуском самого застосунку. Обробка такого запиту виконується операційною системою,

що виключає помилки на етапі створення розробниками власних механізмів аутентифікації та помилок при збереженні і обробці облікових даних (використовуються дані, надані системі користувачем для аутентифікації). Деякі виробники використовують цей та інші програмні інтерфейси, пов’язані з безпекою, для розширення можливостей стандартної версії Android. Так, в пристроях виробника Xiaomi, що працюють на базі надбудови розширеної версії Android , MIUI, можливість налаштування додаткових запитів на аутентифікацію передбачена на рівні операційної системи, що дає змогу викликати цей процес перед запуском будь-якого обраного користувачем застосунку, навіть того, де такий функціонал не було передбачено розробниками. Також в MIUI передбачена обов’язкова аутентифікація за паролем кожні 72 години для пристроїв, що використовують біометричні дані для аутентифікації.

1.1.2.2 Загальна структура операційної системи Android

Рисунок 1.1 зображує основні рівні програмного стеку Android та їх компоненти. Ідея полягає в тому, що кожний компонент у стеку вважає всі компоненти нижче за стеком надійними з точки зору безпеки [4].

Рисунок 1.1 – Програмний стек Android

Починаючи знизу кожен рівень виконує:

- Linux Kernel – ядро операційної системи, надає драйвери пристроїв та керує живленням
- HAL (Hardware Abstraction Layer, рівень апаратної абстракції) – загальний інтерфейс для драйверів рівня ядра, реалізується виробниками пристроїв, більшість тестувань на отримання сертифікату сумісності від Google припадає саме на функціонування HAL [5]
- Native Libraries (бібліотеки, написані мовою C/C++) – надають основний функціонал для іншого програмного забезпечення (робота з SSL, СУБД SQLite та інше)

- Android Runtime – основні бібліотеки та віртуальна машина для виконання застосунків Dalvik VM
- Android Framework – «обгортка» навколо native libraries для створення програмного забезпечення на мовах програмування високого рівня Java та Kotlin, цей код транслюється у байт-код для Dalvik VM та поставляється, як частина інсталяційного файлу кінцевому користувачу
- Applications – рівень прикладного програмного забезпечення, представлений попередньо встановленими застосунками та застосунками, встановленими користувачем

З метою розділення ресурсів між застосунками, ізоляції системних ресурсів від застосунків та даних користувача від застосунків, система використовує певні особливості роботи ядра Linux, зобов’язує всі застосунки працювати у режимі sandbox («пісочниця», механізм, що надає обмежені права та ресурси процесу), реалізує безпечний зв’язок між процесами.

1.1.2.3 Безпека на рівні операційної системи

В якості основи для мобільної операційної системи, ядро Linux надає такі елементи посилення безпеки:

- Модель дозволів, заснована на користувачах
- Ізоляція процесів
- Безпечний механізм IPC (Inter-Process Communication, зв’язок між процесами)

Ці механізми, а також дозволи файлової системи аналогічні до інших Unix-систем, лягли в основу надійний механізму sandbox для програмного забезпечення. За винятком ядра та деякого системного програмного забезпечення,

весь код операційної системи виконується у режимі sandbox (лише ядро та деяке системне програмне забезпечення працюють в режимі root-користувача).

1.1.2.4 Sandbox застосунків

Платформа використовує переваги моделі користувачів Linux-систем для ізоляції ресурсів застосунків, а також захисту системи та застосунків від шкідливих програм.

Для цього операційна система призначає унікальний ідентифікатор користувача (UID, user ID) кожному Android-застосунку та запускає їх, як окремі процеси. Таким чином, sandbox встановлюється системою на рівні ядра. Ядро відповідає за безпеку між застосунками та операційною системою на рівні процесів за допомогою стандартних об’єктів Linux (ідентифікатори групи та користувачів, що надаються програмному забезпеченню). За замовченням, застосунки не можуть звертатися один до одного та мають вкрай малий доступ до сервісів операційної системи. Так, наприклад, якщо застосунок намагається надіслати SMS-повідомлення, але не має на це дозволу, то операційна система не дозволить цього. Механізм роботи sandbox базується на основі перевіреної роками Unix-подібної системи розділення користувачів, процесів та файлів.

Враховуючи те, що sandbox здійснюється на рівні ядра, то він розповсюджується не тільки на рівень застосунків, але й на native libraries та програмне забезпечення операційної системи.

Даний механізм зазнавав оновлень та покращень з оновленнями операційної системи [6]. Так, починаючи з Android 5.0, за допомогою SELinux впроваджено мандатне керування доступом для розділення операційної системи та застосунків. З Android 6.0 за допомогою SELinux можливості sandbox були розширені для розділення застосунків між декількома фізичними користувачами пристрою, а стандартні права доступу на домашній каталог застосунку були змінені з 751 на

більш безпечні 700. До Android 9 sandbox виконував всі сторонні додатки в одному окремому від системи контексті SELinux, з цієї версії всі непривілейовані програми повинні запускатися в окремому контексті, що зробило розділення застосунків між собою ефективнішим.

1.1.2.5 Система дозволів

Android влаштовано таким чином, що будь-який застосунок повинен запросити дозвіл на виконання дій, що потенційно можуть зашкодити іншим застосункам, операційній системі або особистим даним користувача. Тобто на виконання таких дій, як: отримання доступу до списку контактів користувача, доступу в мережу Інтернет та використання камери пристрою застосунку необхідно отримати дозвіл.

Під час створення застосунку розробник повинен зазначити необхідні для функціонування дозволи у спеціальному файлі – маніфесті застосунку. В залежності від типу дозволу він може бути наданий системою автоматично на етапі встановлення, за результатом перевірки під час виклику метода, що потребує цього дозволу, або ж тільки з підтвердження користувача.

З розвитком операційної система, система дозволів також зазнала певних змін. На даний момент всі дозволи поділені на рівні захищеності:

- Дозволи нормального рівня – дозволи цього рівня надаються системою автоматично під час встановлення застосунку, до них відносяться дозволи на доступ в мережу Інтернет, використання вібрації, використання відбитків пальців, доступ до NFC та інше
- Сигнатурні дозволи – так як кожний застосунок повинен бути підписаний сертифікатом розробника, дозволи цього рівню дозволяють керувати, які сервіси чи застосунки можуть звертатися до даного, система надає такі дозволи тільки під час запиту застосунків,

що підписані тим самим сертифікатом, що й застосунок, який оголосив ці дозволи (тобто, якщо застосунок А оголосив сигнатурні дозволи, то при такому запиті від застосунку Б до А, що потребує наявності в Б дозволів, оголошених А, вони будуть надані системою тільки якщо обидва застосунки А і Б були підписані однаковим сертифікатом)

- Небезпечні дозволи – дозволи, що стосуються доступу до приватної інформації користувача чи потенційно здатні змінювати дані користувача чи інших застосунків; такі дозволи надаються системою лише після того, як користувач підтвердить запит на дозвіл
- Спеціальні дозволи – декілька особливих дозволів, що оброблюються системою інакше і на практиці зустрічаються вкрай рідко через свою специфічність

1.1.2.6 Розповсюдження та інсталяція застосунків

В даній операційній системі застосунки представлені файлами розширення apk (Android Package Kit), що являють собою варіант jar-архіву, який, в свою чергу, побудований на основі zip-архіву. Такий архів містить відкомпільований вихідний код у dex-файлі (Dalvik Executable), необхідні ресурси (розмітки графічного інтерфейсу, строки та інше), асети, сертифікати та маніфест.

Розповсюдження таких файлів відбувається через офіційні платформи цифрової дистрибуції – такі сервіси, як Google Play Store – або ж через завантаження з мережі Інтернет. У випадку встановлення застосунків через Google Play Store процес проходить автоматично, що стосується встановлення apk-застосунків, завантажених з мережі Інтернет, то для цього необхідно через налаштування надати дозвіл на встановлення застосунків з ненадійних джерел. В старих версіях Android цей дозвіл надавався на рівні всієї операційної системи,

таким чином даючи можливість провести спробу встановлення застосунку з ненадійних джерел будь-якому вже встановленому на пристрої програмному забезпеченні. В останніх версіях Android цей дозвіл зробили гранулярним, таким чином користувач може самостійно обирати, які застосунки мають право на встановлення, а які ні. На рисунку 1.2. зображено панель налаштувань для даного дозволу.

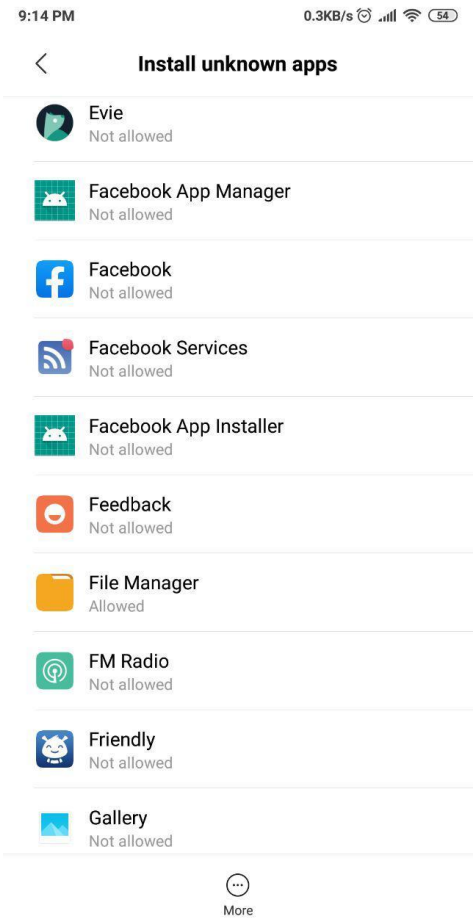

Рисунок 1.2 – Панель налаштування дозволів

1.1.2.7 Особливості файлового сховища

На рівні прикладного програмного забезпечення можливо виділити два типи сховищ [7]:

- Внутрішнє сховище

- Зовнішнє сховище

Внутрішнє сховище – окремий для кожного застосунку каталог, він та його вміст не може бути змінений чи прочитаний а ні іншими застосунками, а ні користувачем. Ця особливість робить внутрішнє сховище найкращим місцем для зберігання важливих для застосунку файлів, в прямому доступі до яких користувач не має потреби.

Зовнішнє сховище являє собою спеціально відформатований розділ внутрішньої файлової системи або зовнішній накопичувач (карта пам’яті). Дане сховище за своєю природою не є постійно доступним на відміну від внутрішнього сховища – спеціальний розділ може бути підключено до комп’ютера як зовнішній накопичувач, чи, у випадку карти пам’яті, фізично видалено. Також важливою особливістю є те, що це сховище доступне для читання та запису як для інших застосунків системи та користувача, так і для будь-кого, хто отримує доступ до пристрою.

В залежності від типу запитів до зовнішнього сховища, застосунку може знадобитися дозвіл з групи небезпечних. У випадку, коли застосунок звертається тільки до свого «приватного» каталогу (окремий каталог застосунку, в якому можуть зберігатися ассети у формі .obb файлів або файли, необхідні для роботи застосунку) необхідності у дозволах немає. У випадку, коли застосунок звертається до будь-яких інших каталогів зовнішнього сховища, застосунок повинен запитувати один з наступних дозволів:

- READ_EXTERNAL_STORAGE – дозвіл на читання зовнішнього сховища
- WRITE_EXTERNAL_STORAGE – дозвіл на запис до зовнішнього сховища; отримання цього дозволу неявно надає дозвіл на читання READ_EXTERNAL_STORAGE

1.2 Огляд атаки «Man in the Disk»

1.2.1 Передумови виникнення

Дана атака була виявлена в серпні 2018 року. Вона базується на певних особливості в дизайні зовнішнього файлового сховища операційної системи Android. Як згадувалося раніше, жорстке розмежування доступу, що застосовується до файлів внутрішнього сховища, не поширюється на розділ зовнішнього сховища. Таким чином, недбале використання розробником зовнішнього сховища призведе до вразливості застосунку.

1.2.2 Можливі наслідки

На даний момент відомо, що вдале проведення даної атаки може призвести до наступних наслідків [8]:

- «тихе» встановлення небажаного та потенційно небезпечного програмного забезпечення
- Відмова в обслуговуванні програмного забезпечення
- Критичний збій у роботі програми, потенційно здатний відкрити шлях до ін’єкцій зловмисного коду, який буде виконано в контексті та з набором привілеїв скомпрометованого застосунку

Таким чином, вдало проведена атака «Man in the Disk» призводить до компрометації механізму захисту sandbox в Android, так як механізм «пісочниці» не зміг виконати деякі з своїх основних цілей: ізоляція застосунків один від одного та забезпечення захисту операційної системи від зловмисних застосунків.

1.2.3 Причини використання зовнішнього сховища

Однією з головних причин використання розробниками зовнішнього сховище є обмеженість простору внутрішнього сховища. Також велика кількість розробників популярного програмного забезпечення використовує зовнішнє сховище, як місце тимчасового зберігання даних при їх завантаженні з мережі Інтернет. Також існують випадки, в яких користувачі хотіли б розділити деякі свої файли (фотографії, аудіофайли, відеозаписи та інше) між декількома застосунками, або ж копіювати ці дані на інші пристрої.

1.2.4 Проведення атаки

Вразливим слід вважати застосунок, який:

- зберігає файли, що підлягають подальшій обробці, на зовнішньому сховищі, нехтуючи або втілюючи некоректно процес валідації таких файлів
- зберігає на зовнішньому сховищі виконувані файли або файли класів
- не виконує перевірку цифрового підпису на файлах, що підлягають подальшому копіювання у внутрішню пам’ять

Атака починається із знаходження вразливого застосунку та подальшого вивчення, як і для чого застосунок використовує зовнішнє сховище. В залежності від цього атака може бути розгорнута наступним чином:

1) вразливий застосунок зберігає в приватному каталозі файли для подальшої обробки: перезаписати файл таким чином, що його обробка призведе до відмови в обслуговуванні чи завчасного завершення роботи внаслідок критичної помилки
2) вразливий застосунок завантажує оновлення з серверу оновлень до зовнішнього сховища з метою подальшого встановлення: файл з

оновленням перезаписується з метою встановлення замість оновлень небажаного програмного забезпечення

Можна виділити ще один більш тривіальний вид даної вразливості: застосунок зберігає чутливу інформацію про користувача на зовнішньому сховищі.

1.3 Огляд джерел

Під час вивчення особливостей атаки «Man in the Disk» було проведено пошук та аналіз джерел, що документують основні передумови для проведення та основні етапи проведення даної атаки. З них слід виділити наступні:

- Стаття дослідників лабораторії Check Point Research [8], що містить детальний огляд атаки, її передумов, наслідків, містить відео з демонстрацією проведення атаки на декілька вразливих застосунків, приклади доповнені переліком файлів у зовнішньому сховищі, через які було здійснено атаку у кожному конкретному випадку
- Стаття більш загального характеру з ілюстраціями потоку атаки, без подробиць та конкретних прикладів, у блогу компанії Check Point [9]
- Доповідь дослідника з команди Check Point на конференції DefCon 26, що представляє собою агрегацію двох попередніх джерел, доповнену деталями та подробицями щодо особливостей функціонування операційної системи, пов’язаних з даною атакою, аналізом рекомендацій від Google по використанню зовнішнього сховища та інформацією щодо інструментів пошуку вразливості подібного типу [10]

1.3.1 Відомі вразливі застосунки

За результатами досліджень спеціалістів з Check Point, наступні застосунки виявилися вразливими до атаки «MitD»:

- Google Translate
- Yandex Translate
- Google Voice Typing
- LG Application Manager
- LG World
- Google Text-to-speech
- Xiaomi Browser

З метою отримання кращого розуміння особливостей проведення атаки розглянемо наведені вище випадки в деталях.

1.3.1.1 Кроки атаки в деталях

Незважаючи на зовсім різні цілі та засоби функціонування наведених вище застосунків, всі з них використовують зовнішнє сховище, приділяючи недостатньо уваги механізмам захисту. Таким чином, атаки всіх цих застосунків починаються з пошуку файлів, які завантажуються до зовнішнього сховища. Для цього використовують базові техніки спостереження за файлами:

- Шляхом наслідування та імплементації спеціально сконструйованого абстрактного класу для вирішення цих задач – android.os.FileObserver
- Однак попередній підхід не надасть жодної корисної інформації, якщо необхідно наглядати за приватними каталогами застосунків, для цього слід наслідувати клас TimerTask, де, в якості запланованої дії, виконати приватних каталогів за допомогою класу File

Як показали дослідження, всі названі вразливі застосунку зберігали необхідні для проведення атаки файли саме у приватних каталогах, тож другий підхід є більш ефективним.

Таким чином, зловмисний застосунок у всіх наявних сценаріях проведення атаки виконую перевірку певних каталогів (в залежності від застосунку-жертви це можуть бути як приватні каталоги, так і звичайні каталоги зовнішнього сховища) та проводити певний перезапис цільових файлів одразу після їх виявлення. Для цього йому знадобиться отримати дозвіл WRITE_EXTERNAL_STORAGE, що відноситься до категорії небезпечних, тобто повинен бути виданий лише з дозволу користувача на запит операційної системи. Так як велика кількість сучасних застосунків просить такий доступ, користувачі звикли гарантувати його майже будь-якому застосунку. Спосіб перезапису цільових файлів залежить від роботи кожного окремого застосунку-жертви.

Google Translate та Yandex Translate піддаються абсолютно ідентичному алгоритму атаки, так як вони зберігаються пакети для оффлайн перекладу в приватних каталогах на зовнішньому сховищі. Так, Google Translate зберігає пакети у своєму приватному каталогу ../Android/data/com.google.android.apps.translate/files/olpv3/v5/25/r11/, тоді як перекладач Yandex Translate зберігає пакети у свій приватний каталог ../Android/data/ru.yandex.translate/files/offline/translation/. Таким чином, зловмисний застосунок перезаписує файли словників таким чином, що це призводить до критичних помилок під час спроби завантаження додатками. Помилки виникнуть при наступній спробі використати перекладачі в режимі оффлайн перекладу, що призводить до спроби використання перезаписаних файлів бібліотеками native libraries кожного з застосунків:

- libtranslate.so у випадку Google Translate
- та libmobile-android.so

Вразливість даних застосунків полягає у тому, що впроваджений в них процес валідації файлів зовнішнього сховища не є достатнім.

Застосунок Google Voice Typing є вразливим через необачне використання власного приватного каталогу ../Android/data/com.google.android.googlequicksearchbox/files/download_cache/ для тимчасового збереження допоміжних файлів оффлайн розпізнавання мов після їх завантаження з мережі Інтернет. Цікавим цей випадок робить те, що за даними дослідників після завантаження файлів та до початку їх копіювання не виникає жодних перевірок.

Застосунок Google Text-to-speech завантажує допоміжні дані у вигляді архіву до приватного каталогу ../Android/data/com.google.android.tts/files/download_cache/, проводить перевірку цифрового підпису та розпаковує їх до внутрішньої пам’яті. В даному випадку застосунок є вразливим через те, що перевірки підпису та розпаковування не є повністю автоматичними, що надає час зловмисному застосунку для перезапису цього файлу після перевірки підпису. Результати аналогічні до описаних вище – критична помилка рівня native library бібліотеки додатку libtts_android.so.

Наступні випадки вразливих додатків об’єднує те, що вони завантажують до зовнішнього сховища виконувані файли.

Пристрої LG постачаються разом із застосунком LG Application Manager для доступу до власного сервісу цифрової дистрибуції. Даний випадок є цікавим через те, що LG, як і багато інших виробників, надають деяким своїм попередньо встановленим застосункам досить високі привілеї за замовчуванням. Так, даний застосунок відповідальний за завантаження, оновлення та встановлення програмного забезпечення на мобільних пристроях від LG. Вразливість полягає в тому, що даний застосунок завантажує виконуваний файл .apk у прихований каталог ../.dwnld/, після проводить встановлення завантаженого застосунку. У

цьому випадку зловмисний застосунок одразу після завантаження перезаписує .apk файл вмістом .apk будь-якого іншого потенційно небезпечного застосунку.

Схожа ситуація відбувається з іншим попередньо встановленим застосунком на пристроях LG – LG World, що відповідає за відображення оформлення графічного інтерфейсу, так званих тем. Застосунок завантажує власне оновлення, зберігаючи його перед встановленням до прихованого каталогу ../LGWorld/.Temp/. Алгоритм дій зловмисного застосунку аналогічний до наведеного вище.

Попередньо встановлений застосунок Xiaomi Browser завантажує власне оновлення до зовнішнього сховища, проводить перевірку хеш-суми SHA-1 та встановлює оновлення. Ситуація певним чином схожа на Google Text-to-speech. Справа в тому, що встановлення і верифікація – це два окремі процеси, між якими існує пауза наступного характеру. Рисунок 1.2 ілюструє її.

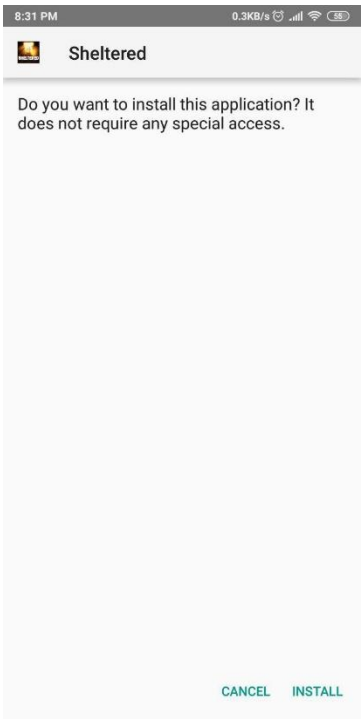

Рисунок 1.2 – Запит на встановлення

Тобто процес оновлення вимагає від користувача підтвердження. Цього часу цілком достатньо для перезапису вмісту даного apk-файлу вмістом apk-файлу потенційно небезпечного застосунку.

1.3.1.2 Випадки, виявлені після дослідження Check Point

Наступне викриття даної вразливості трапилося через три дні після публікації статті у блогу компанії Check Point, 15 серпня 2018 року [11]. Вразливість була виявлена спеціалістами Google у застосунку Fortnite Installer (com.epicgames.portal). Специфіка проблеми подібна до випадків, досліджених спеціалістами Check Point – застосунок завантажує apk-файл (com.epicgames.fortnite) для встановлення мобільної гри у власний приватний каталог на зовнішньому сховищі. Далі Fortnite Installer проводить валідацію та встановлення завантаженого застосунку. Принцип атаки аналогічний описаному у попередньому підрозділі – зловмисний застосунок перезаписує файл, що підлягає встановленню.

З особливостей слід відзначити використання застосунком встановлення спеціального API, що надається розробникам операційною системою Android на пристроях Samsung, Galaxy Apps API. Даний прикладний програмний інтерфейс дозволяє тихе встановлення. Хоча цей інтерфейс проводить перевірку підписів перед виконанням встановлення (перевіряє, щоб ім’я встановлюваного пакета співпадало з іменем пакета застосунка, який надіслав запит на встановлення). Таким чином, та пристроях Samsung можливо провести тихе встановлення небажаного застосунку завдяки їх власним програмним інтерфейсам.

Розробник, EpicGames, обрав подібний спосіб поширення продукту через достатньо високий процент стягнень з прибутків застосунків, поширюваних через Google Play Store – 30%. Відмова від поширень через перевірену роками

інфраструктуру зобов’язує розробників самостійно дбати про створення власної надійної інфраструктури.

1.3.2 Запропоновані рішення

У всіх вищезазначених джерелах, пов’язаних з дослідженням атаки, зазначається існування переліку рекомендацій по роботі з зовнішнім сховищем від Google, що надаються у циклі статей «Android developer training articles». У доповіді на конференції DefCon 26 були перелічені наступні рекомендації:

- Необхідно проводити валідацію при використанні даних з зовнішнього сховища
- Ніколи не зберігати виконувані файли чи файли класів на зовнішньому сховищі
- Файли на зовнішньому сховищі повинні мати цифровий підпис та проходити його криптографічну перевірку перед наступним їх завантаженням

На прикладі вразливих застосунків було показано, що навіть великі компанії на ринку Android-пристроїв не дотримуються даних рекомендацій або реалізують їх недостатньо ефективно. Окрім цього, ситуації Google Text-to-speech та Xiaomi Browser показали, що навіть виконання рекомендацій не є запорукою створення безпечного застосунку. Також велика частина помилок при використанні зовнішнього сховища виникає через необізнаність розробників чи байдужість розробників.

Висновки до розділу 1

Під час роботи над даним розділом було зібрано інформацію про особливості функціонування операційної системи Android, на яких базується експлуатація вразливості, що призводить до атаки «MItD». На основі розглянутих джерел було проаналізовано алгоритм проведення атаки та її можливі наслідки. Отримані знання будуть враховані при розгляді рішень подібних вразливостей у наступному розділі.

2 АНАЛІЗ ПІДХОДІВ ДО ВИРІШЕННЯ ПОДІБНИХ ПРОБЛЕМ

2.1 Загальний підхід

2.1.1 Огляд існуючих рішень

Незважаючи на те, що вразливість була відкрита у серпні 2018 року, згадувань про неї на офіційних ресурсах, присвячених розробці застосунків під Android (Developer Android) та розробці платформ на основі Android (Source Android), не було знайдено згадувань про цей спосіб атаки та пов’язану з ним вразливість зовнішнього сховища. Також не було знайдено коментарів Google щодо цієї вразливості. Таким чином, на даний момент єдиним рішенням даної проблеми є свідомий підхід розробників застосунків до використання зовнішнього сховища.

З архівного листування між спеціалістами Google та EpicGames, що згадувалося у попередньому розділі, можна дізнатися підхід до вирішення. Розробник відмовився від завантаження виконуваних файлів до зовнішнього сховища, натомість завантаження у всіх наступних версіях їх застосунку відбувається одразу до внутрішнього сховища, що єдиним безпечним способом завантаження виконуваних файлів.

У випадку Google Translate рішення було досить простим – допоміжні файли для оффлайн-роботи перекладача тепер не зберігаються на зовнішньому сховищі. Рисунок 2.1 підтверджує це.

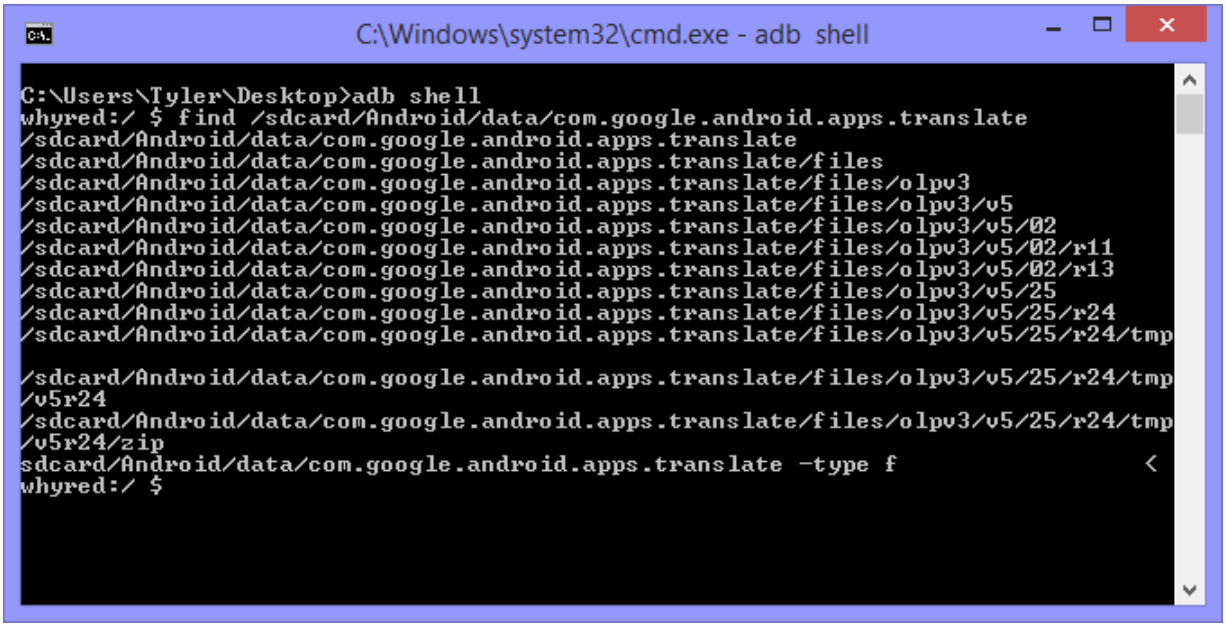

Рисунок 2.1 – Перевірка каталогів застосунку

2.1.2 Визначення сфер відповідальності

Як зазначено у попередньому розділі, рекомендації від Google для розробників прикладного програмного забезпечення є необхідними, але недостатніми для створення безпечних застосунків. Хоча валідація при обробці файлів із зовнішнього сховища є необхідною, приклади атак на застосунки Xiaomi Browser, Google Text-to-speech, а також виявлена вразливість у застосунку завантаження гри Fortnite підтверджують коментар спікера від дослідників вразливості вищезгаданої конференції DefCon 26 – однієї валідації недостатньо для захисту від перезапису даних. Це, а також недостатня обізнаність розробників навіть у великих компаніях, підтверджує необхідність внесення змін у функціонування зовнішнього сховища або ж до фреймворку Android чи хоча б деяких окремих прикладних програмних інтерфейсів, що пов’язані з зовнішнім сховищем.

Для розробки ефективних рекомендацій слід зрозуміти, на якому рівні ці рекомендації потрібно впроваджувати. Також це допоможе зрозуміти, хто є відповідальним за втілення подібних рекомендацій.

Як згадувалося у попередньому розділі, платформа Android потребує налаштувань та підстроювань під кожний конкретний пристрій, тож Google є не єдиним розробником. Рисунок 2.2 зображує дещо спрощений стек платформи.

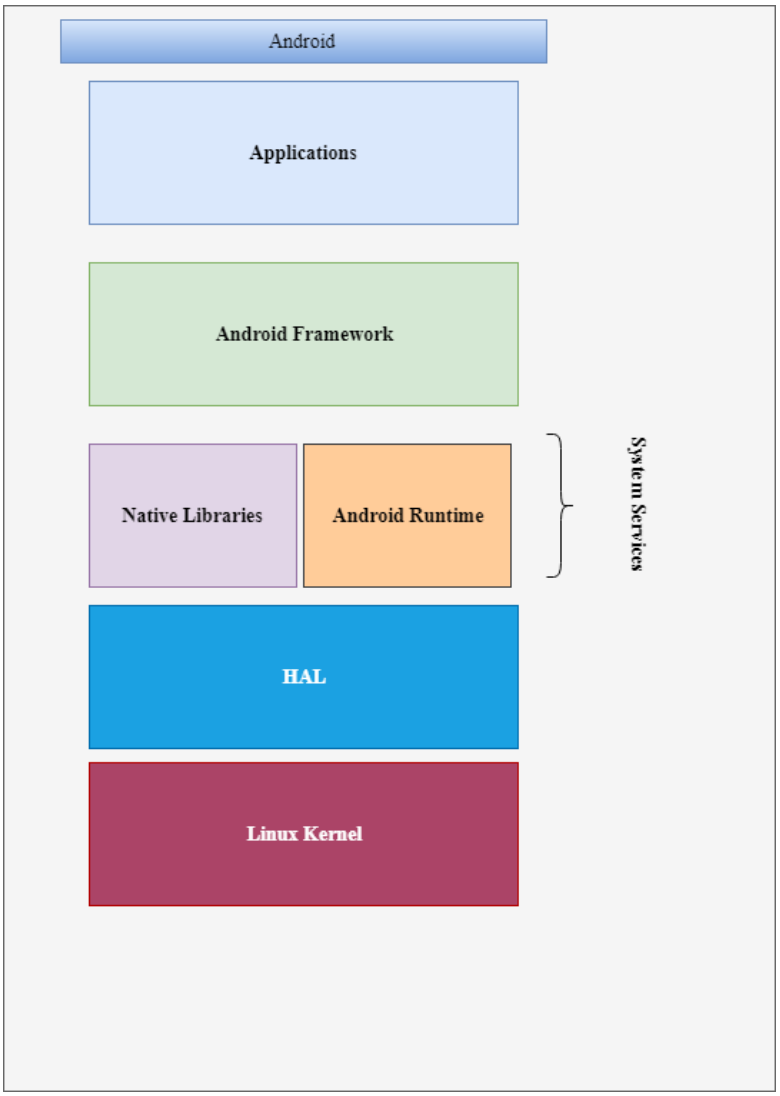

Рисунок 2.2 – Спрощений стек Android

У більш загальному представленні архітектури платформи Android розглядається прикладний програмний інтерфейс для розробки застосунків – рівень Android Framework – він, в свою чергу, реалізований на так званих

системних сервісах (System Services), вони реалізують логіку програмного інтерфейсу. За ними йде апаратний рівень абстракції та модифіковане ядро Linux.

Google відповідає за розробку програмного інтерфейсу, системних сервісів та модифікацію ядра. Виробники пристроїв виникають у цій схемі для розробки рівня апаратної абстракції для своїх продуктів, конфігурування стандартної версії Android, що надана Google. Рисунок 2.3 зображує загальну схему розробки платформи.

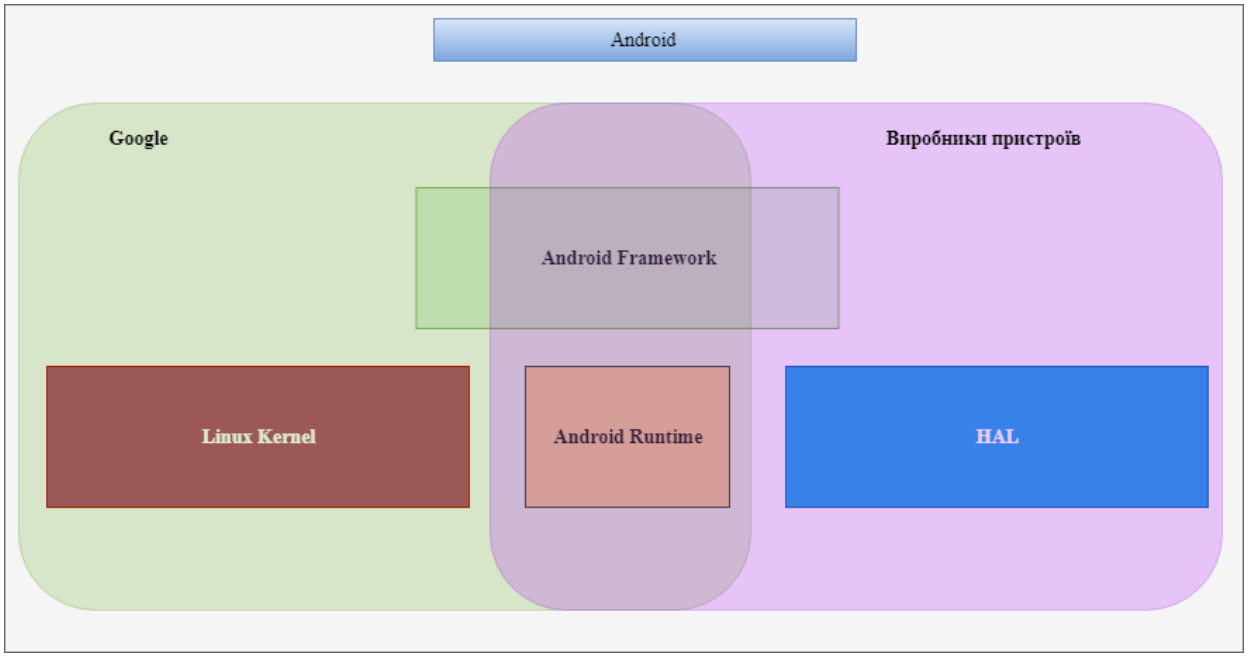

Рисунок 2.3 – Загальна схема розробки платформи

З рисунку видно, що сфери відповідальності за розробку системних сервісів та прикладного програмного інтерфейсу перекриваються. Google надає «чисту» версії інтерфейсу та системних сервісів, виробники мають можливість змінювати принцип роботи цих компонент та розширювати їх власними наробками. Так, відомо, що у деяких виробників – Samsung, HTC, Xiaomi – одні й ті самі методи програмного інтерфейсу працюють інакше, ніж це задано Google в стандартній версії.

Можна виділити декілька підходів усунення даної вразливості:

- Внесення змін на рівні архітектури системи

- Внесення змін на рівні прикладного програмного інтерфейсу з метою усунення можливості створювати застосунки, вразливі до даного типу атаки
- Внесення змін у керування доступом у зовнішньому сховищі

Наведені напрямки не є взаємовиключними і, якщо це можливо, повинні використовуватися у поєднанні задля створення комплексного рішення. Дослідники з Check Point у своїх роботах висловлюють думку, що для максимально ефективного вирішення даної проблеми зміни повинні бути впроваджені на рівні операційної системи з боку Google. В такому випадку очікується внесення змін на декількох рівнях одночасно.

Існує інша точка зору. Так як зовнішнє сховище функціонувало так з перших версій платформи, Google навряд чи буде вносити кардинальні зміни у функціонування сховища и залишить цю ситуацію у теперішньому стані.

Метою є формування можливих підходів усунення вразливості на основі вже існуючих рішень в інших підсистемах платформи Android та нещодавно розкритої вразливості в macOS, які можливо впровадити на стороні виробників пристроїв.

2.1.3 Постановка завдання

На основі матеріалу, викладеного у попередньому розділі, мета роботи вимагає знайти рішення для подолання наступних проблем:

- Неефективність або відсутність валідації при обробці даних із зовнішнього сховища
- Завантаження виконуваних файлів до зовнішнього сховища

Задля досягнення мети роботи потрібно розробити рекомендації щодо:

- Вирішення проблеми неефективності валідації в якості захисту від перезапису
- Унеможливлення завантаження виконуваних файлів до зовнішнього сховища

2.2 Аналіз існуючих підходів

2.2.1 Відмова від використання вразливого сервісу

Коли ми говоримо, що те чи інше прикладне програмне забезпечення вразливе, то першим можливим рішенням є припинення використання вразливого застосунку. У даному випадку, відмовитися від використання зовнішнього сховище не є можливим, оскільки таке його функціонування стало невід'ємною частиною користувацького досвіду. Користувачі звикли до можливості до можливості простого обміну будь-якими файлами між зовнішнім сховищем пристрою та комп'ютером. Мова йде про часткову відмову від використання зовнішнього сховища розробниками у своїх застосунках.

У наявних прикладах рішень, що були використані Google у Google Translate та EpicGames у застосунку для завантаження Fortnite, розробники відмовилися від тимчасового чи довгострокового використання зовнішнього сховища.

Подібний підхід можна розглядати, як тимчасове рішення наявної проблеми. Нещодавно було розголошено схожу певним чином до даної вразливість в операційних системах комп'ютерів Apple [12], яка також призводить до обходження sandbox, вбудованого в macOS.

Вразливість дозволяє обходити Gatekeeper – механізм, що несе відповідальність за захист операційної системи від виконання ненадійних застосунків шляхом перевірки наявності виданого Apple цифрового сертифікату.

Вразливість полягає в тому, що механізм Gatekeeper розглядає зовнішні диски та мережі як безпечні, що в поєднанні з іншими легітимними функціями операційної системи може призводити до запуску ненадійних застосунків без участі користувача. Інструмент автоматичного монтування autofs дисків дозволяє також виконати монтування мережі. Інша складова вразливості – можливість zip-архіву містити символічні посилання на будь-які місця та те, що програмне забезпечення macOS, відповідальне за розпаковування архівів, не виконує жодних перевірок цих посилань перед їх створенням. У звіті приведено наступний приклад: зловмисник створює zip-архів, що містить спеціальне символічне посилання на точку автоматичного монтування до керованої ним мережі, і відправляю жертві; жертва проводить розпаковування та переходить за символічним посиланням, що призводить до можливості виконання будь-якого контрольованого зловмисником файлу у цій мережі без попередження. Gatekeeper при цьому буде вважати такий сценарій безпечним.

Ця вразливість була досліджена в лютому 2019 року, дослідник, Filippo Cavallarin, повідомив представників компанії про неї. Представники Apple пообіцяли закрити дану вразливість до 15 травня. Так як у багатьох програмах ліквідацій вразливостей надається термін у 90 діб з моменту інформування розробника на ліквідацію, Filippo Cavallarin розкрив вразливість. Станом на сьогодні дана вразливість досі наявна в операційній системі. В якості тимчасового рішення дослідник запропонував відключення певного функціоналу утиліти автоматичного монтування.

Описана вразливість схожа на досліджувану в роботі тим, що за рахунок не достатньої жорсткості політики безпеки щодо зовнішніх монтованих сховищ у стандартному програмному забезпеченні системи дозволяє уникнути механізму sandbox, що наражає користувачів на сильну небезпеку. Також простежується певна схожість ситуації з ліквідацією вразливості – за проміжок часу майже у рік так релізом однієї повноцінної версії, Android 9, та бета-версій Android Q у Google

ніяк не відреагували на відому вразливість, полишивши її на розробників застосунків та користувачів.

Як показує практика, рішення, яке призводить до погіршення користувацького досвіду не слід розглядати як повноцінне, але безпека даних користувача є важливішою, тому слід взяти даний підхід усунення вразливості до уваги під час розробки рекомендацій.

2.2.2 Обмежений доступ до каталогів

Враховуючи, специфіку роботи застосунків, велика кількість з них потребує доступу лише до певних конкретних каталогів. Так, наприклад, більшість аудіоплеєрів потребує доступу лише до каталогу Music. До Android 7 наявні методи отримання доступу до зовнішнього сховища не сприяли зручному отриманню доступу лише до певних обраних каталогів:

- запит на отримання небезпечних дозволів READ_EXTERNAL_STORAGE та WRITE_EXTERNAL_STORAGE, що призводило до отримання доступу до всіх каталогів зовнішнього сховища, що не завжди відповідає потребам застосунку
- іншим способом отримання доступу до зовнішнього сховища було використання Storage Access Framework, що надає користувачу можливість обрати необхідний каталог через графічний інтерфейс, що не є зручним та надійним способом у тих випадках, коли застосунок завжди потребує доступу до одного каталогу

В Android 7 була надана можливість отримувати обмежений доступ каталогів (Scoped Directory Access), що надає можливість більш гранулярного контролю дозволів застосунку і сприяє логічному розвитку принципу найменших привілеїв в екосистемі Android – розроблюваний додаток повинен мати лише ті дозволи, що є справді необхідними для його роботи.

Цей програмний інтерфейс надає можливість отримувати доступ до конкретного каталогу за відомою назвою чи визначати необхідний каталог за деталізованою інформацією про шуканий файл, що надається методу StorageManager.getStorageVolume(File) у вигляді параметру типу File. Якщо користувач відповідає згодою, то для доступу буде надано об’єкт типу URI (уніфікований ідентифікатор ресурсів).

Для того, щоб не запитувати в користувача подібний дозвіл кожного разу слід зберегти отриманий ідентифікатор ресурсів. Про вирішення питань менеджменту таких ідентифікаторів вже подбали спеціалісти Google – за допомогою метода ContentResolver.takePersistableUriPermission() дозволяє отримати застосунку гранулярне дозвіл на доступ лише до конкретного каталогу на зовнішньому сховищі та позбавляє розробника додаткових ризиків при зберіганні та обробці подібних ідентифікаторів.

2.2.3 Обмежений доступ до сховища

У бета-тестування Android Q ідея із гранулярним доступом до каталогів просунулася ще на крок вперед [13]. Два основні дозволи для доступу до зовнішнього сховища, що спричинили появу розглянутого типу атак – READ_EXTERNAL_STORAGE та WRITE_EXTERNAL_STORAGE – залишилися без змін, але для надання користувачу більшого контролю щодо доступу застосунків до зовнішнього сховища використовується система доступу sandboxed view (система доступу «обмеженого пісочницею»). Тобто будь-які застосунки при написанні яких Android Q обирається за основну цільову версію платформи будуть отримувати замість прямого доступу (доступу, який розглядався протягом всієї роботи) обмежений.

При такому підході застосунок може зберігати дані, призначені для внутрішнього використання у приватному каталогу на зовнішньому сховищі, а

також завжди має доступ до створюваних файлів як у межах власного приватного каталогу, так і створюваних за його межами. При цьому застосунок може доступатися до файлів, що створили інші застосунки лише у тому випадку, якщо ці файли знаходяться в одному з наступних каталогів:

- MediaStore.Images
- MediaStore.Video
- MediaStore.Audio

Якщо ж у застосунка виникне потреба отримати доступ до файлів інших застосунків з інших каталогів, то даний застосунок повинен виконати звернення через Storage Access Framework, що надає користувачу відповідний графічний інтерфейс для вибору необхідного файлу. Таблиця 2.1 демонструє описаний принцип отримання доступу.

Таблиця 2.1 – Принцип отримання доступу

Місцезнаходження файлу	Необхідний дозвіл	Метод отримання доступу	Чи видаляються файли після видалення застосунку
Приватний каталог застосунку	Ніякий	getExternalFilesDir()	Так
Медіа-файли	READ_EXTERNAL_STORAGE	MediaStore	Ні
Downloads (каталог завантажень)	Ніякий	Storage Access Framework	Ні

Даний механізм може бути використано не лише у Java/Kotlin коді, але й при розробці нативного коду на C/C++, для цього необхідно отримати через рівень Java/Kotlin об’єкт класу MediaStore та передати його дескриптор до нативного коду.

Слід зазначити, що у попередніх версіях бета-релізів були наявні окремі дозволи на читання різних типів медіа-файлів: READ_MEDIA_IMAGES, READ_MEDIA_AUDIO та READ_MEDIA_VIDEO. Тепер вони відмічені, як

застарілі. З цією версією платформа зробить контроль над зовнішнім сховищем більш гранульованим, що є вкрай важливим для подолання наявної вразливості, але, виходячи з того, як стрімко проходить розвиток у межах тільки бета-тестувань, можна припустити, що підхід до рішення чи програмний інтерфейс момент на релізі можуть бути кардинально іншими.

Для тих випадків, коли передбачається, що застосунок повинен мати доступ до багатьох файлів чи навіть підкаталогів передбачено спеціальний об’єкт міжпроцессного зв’язку ACTION_OPEN_DOCUMENT_TREE, який надає користувачу графічний інтерфейс для вибору конкретного дерева каталогів до якого застосунку буде надано доступ. Отримавши такий доступ застосунок зможе змінювати тільки файли обраного користувачем каталогу чи його підкаталогів.

Також заявлено, що у майбутніх релізах, коли застосунок з обмеженим через sandboxed view доступом до зовнішнього сховища буде запитувати небезпечні дозволи, пов’язані зі сховищем, то він буде отримувати тільки доступ до свого приватного каталогу чи до вищезгаданих каталогів мультимедіа. При спробі отримання доступу через звичайний необмежений механізм (file-system view) застосунок отримає виключення типу FileNotFoundException або помилку нативного коду.

Також в бета-тесті Android Q були впроваджено флаг IS_PENDING, який дозволяє застосунку, що володіє певним мультимедіа-файлом, відхиляти запити на доступ до цього файлу від інших застосунків на час виконання ним (володільцем) певних дій над цим файлом.

Висновки до розділу 2

Під час роботи над даним розділом було зібрано та проаналізовано інформацію про спосіб усунення вразливості у застосунках Google Translate та Fortnite Downloader, також розглянуто рішення структурно подібної вразливості у

системному застосунку macOS. В усіх розглянутих випадках розробники відмовлялися від використання потенційно небезпечного функціоналу. Випадок macOS є особливо важливим з організаційної точки зору, так як і розробники з Apple, і розробники з Google, знаючи про наявну вразливість своїх платформ не надали рішення протягом трьох місяців. Так, Filippo Cavallarin пропонує тимчасове вирішення питання вразливості до поки офіційний розробник не надасть комплексного рішення. Рекомендації дослідника також базуються на відмові від використання вразливого функціоналу. Отримана в результаті аналізу рішень, стане основою вибору підходу під час формування механізмів захисту у наступному розділі.

Також була проаналізована документація нещодавно випущеної Google бета-версії Android Q, з якої слідує, що компанія почала впроваджувати комплексне рішення даної вразливості одразу на декількох рівнях – зміни політики безпеки, прикладного програмного інтерфейсу та надання детальної документації. Так як дане рішення перебуває тільки на етапі бета-тестування, його кінцевий варіант може кардинально змінитися та буде проходити доопрацювання. Базуючись на цьому, а також на особливостях постачання на ринок нових версій операційної системи виробниками пристроїв, можна зробити висновки, що більша половина існуючих на даний момент пристроїв під керуванням даної операційної системи, буде працювати на версіях, вразливих до даної атаки, як мінімум до середени осені 2019 року. Також слід зауважити, що певна частина пристроїв (близько половини всіх бюджетних смартфонів та планшетів) взагалі не отримають оновлення до Android Q. Це підтверджує актуальність пошуку тимчасового та відносно простого у впровадженні рішення навіть при наявності потенційного рішення від основного розробника платформи.

3 ФОРМУВАННЯ МЕХАНІЗМУ ЗАХИСТУ

3.1 Вибір підходу

Враховуючи поставлені у розділі 2.1.3 завдання, а також те, що експлуатація даної вразливості стає можливою завдяки неналежному використанню зовнішнього сховища розробниками, в якості основного підходу рішення проблеми було обрано відмову від використання вразливого сервісу – відмовитися від завантаження файлів із мережі Інтернет до зовнішнього сховища. Таким чином можна вирішити проблему неефективності валідації файлів із зовнішнього сховища та унеможливити завантаження виконуваних файлів до зовнішнього сховища.

Обраний підхід вимагає програмне рішення для завантаження файлів із мережі минуючи зовнішнє сховище та проводячи при цьому обов’язкову валідацію, як це рекомендує керівництво для розробників від Google.

Слід враховувати, що виробники можуть реалізовувати таке рішення різними шляхами чи не реалізовувати взагалі, а так як більшість застосунків розробляються під стандартну версію операційної системи Android, то механізм впровадження рішення повинен підштовхувати розробників до його використання, але не нашкодити сумісності застосунків, що з тих чи інших причин не використовують дане рішення.

3.2 Опис механізму

Згідно наведених вище вимог до програмної реалізації, що накладає обраний підхід, було запропоновано наступну схему:

- розробити системний додаток, що буде приймати запити на завантаження файлів із мережі та проводити автоматичну валідацію файлів за механізмом, що було обрано розробником застосунку
- для спонукання розробників до використання даного рішення слід впровадити зміни ра рівні прикладного програмного інтерфейсу з метою інформування про небезпечність прямого завантаження до зовнішнього сховища

Загальна схема такого рішення та подана на рисунку 3.1.

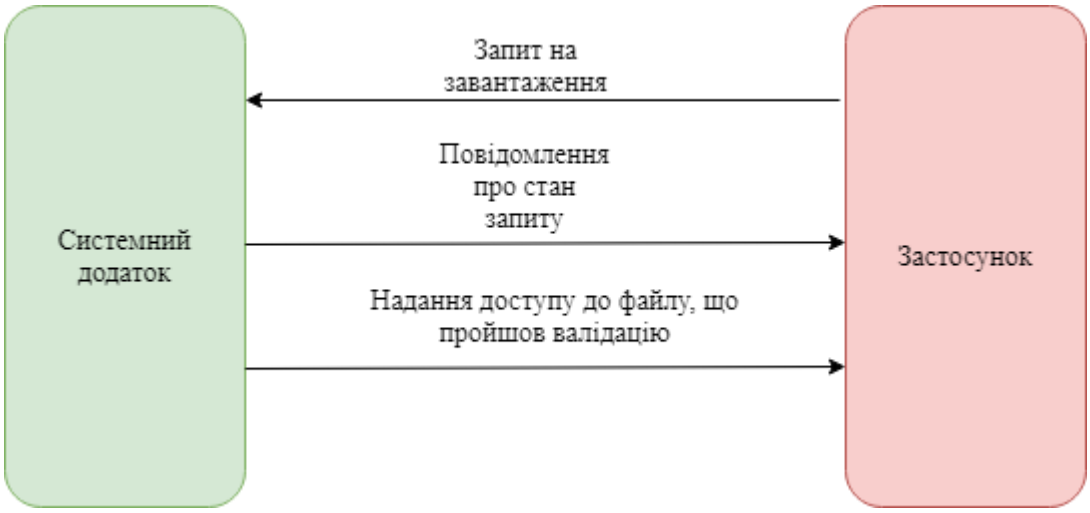

Рисунок 3.1 – Загальна схема рішення

3.3 Реалізація

3.3.1 Структура застосунків

Для подальшого опису програмної реалізації слід детальніше розглянути структуру застосунків в операційній системі Android.

Для застосунків даної платформи характерна відсутність однієї основної точки входу – main-функції чи Main-класу – замість цього застосунок складається з багатьох компонент, кожна з яких може слугувати точкою входу. Виділяють наступні типи компонент:

- Activity – представляє собою один «логічний екран» застосунку та надає графічний інтерфейс користувачу для взаємодії
- Service – сервіси, точки входу без графічного інтерфейсу, що представляють фонові довгострокові процеси
- Broadcast Receiver – отримувач широкомовних повідомлень від системи, дозволяє реагувати на системні повідомлення навіть коли основні компоненти застосунку неактивні
- Content Provider – провайдер контенту, дозволяє надання та контроль доступу до даних даного застосунку іншим застосункам

Так як додаток повинен виконувати тільки завантаження файлів та їх валідацію, йому не потрібен графічний інтерфейс. Враховуючи це, було обрано реалізовувати рішення у вигляді сервісу. В операційній системі Android сервіси поділяють на декілька типів:

- Foreground – сервіси, дія яких помітна для користувача (програвання музики, наприклад), такий сервіс продовжує роботу навіть коли користувач не взаємодіє із застосунком
- Background – фонові сервіси, виконуть операції, які не є безпосередньо помітними для користувача
- Bound – сервіси, які застосунок пов’язує викликом методу bindService(), такі сервіси надають клієнт-серверний інтерфейс, що надає можливість взаємодіяти компонентам застосунку із пов’язаним сервісом та підтримувати міжпроцесний зв’язок, до того ж такий тип сервісів працює доти, доки цього потребує компонент, що виконав прив’язку сервісу (такий сервіс має найнижчий пріоритет на зупинення системою у випадку нестачі ресурсів)

Враховуючи описані та проілюстровані вимоги до сервісу, було обрано створення пов’язаного сервісу. Це дозволяє уникнути небажаного зупинення

сервісу операційною системою та підтримувати необхідний міжпроцесний зв’язок.

На даний момент міжпроцесний зв’язок у пов’язаних сервісах можливо реалізувати у три способи:

- У випадку, коли сервіс є приватним лише для даного застосунку, можна використати клас Binder, що надасть змогу напряму викликати методи, визначені при наслідувані даного класу або напряму у класі сервісу Service
- У випадку, коли сервіс є окремим додатком и може бути пов’язаний різними застосунками, міжпроцесний зв’язок можна реалізувати через клас Messenger, також цей підхід дозволить створити зворотній зв’язок сервісу-серверу з клієнтом-застосунком, також клас Messneger бере на себе роль керування порядком прийому та обробки повідомлень від пов’язаних застосунків, що значно спрощує реалізацію зв’язку та зменшує кількість критичних помилок, що можуть бути внесені на етапі розробки
- Існує другий спосіб реалізувати зв’язок для попереднього випадку – Android Interface Defining Language (AIDL, мова визначення інтерфейсів Android), що розбирає об’єкти на примітивні типи, зрозумілі операційній системі, яка бере на себе передачу їх цільовому процесу, більш складний спосіб, що дозволяє самостійно визначити порядок обробки повідомлень, клас Messenger засновано на цьому підході

Потреба існування сервісу, як окремого системного додатку, а також необхідність двостороннього зв’язку, зумовила вибір класу Messenger для реалізації зв’язку.

В якості валідації було вирішено зупинитися на перевірці контрольних сум за алгоритмами, обраним розробником, оскільки цього достатньо в рамках перевірки концепції.

Так як описаний сервіс повинен бути реалізований окремим додатком, то він має власний приватний каталог у внутрішньому сховищі, куди буде проводити завантаження запитуваних файлів та проводити їх валідацію. Так як доступ до цього каталогу має лише даний сервіс, то завантажений та перевірений файл можливо надати пов’язаному застосунку за допомогою компоненту Content Provider.

Рисунок 4.2 зображує схему роботи запропонованого рішення у термінах програмних компонент платформи Android:

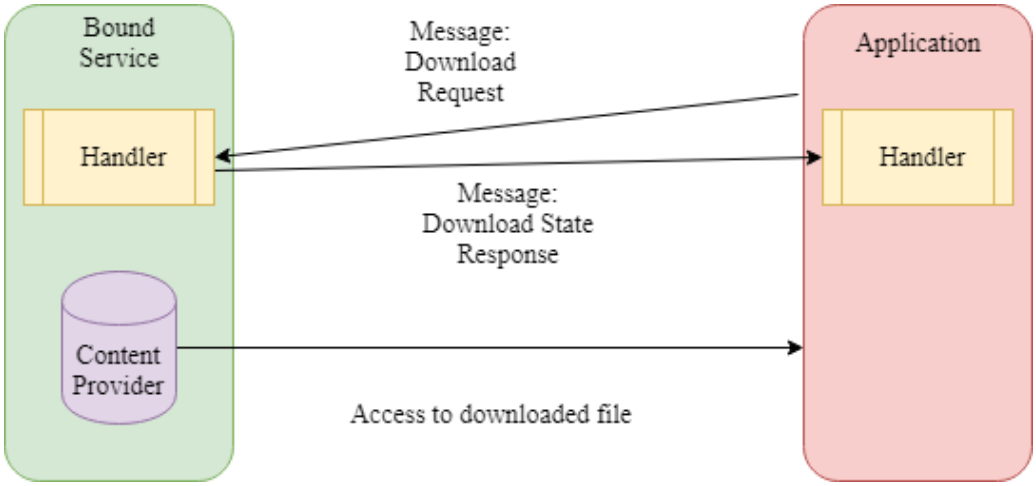

Рисунок 3.2 – Схема в термінах програмних компонент Android

3.3.2 Складнощі на етапі реалізації концепції

Перша складність, що виникла під час реалізації розглянутого додатку полягає в тому, що не існує єдиного способу завантаження файлів, так як різні web-сервіси надають доступ у різні способи – подекуди достатньо мати звичайне посилання, інші ж сервіси потребують використання спеціальних запитів від їх власних прикладних програмних інтерфейсів, що виконують комплексну роботу з

валідації користувача та розмежування доступу. Таким чином, сервіс повинен надати можливість розробникам застосунків описати процес завантаження і надати створену функцію через міжпроцесний зв’язок сервісу для подальшого виконання. Необхідність передати метод в якості параметру у багатьох мовах реалізована за принципом функціонального вказівника у C++. Так, мова C# надає повний набір можливостей функціонального вказівника, надаючи при цьому надійні механізми безпеки, через клас Delegate.

Відсутність подібного рішення на рівні стандартних бібліотек мови Java ускладнює реалізацію. На ранніх етапах розробки для вирішення цієї проблеми було вирішено розробити абстрактний клас або інтерфейс, подібний до інтерфейсу програмного інтерфейсу Android SDK – View.OnClickListener. Даний інтерфейс містить лише одну сигнатуру:

abstract void onClick(View v)

Головна мета такого методу надати розробнику можливість реалізувати відповідь певного елементу графічного інтерфейсу на його активацію користувачем. Метод приймає єдиний параметр, об’єкт класу View, який виконує роль контейнера для даного інтерактивного елемента. Таким чином, було створено абстрактний клас Downloadable, який виконував роль контейнера для параметрів (назва завантажуваного файлу та назва алгоритму знаходження контрольних сум) та визначав сигнатуру методу для реалізації у класах-нащадках, подібну для інтерфейсу View.OnClickListener:

public abstract void download(File pathOnDevice)

Даний клас було винесено в окрему бібліотеку, яку розробники застосунків повинні були додати до свого проекту. На цьому етапі передбачалося, що задля використання сервісу безпечного завантаження розробники повинні будуть реалізувати метод download(File), помістивши туди всю необхідну логіку завантаження файлу з мережі, а для зберігання звертатися до параметру, який буде наданий сервісом при виклику. Основна робота по окремого потоку

виконання та валідації покладалася на сервіс, що потенційно полегшувало інтеграцію сервісу у розробку застосунків та спрощувало ознайомлення з сервісом розробників.

Однак під час тестування виникла помилка, стек викликів якої наведено на рисунку 4.3.

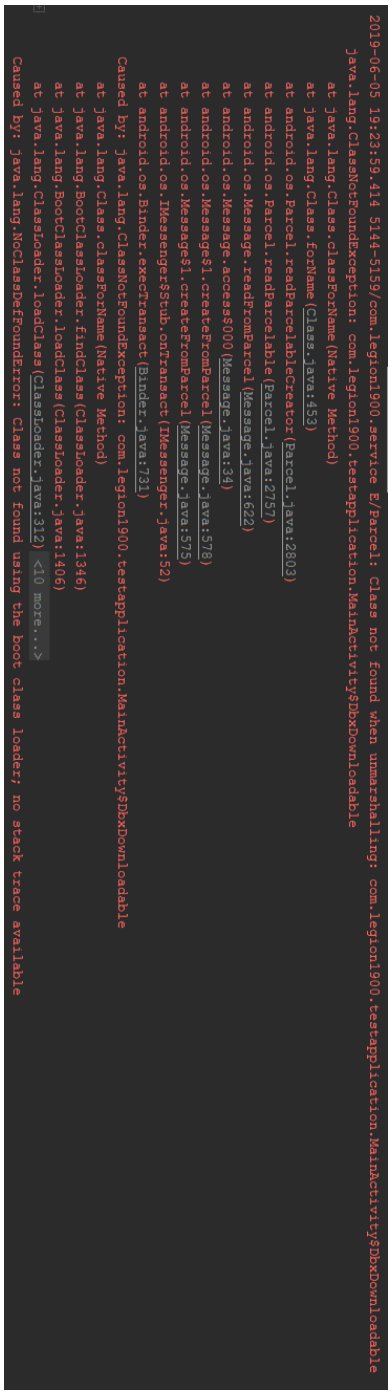

Рисунок 3.3 – Стек викликів

Виключення ClassNotFoundException говорить про те, що застосунок не знайшов байт-код означеного класу. На етапі проектування було допущено хибне трактування поняття Bound Service з типом зв’язку Messenger, було зроблено висновок, що такий сервіс може бути повністю відокремлений від застосунку, що виконав прив’язування до сервісу. З метою усунення залежності між реалізацією сервісу-додатку та застосунками було створено абстрактний клас Downloadable, до типу якого виконувалося приведення об’єкту класу-нащадку із пакету застосунку. При цьому до уваги не було взято той факт, що Dalvik VM, як і будь-яка реалізація JVM, буде враховувати фактичний тип об’єкту замість типу посилання на нього, що призведе до помилок при спробі завантажити клас, невідомий сервісу-застосунку.

3.3.3 Вирішення ускладнень

Для вирішення ускладнень було вирішено змінити підхід до загальної будови рішення. Спроектоване на попередньому етапі рішення складалося з трьох компонент:

- Сервіс-додаток для безпечного завантаження
- Бібліотека, що містить необхідні для зв’язку із сервісом класи
- Застосунок

Рисунок 4.4 ілюструє зв’язок між цими модулями, суцільною лінією позначено перетворення необхідних даних у форму параметрів та визначення метода download(File) у класі-нащадку Downloadable із бібліотеки допоміжних класів, пунктиром позначено подальшу спробу обробки об’єкту класу Downloadable, отриманого від застосунку.

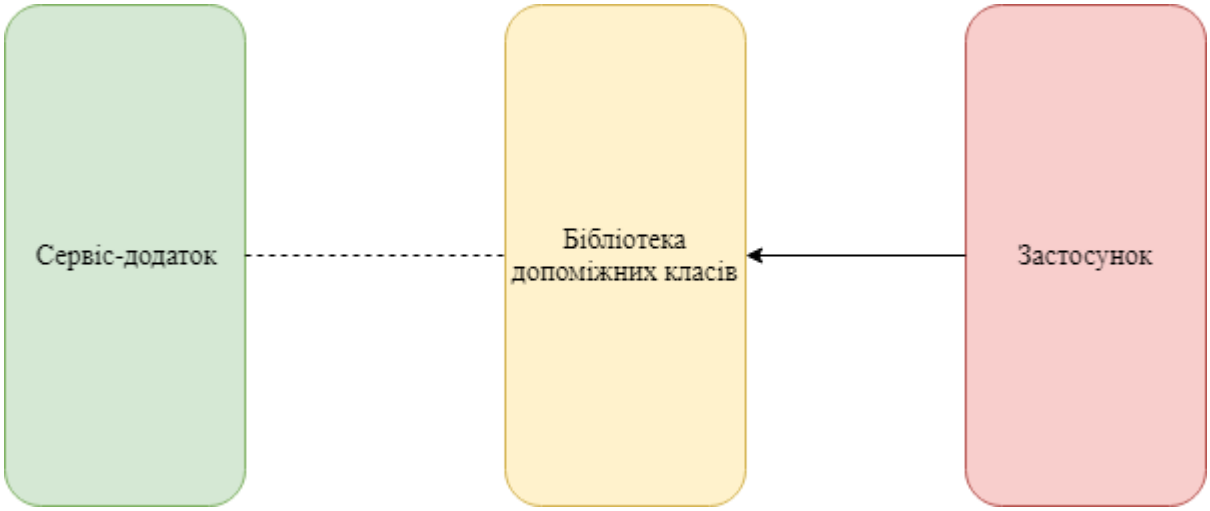

Рисунок 3.4 – Зв’язок між модулями

Було вирішено дотримуватися концепції пов’язаного сервісу для даного додатку, оскільки головні передумови вибору не змінилися, однак, була змінено загальний підхід до рішення, тепер пов’язаний сервіс повинен бути частиною застосунку, але при цьому не втратити простоти реалізації розробниками застосунків. Логіку сервісу було вирішено винести в абстрактний клас, єдиний абстрактний метод якого вимагає від розробників застосунків реалізації алгоритму завантаження файлів. Таким чином, основна логіка сервісу залишилася без змін, відповіддю на необхідність використання функціональних вказівників став абстрактний метод цього класу:

abstract protected void download(File pathOnDevice, String downloadFrom)

UML-діаграму даного модуля подано на рисунку 3.5.

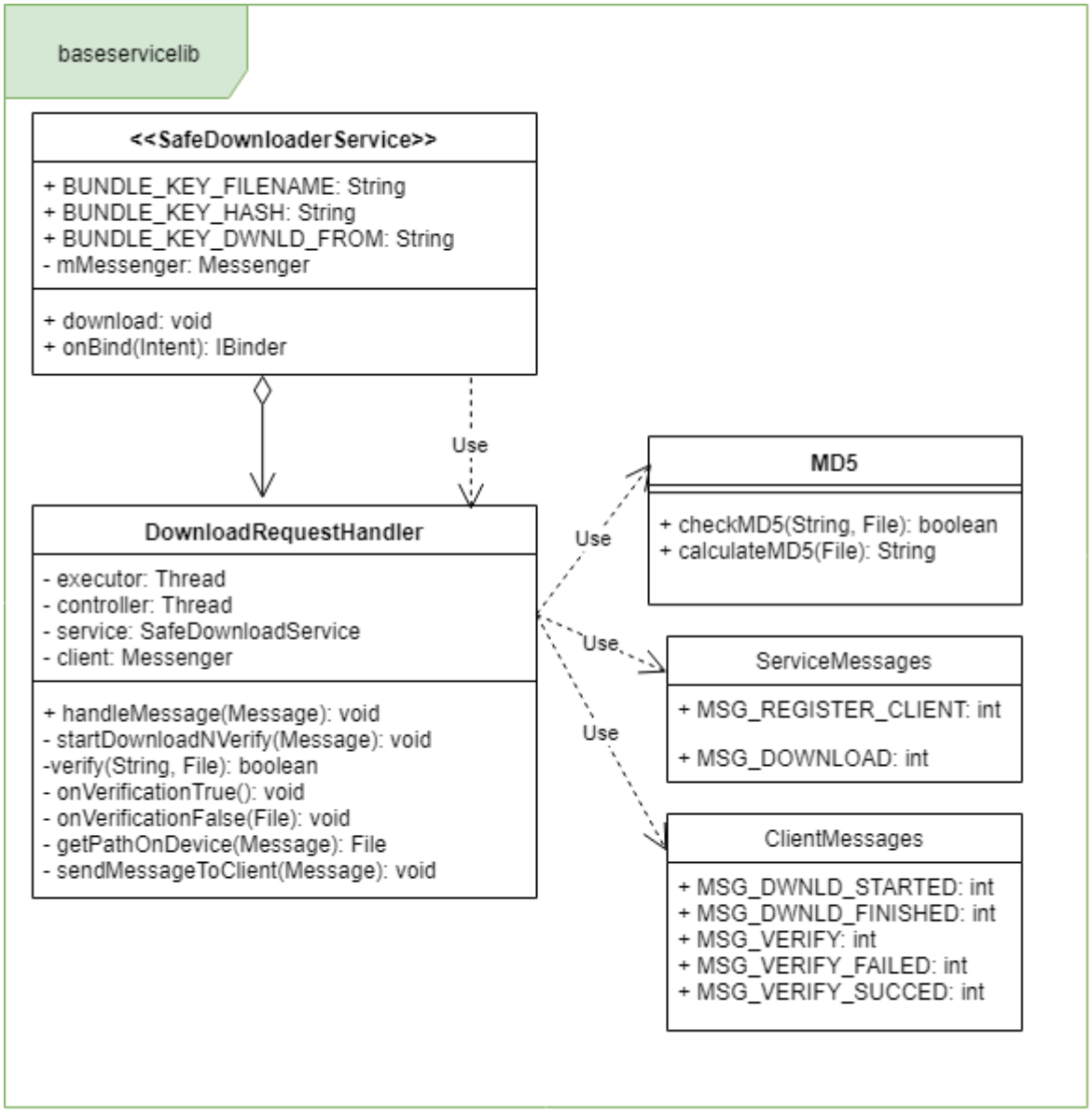

Рисунок 3.5 – UML-діаграма модуля baseservicelib

Таким чином, для використання цієї бібліотеки розробники застосунків повинні будуть реалізувати абстрактний інтерфейс SafeDownloaderService через метод download, описавши у ньому логіку завантаження. Також для реалізації зворотного зв’язку потрібна створити наслідувати клас Handler для отримання і обробки повідомлень від сервісу. Сервіс у такій реалізації є частиною розроблюваного застосунку.

3.3.4 Розробка тестового застосунку

Для тестування ефективності отриманої бібліотеки було спроектовано застосунок, що відтворює дії, подібні до розглянутих застосунків першого розділу. Застосунок імітує завантаження виконуваного файлу з серверу, використовуючи розроблену бібліотеку. В якості файлового серверу для спрощення було обрано використання облікового запису сервісу Dropbox. Рисунок 3.6 зображує UML-діаграму розробленого застосунку та його зв’язок із бібліотекою.

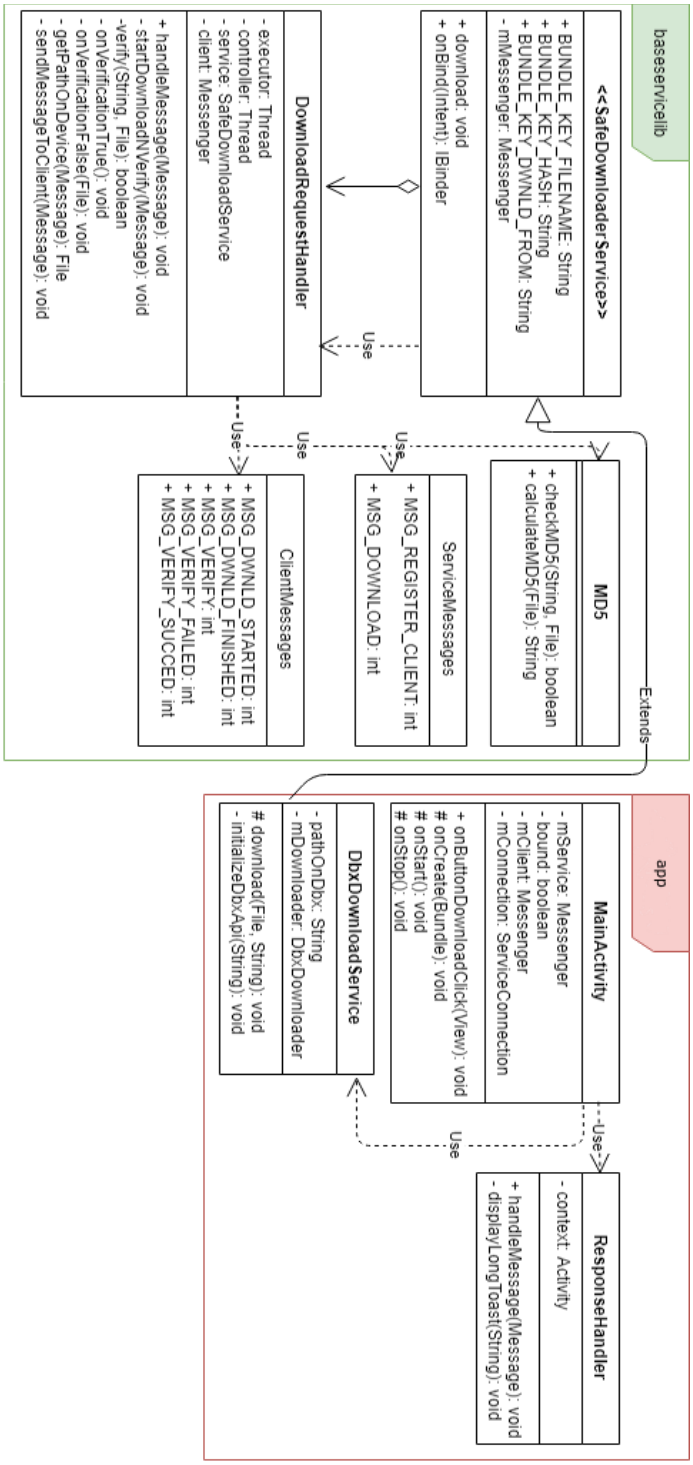

Рисунок 3.6 – UML-діаграма розробленого застосунку

3.3.5 Тестування

По натисканню на кнопку «Download» застосунок відправляє сервісу запит на завантаження файлу, сервіс відповідає застосунку повідомленням про стан запиту, реакція застосунку зображена на рисунку 3.7.

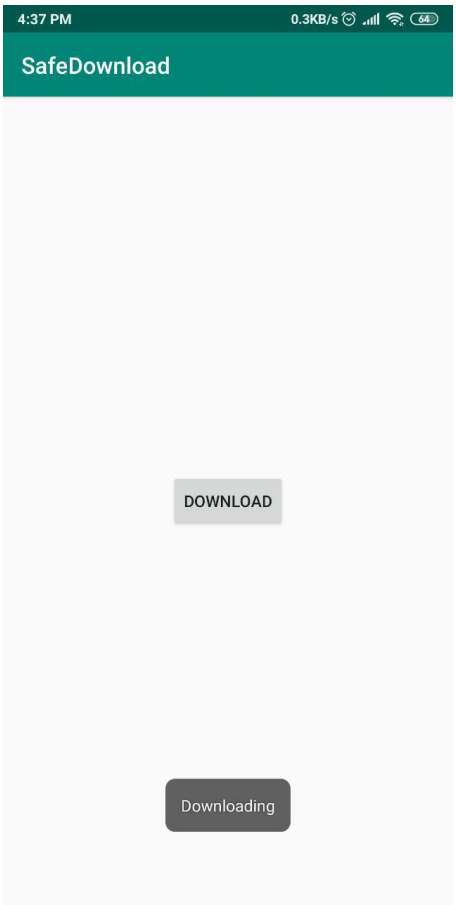

Рисунок 3.7 – Початок завантаження

Після закінчення завантаження починається процес завантаженого файлу. У випадку, якщо надана розробником контрольна сума співпадає з знайденою застосунком, то застосунок отримує від сервісу повідомлення із кодом MSG_VERIFY_SUCCEDSUCCEED, надалі застосунок може виконувати необхідні дії над файлом, рисунок 3.8 зображує обробку коду застосунком.

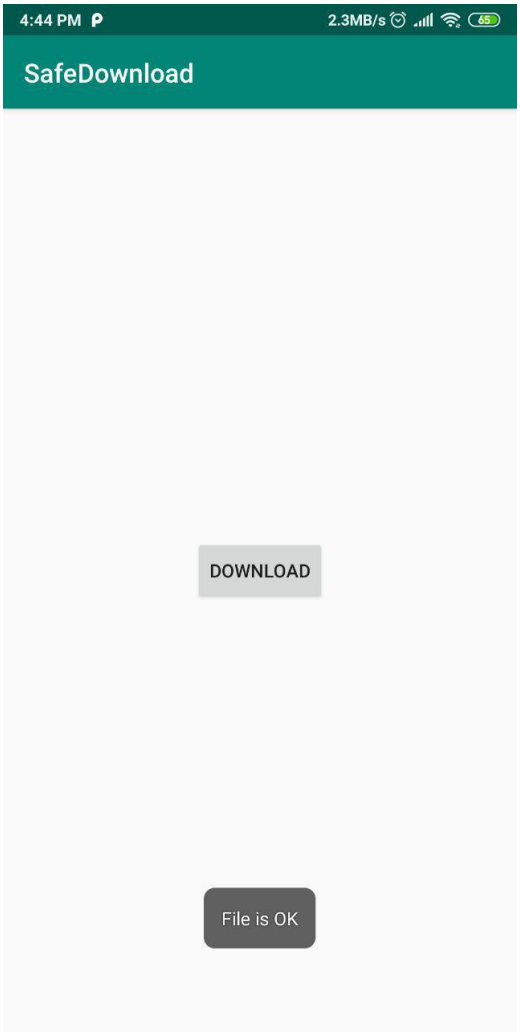

Рисунок 3.8 – Позитивний результат перевірки

У випадку, якщо контрольна заявлена сума відрізняється від отриманої сервісом, останній видалить завантажений файл та відправить застосунку повідомлення з кодом MSG_VERIFY_FAILED, обробка цього коду застосунком зображена на рисунку 3.9.

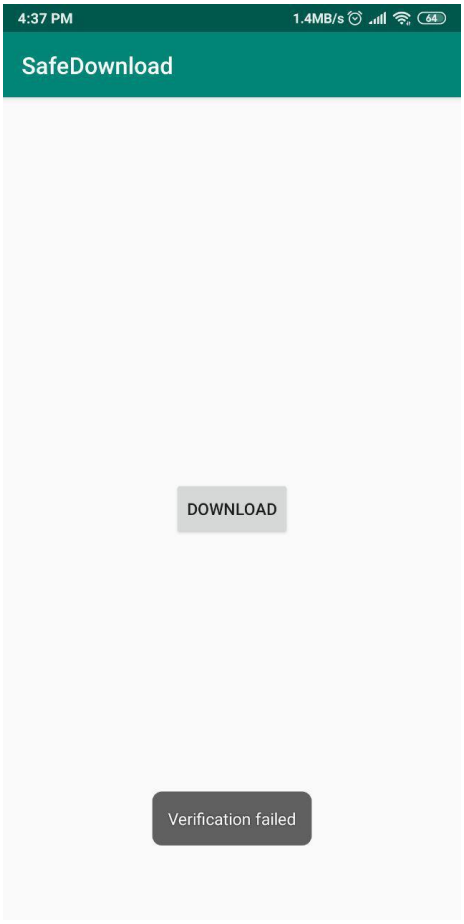

Рисунок 3.9 – Негативний результат перевірки

Висновки до розділу 3

Під час роботи над даним розділом було сформовано механізм захисту від атак типу «Man in the Disk» на основі відмови від використання ненадійного типу сховища, проведено проектування програмної реалізації цих механізмів. Досвід, отриманий на ранніх етапах реалізацій допоміг скорегувати загальний підхід до реалізації, на основі якого була розроблена бібліотека для створення зв’язаного сервісу для безпечного завантаження файлів із мережі Інтернет.

Досвід про особливості міжпроцесного зв’язку застосунків, виконуваних на варіації JVM, отриманий під час роботи цим розділом ліг в основу рекомендацій, що будуть запропоновані у наступному розділі. Лістинги вихідного коду

бібліотеки наведені у Додатку А, Додатку Б, Додатку В, Додатку Г та Додатку Д. Вихідні коди застосунку наведені у Додатку Є, Додатку Ж та Додатку З.

4 ПОБУДОВА РЕКОМЕНДАЦІЙ

4.1 Формування рекомендацій

Так як підхід, який було реалізовано у попередньому розділі базується на повній відмові від використання вразливого функціоналу, це гарантує неможливість проведення атаки «Man in the Disk», так як файли не потрапляють до зовнішнього сховища на жодному з етапів, які передують подальшому використанню застосунком файлу. Тому отриманий досвід стане основою рекомендацій, що будуть запропоновані у цьому розділі.

4.1.1 Рекомендації по розробці програмного рішення

Пропонується розробити бібліотеку класів, які нададуть розробнику можливість завантажувати файли з мережі Інтернет до внутрішнього сховища застосунку, минаючи зовнішнє сховище, що зробить неможливим проведення атаки «Man in the Disk» через усунення з області досяжності потенційної цілі.

Дана бібліотека повинна в обов’язковому порядку проводити наступну валідацію завантажених файлів, надаючи розробнику можливість самостійно обрати механізми валідації.

Враховуючи особливості функціонування платформи Android та віртуальної машини Dalvik VM, рисунок 4.1 пропонує наступний UML-діаграму такої бібліотеки.

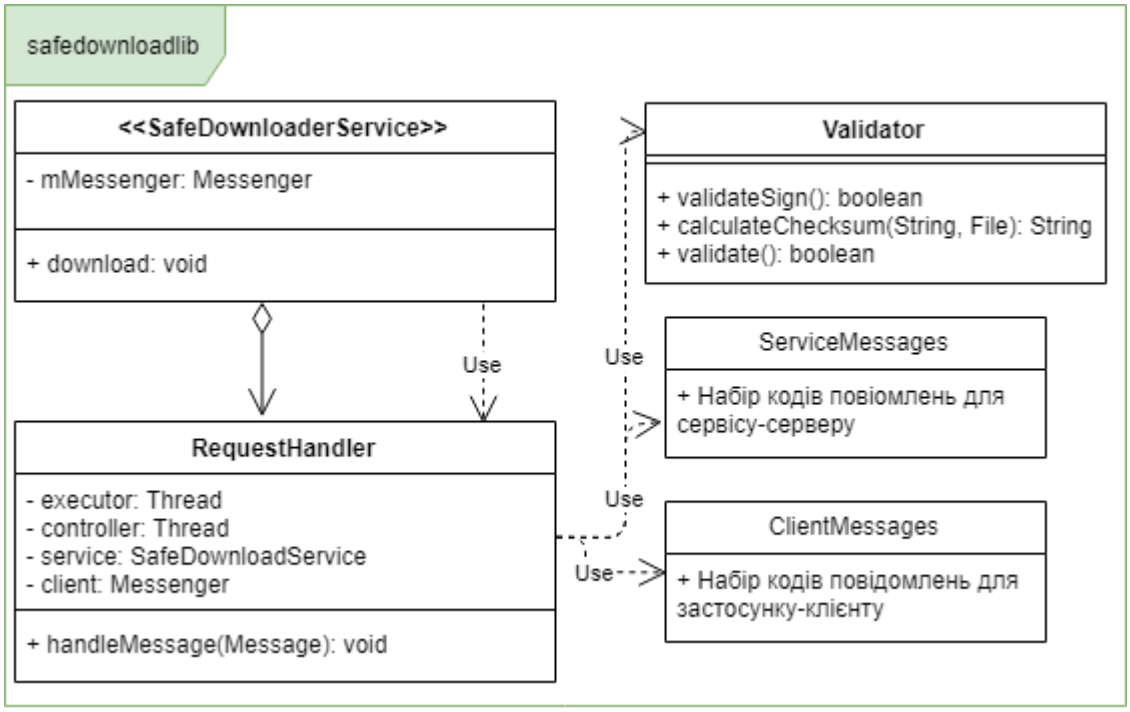

Рисунок 4.1 – UML-діаграма запропонованого рішення

Клас SafeDownloaderService є абстрактним та повинен реалізувати основу для сервісу завантажень, його детальний розбір надано у таблиці 4.1.

Таблиця 4.1 – Клас SafeDownloaderService

SafeDownloaderService	
Методи та поля	Опис
Messenger mMessenger	Об’єкт, який виконує міжпроцесний зв’язок
abstract void download(File, String)	Абстрактний метод, що повиненн бути реалізований розробником застосунку; виконує роль функціонального вказівника на метод з логікою завантаження

Клас RequestHandler повинен бути наслідуваний від вбудованого класу android.os.Handler, він бере на себе обробку отриманих від застосунку клієнта повідомлень. У таблиця 4.2 детально описані необхідні поля та методи.

Таблиця 4.2 – Клас RequestHandler

RequestHandler	
Методи та поля	Опис
Thread executor	Потік для виконання делегату з логікою завантаження, наданого розробником застосунку
Thread controller	Потік, що очікує кінця виконання завантаження з метою проведення валідації та надання відповідних повідомлень про стан застосунку
SafeDownloadService service	Посилання на сервіс, що створив об’єкт цього типу для доступу до методу download
Messenger client	Посилання на об’єкт, через який здійснюється зв’язок із застосунком.

Клас Validator повинен проводити валідацію за обраними розробником застосунку механізмами, деталі щодо його полів та методів подані у таблиці 4.3.

Таблиця 4.3 – Клас Validator

Validator	
Методи та поля	Опис
boolean validateSign()	Реалізовує перевірку цифрового підпису за обраним алгоритмом
String calculateChecksum(String, File)	Реалізовує знаходження контрольної суми за обраним алгоритмом
boolean validate()	Виносить фінальне рішення за результатами перевірки цифрового підпису та контрольної суми, рішення повинно виноситися, як «логічне і» результатів перевірки контрольної суми та цифрового підпису

Реалізація класу Validate повинна допускати певний рівень свободи для розробників застосунків. Так, розробник повинен мати можливість обирати з наданих механізмів валідації зручні для нього. Так, наприклад, повинна залишатися можливість обрати перевірку тільки одного підпису чи тільки однієї контрольної суми. Також повинна надаватися можливість вибору алгоритмів перевірки цифрового підпису та контрольної суми. Важливо зазначити, що даний клас повинен бути приватним на рівні його пакету та оголошений з модифікатором final для виключення можливості його наслідування.

Класи ServiceMessages та ClientMessages ідентичні за структурою, вони повинні бути визначені з модифікатором доступу public задля надання доступу застосунку до них та з модифікатором final. Їх поля визначаються сукупністю модифікаторів public static final для надання доступу на читання полів. Тип полів – цілі числа int.

4.1.2 Рекомендації по впровадженню програмного рішення

Так як дані рекомендації запропоновані до втілення виробникам пристроїв, їх реалізація може відрізнятися, тоді як розроблювані застосунки створюються з

урахуванням стандартної версії Android, тож кардинальні зміни, що будуть призводити до несправності у роботі застосунків негативно позначиться на досвіді користувачів.

Один із основних способів завантаження файлів – це використання класу BufferedInputStream для завантаження даних та класу FileOutputStream для завантаження. Метод BufferedInputStream.read() використовується для завантаження, тоді як для запису на пристрій використовується FileOutputStream.write(). Рисунок 4.2 ілюструє алгоритм необхідних до внесення змін.

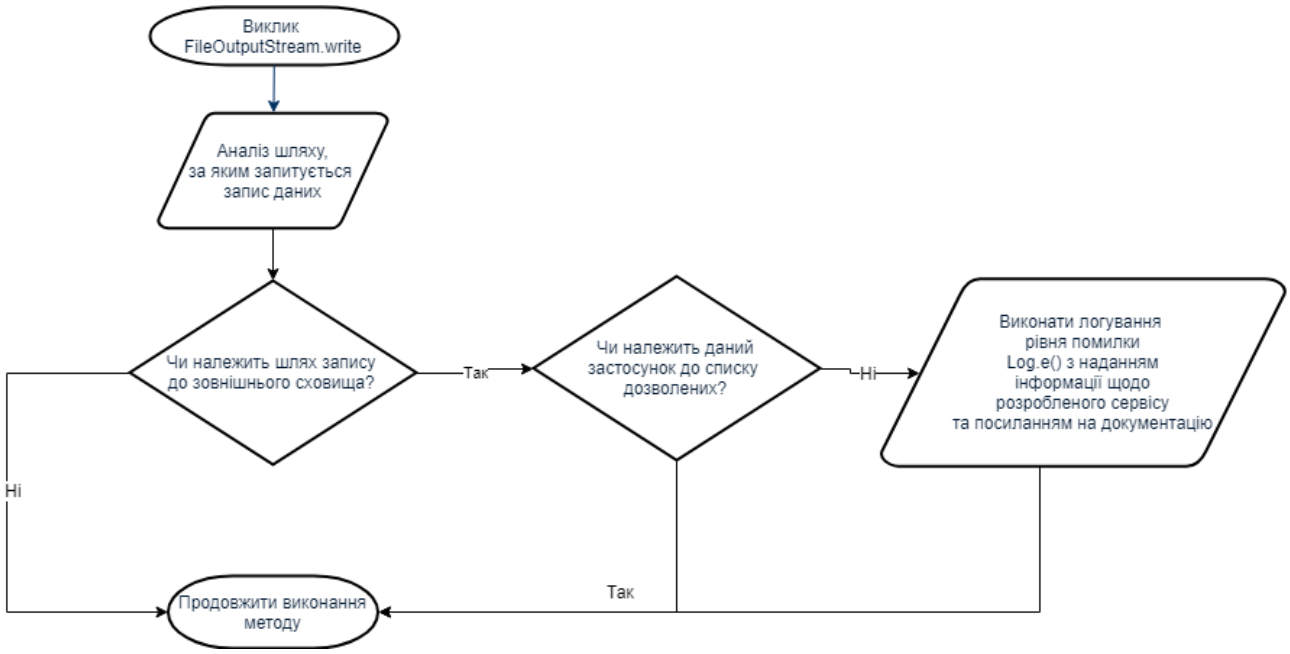

Рисунок 4.2 – Блок-схема запропонованих змін

Таким чином, головною концепцією інтеграції є повідомлення розробника застосунку через реєстрацію повідомлення на рівні помилок з посиланням на документацію до бібліотеки для даної надбудови над Android.

Невід’ємною частиною впровадження подібних програмних рішень є організаційна робота. Повинна бути створена зручна з точки зору навігації та зрозуміла документація. У випадку реалізації бібліотеки мовою Java, найкращим варіантом є використання анотацій документування, що дозволить автоматично

згенерувати зручну документацію у вигляді HTML-сторінок за допомогою утиліти javadoc. Також слід надати розробникам приклад простого застосунку, що ілюструє інтеграцію бібліотеки до проекту та використання основного функціоналу. Розміщувати такий «посібник» рекомендується на одному популярних та простих у використанні онлайн-репозиторіїв, наприклад Github або Gitlab, забезпечивши його детальним описом з посиланням на згенеровану за допомогою javadoc.

Висновки до розділу 4

Під час роботи над даним розділом було використано отримані протягом роботи над попередніми розділами дані для формування рекомендацій по створенню та впровадженню програмного рішення. Запропоновані у цьому розділі рекомендації спрямовані на впровадження виробниками пристроїв з метою створення тимчасового рішення даної вразливості. Запропоновані рекомендації по впровадженню спрямовані на збереження оберненої сумісності операційної системи з додатками, які не використовують надане рішення з метою підтримання досвіду користувача на заданому рівні.

ВИСНОВКИ

У даній роботі було розглянуто питання захисту від атак типу «Man in the Disk» для платформи Android.

Згідно поставленим завданням, було досліджено особливості роботи операційної системи, пов’язані з атакою «MitD», та алгоритм її проведення шляхом вивчення джерел та документації. Отримані дані поглибили розуміння функціонування платформи Android загалом та були враховані при аналізі підходів до подібних проблем.

Було проаналізовано підхід до вирішення подібних вразливостей, а також до вирішення даної вразливості у застосунках, де вона була знайдена. Всі розглянуті випадки об’єднував підхід до вирішення шляхом відмови від потенційно небезпечного функціоналу. Це, а також ситуація з розглянутою вразливістю macOS, зумовили необхідність тимчасового рішення даної вразливості шляхом посильного уникнення використання зовнішнього сховища застосунками.

На основі обраного в результаті аналізу підходу було сформовано механізм захисту, який базується на впровадженні системного додатку. Дана концепція зазнала змін внаслідок обмежень, що стали очевидними під час створення програмної реалізації механізму. В результаті було розроблено бібліотеку, яка надає простий інтерфейс для безпечного завантаження файлів з мережі та виконує їх подальшу валідацію.

Отриманий під час реалізації сформованого механізму захисту досвід допоміг запропонувати рекомендації щодо створення програмного рішення для захисту від даного типу атак та впровадження цього рішення. Отримані рекомендації можуть бути використані виробниками пристроїв для реалізації у власних версіях Android тимчасового захисту від даної вразливості.

Планується подальше вивчення запропонованих в бета-версіях Android Q рішень щодо використання зовнішнього сховища з метою удосконалення запропонованих рекомендацій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018 [Електронний ресурс] // Statista – Режим доступу до ресурсу: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.
2. Android (operating system) [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Android_(operating_system).
3. We Broke Into A Bunch Of Android Phones With A 3D-Printed Head [Електронний ресурс] // Forbes. – 2018. – Режим доступу до ресурсу: https://www.forbes.com/sites/thomasbrewster/2018/12/13/we-broke-into-a-bunch-of-android-phones-with-a-3d-printed-head/#692000e51330.
4. Secure an Android Device [Електронний ресурс] // Source Android – Режим доступу до ресурсу: https://source.android.com/security.
5. Системное программирование [Електронний ресурс] // Подкаст "Podlodka". – 2018. – Режим доступу до ресурсу: https://soundcloud.com/podlodka/podlodka-86-sistemnoe-programmirovanie.
6. Application Sandbox [Електронний ресурс] // Source Android – Режим доступу до ресурсу: https://source.android.com/security/app-sandbox.
7. Data and file storage overview [Електронний ресурс] // Android Developer – Режим доступу до ресурсу: https://developer.android.com/guide/topics/data/data-storage#filesInternal.
8. 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/.
9. Man-in-the-Disk: A New Attack Surface for Android Apps [Електронний ресурс] // Check Point Blog. – 2018. – Режим доступу до ресурсу:

https://blog.checkpoint.com/2018/08/12/man-in-the-disk-a-new-attack-surface-for-android-apps/.

10. DEF CON 26 - Slava Makkaveev - Man In The Disk [Електронний ресурс] // DEFCONConference YouTube channel. – 2018. – Режим доступу до ресурсу: https://www.youtube.com/watch?v=vvfs0u1or3M.
11. Fortnite Installer downloads are vulnerable to hijacking [Електронний ресурс] // Google Issue Tracker. – 2018. – Режим доступу до ресурсу: https://issuetracker.google.com/issues/112630336
12. MacOS X GateKeeper Bypass [Електронний ресурс] // Filippo Cavallarin. – 2019. – Режим доступу до ресурсу: https://www.fcvl.net/vulnerabilities/macosx-gatekeeper-bypass.
13. Android Q privacy change: Scoped storage [Електронний ресурс] // Developers Android – Режим доступу до ресурсу: https://developer.android.com/preview/privacy/scoped-storage.

ДОДАТКИ

ДОДАТОК А

SafeDownloaderService.java

```
package com.legion1900.baseservicelib.service;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.Messenger;

import java.io.File;

public abstract class SafeDownloaderService extends Service {

    public static final String BUNDLE_KEY_FILENAME = "fileName";
    public static final String BUNDLE_KEY_HASH = "md5";
    public static final String BUNDLE_KEY_DWNLD_FROM = "downloadFrom";

    private Messenger mMessenger;

    abstract protected void download(File pathOnDevice, String
downloadFrom);

    @Override
    public IBinder onBind(Intent intent) {
        mMessenger = new Messenger(new DownloadRequestHandler(this));
        return mMessenger.getBinder();
    }
}
```

ДОДАТОК Б

DownloadRequestHandler.java

```
package com.legion1900.baseservicelib.service;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;

import java.io.File;

/*
 * Handler of incoming messages from clients
 * */
class DownloadRequestHandler extends Handler {

    private static final String TAG_ERROR_CONTROLLER = "Verifier";
    private static final String TAG_ERROR_RESPONDING =
"DownloadRequestHandler";

    private Thread executor;

    private Thread controller;

    // Reference to parent service.
    private SafeDownloaderService service;

    private Messenger client;

    DownloadRequestHandler(SafeDownloaderService service) {
        this.service = service;
    }

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case ServiceMessages.MSG_REGISTER_CLIENT:
```

```
                    client = msg.replyTo;
                    break;
                case ServiceMessages.MSG_DOWNLOAD:
                    startDownloadNVerify(msg);
                    // TODO add verification
                default:
                    super.handleMessage(msg);
            }
        }

        private void startDownloadNVerify(Message msg) {
            Message response = Message.obtain(null,
ClientMessages.MSG_DWNLD_STARTED);
            sendMessageToClient(response);

            if (msg.obj.getClass() == Bundle.class) {
                Bundle bundle = (Bundle)msg.obj;
                File pathOnDevice = getPathOnDevice(msg);
                String downloadFrom =
bundle.getString(SafeDownloaderService.BUNDLE_KEY_DWNLD_FROM);
                String hash =
bundle.getString(SafeDownloaderService.BUNDLE_KEY_HASH);
                executor = new Thread(new Downloader(pathOnDevice,
downloadFrom));
                controller = new Thread(new Verifier(hash, pathOnDevice));
                executor.start();
                controller.start();
            }
            else {
                throw new IllegalArgumentException("File obj should be Bundle");
            }
        }

        private boolean verify(String hash, File file) {
            Message response = Message.obtain(null, ClientMessages.MSG_VERIFY);
            sendMessageToClient(response);

            return MD5.checkMD5(hash, file);
        }

        private void onVerificationTrue() {
```

```
            Message response = Message.obtain(null,
ClientMessages.MSG_VERIFY_SUCCEDSUCCEED);
            sendMessageToClient(response);
        }

        private void onVerificationFalse(File fileToBeDeleted) {
            Message response = Message.obtain(null,
ClientMessages.MSG_VERIFY_FAILED);
            sendMessageToClient(response);
            fileToBeDeleted.delete();
        }

        private File getPathOnDevice(Message msg) {
            Bundle bundle = (Bundle)msg.obj;
            String fileName =
bundle.getString(SafeDownloaderService.BUNDLE_KEY_FILENAME);
            return new File(
                    service.getFilesDir()
                            + "/"
                            + fileName);
        }

        private void sendMessageToClient(Message msg) {
            try {
                client.send(msg);
            }
            catch (RemoteException e) {
                Log.e(TAG_ERROR_RESPONDING, "Cannot respond to client", e);
            }
        }

        private class Downloader implements Runnable {

            File pathOnDevice;

            String downloadFrom;

            Downloader(File pathOnDevice, String downloadFrom) {
                this.pathOnDevice = pathOnDevice;
                this.downloadFrom = downloadFrom;
            }
```

```
            @Override
            public void run() {
                service.download(pathOnDevice, downloadFrom);
            }
        }

        private class Verifier implements Runnable {

            final String hash;
            final File file;

            Verifier(String hash, File file) {
                this.hash = hash;
                this.file = file;
            }

            @Override
            public void run() {
                try {
                    executor.join();
                    Message response = Message.obtain(null,
ClientMessages.MSG_DWNLD_FINISHED);
                    sendMessageToClient(response);

                    boolean result = verify(hash, file);
                    if (result)
                        onVerificationTrue();
                    else
                        onVerificationFalse(file);
                }
                catch (InterruptedException e) {
                    Log.e(TAG_ERROR_CONTROLLER, "Interrupter", e);
                }
            }
        }
    }
```

ДОДАТОК В

MD5.java

```
package com.legion1900.baseservicelib.service;

import android.text.TextUtils;
import android.util.Log;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class MD5 {
    private static final String TAG = "MD5";

    public static boolean checkMD5(String md5, File file) {
        if (TextUtils.isEmpty(md5) || file == null) {
            Log.e(TAG, "MD5 string empty or updateFile null");
            return false;
        }

        String calculatedDigest = calculateMD5(file);
        if (calculatedDigest == null) {
            Log.e(TAG, "calculatedDigest null");
            return false;
        }

        Log.v(TAG, "Calculated digest: " + calculatedDigest);
        Log.v(TAG, "Provided digest: " + md5);

        return calculatedDigest.equalsIgnoreCase(md5);
    }

    public static String calculateMD5(File updateFile) {
```

```
        MessageDigest digest;
        try {
            digest = MessageDigest.getInstance("MD5");
        } catch (NoSuchAlgorithmException e) {
            Log.e(TAG, "Exception while getting digest", e);
            return null;
        }

        InputStream is;
        try {
            is = new FileInputStream(updateFile);
        } catch (FileNotFoundException e) {
            Log.e(TAG, "Exception while getting FileInputStream", e);
            return null;
        }

        byte[] buffer = new byte[8192];
        int read;
        try {
            while ((read = is.read(buffer)) > 0) {
                digest.update(buffer, 0, read);
            }
            byte[] md5sum = digest.digest();
            BigInteger bigInt = new BigInteger(1, md5sum);
            String output = bigInt.toString(16);
            // Fill to 32 chars
            output = String.format("%32s", output).replace(' ', '0');
            return output;
        } catch (IOException e) {
            throw new RuntimeException("Unable to process file for MD5", e);
        } finally {
            try {
                is.close();
            } catch (IOException e) {
                Log.e(TAG, "Exception on closing MD5 input stream", e);
            }
        }
    }
}
```

ДОДАТОК Г

ServiceMessages.java

```
package com.legion1900.baseservicelib.service;

/*
 * Messages for service
 * */
public final class ServiceMessages {
    public static final int MSG_REGISTER_CLIENT = 1;
    public static final int MSG_DOWNLOAD = 2;
}
```

ДОДАТОК Д

ClientMessages.java

```
package com.legion1900.baseservicelib.service;

/*
* Messages for client
* */
public final class ClientMessages {
    public static final int MSG_DWNLD_STARTED = 1;
    public static final int MSG_DWNLD_FINISHED = 2;
    public static final int MSG_VERIFY = 3;
    public static final int MSG_VERIFY_FAILED = 4;
    public static final int MSG_VERIFY_SUCCEDSUCCEED = 5;
}
```

ДОДАТОК Е

app: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.legion1900.safedownload">
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".service.DbxDownloadService"/>
    </application>

</manifest>
```

ДОДАТОК Є

app MainActivity.java

```
package com.legion1900.safedownload;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

import com.legion1900.baseservicelib.service.SafeDownloaderService;
import com.legion1900.baseservicelib.service.ServiceMessages;
import com.legion1900.safedownload.service.DbxDownloadService;
import com.legion1900.safedownload.service.ResponseHandler;

public class MainActivity extends AppCompatActivity {

    private final static String MOC_HASH =
"00e1f0cb8aaf13c5fe189400182fc82b";

    private static final String TAG_ERROR_SERVICE_CONNECTION =
"ServiceConnection";

    private static final String PATH_TO_FILE = "/test.apk";

    private Messenger mService = null;
    private boolean bound;
    private final Messenger mClient = new Messenger(new
ResponseHandler(this));

    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
```

```
            public void onServiceConnected(ComponentName name, IBinder service)
{
                mService = new Messenger(service);
                bound = true;

                Message msg = Message.obtain(null,
ServiceMessages.MSG_REGISTER_CLIENT);
                msg.replyTo = mClient;
                try {
                    mService.send(msg);
                }
                catch (RemoteException e) {
                    Log.e(TAG_ERROR_SERVICE_CONNECTION, "Cannot send response to
service", e);
                }
            }

            @Override
            public void onServiceDisconnected(ComponentName name) {
                mService = null;
                bound = false;
            }
        };

        public void onButtonDownloadClick(View parent) {
            if (!bound) return;
            Message msg = Message.obtain(null, ServiceMessages.MSG_DOWNLOAD, 0,
0);
            Bundle args = new Bundle();
            args.putString(SafeDownloaderService.BUNDLE_KEY_DWNLD_FROM,
PATH_TO_FILE);
            args.putString(SafeDownloaderService.BUNDLE_KEY_FILENAME,
"test.apk");
            args.putString(SafeDownloaderService.BUNDLE_KEY_HASH, MOC_HASH);
            msg.obj = args;
            try {
                mService.send(msg);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    protected void onStart() {
        super.onStart();
        bindService(new Intent(this, DbxDownloadService.class), mConnection,
                Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onStop() {
        super.onStop();
        if (bound) {
            unbindService(mConnection);
            bound = false;
        }
    }
}
```

ДОДАТОК Ж

DbxDownloadService.java

```
package com.legion1900.safedownload.service;

import android.util.Log;
import android.widget.Toast;

import com.dropbox.core.DbxDownloader;
import com.dropbox.core.DbxException;
import com.dropbox.core.DbxRequestConfig;
import com.dropbox.core.v2.DbxClientV2;
import com.legion1900.baseservicelib.service.SafeDownloaderService;
import com.legion1900.safedownload.BuildConfig;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class DbxDownloadService extends SafeDownloaderService {

    private static final String TAG = "DbxDownloadService";

    private DbxDownloader mDownloader;

    private String pathOnDbx;

    @Override
    protected void download(File pathOnDevice, String downloadFrom) {
        initializeDbxApi(downloadFrom);

        // Create new file.
        try {
            pathOnDevice.createNewFile();
        }
        catch (IOException e) {
            Log.e(TAG, "Cannot create file on device", e);
        }
```

```
            // Download logic itself.
            FileOutputStream fout = null;
            try {
                fout = new FileOutputStream(pathOnDevice);
                mDownloader.download(fout);
            }
            catch (FileNotFoundException e) {
                Log.e(TAG, "Cannot create FileOutputStream", e);
            }
            catch (DbxException e) {
                Log.e(TAG, "Download failure", e);
            }
            catch (IOException e) {
                Log.e(TAG, "Download failure", e);
            }
        }

        private void initializeDbxApi(String pathOnDbx) {
            String id = getApplicationContext().getPackageName();
            DbxRequestConfig config = DbxRequestConfig.newBuilder(id).build();
            DbxClientV2 client = new DbxClientV2(config,
BuildConfig.MyAccessToken);
            try {
                mDownloader = client.files().download(pathOnDbx);
            }
            catch (DbxException e) {
                Log.e(TAG, "API initialization error", e);
            }
        }
    }
```

ДОДАТОК З

ResponseHandler.java

```
package com.legion1900.safedownload.service;

import android.app.Activity;
import android.os.Handler;
import android.os.Message;
import android.widget.Toast;

import com.legion1900.baseservicelib.service.ClientMessages;

public class ResponseHandler extends Handler {

    private static final String ON_DOWNLOAD_STARTED = "Downloading";
    private static final String ON_DOWNLOAD_FINISHED = "Downloading is
finished";
    private static final String ON_VERIFY = "Verifying";
    private static final String ON_VERIFY_FAILED = "Verification failed";
    private static final String ON_VERIFY_SUCCEED = "File is OK";

    private final Activity context;

    public ResponseHandler(Activity context) {
        this.context = context;
    }

    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case ClientMessages.MSG_DWNLD_STARTED:
                displayLongToast(ON_DOWNLOAD_STARTED);
                break;
            case ClientMessages.MSG_DWNLD_FINISHED:
                displayLongToast(ON_DOWNLOAD_FINISHED);
                break;
            case ClientMessages.MSG_VERIFY:
                displayLongToast(ON_VERIFY);
                break;
            case ClientMessages.MSG_VERIFY_FAILED:
```

```
                displayLongToast(ON_VERIFY_FAILED);
                break;
            case ClientMessages.MSG_VERIFY_SUCCEDSUCCEED:
                displayLongToast(ON_VERIFY_SUCCEED);
                break;
            default: super.handleMessage(msg);
        }
    }

    private void displayLongToast(String text) {
        Toast.makeText(context, text, Toast.LENGTH_LONG).show();
    }
}
```