Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПІДВИЩЕННЯ ЗАХИСТУ ДОДАТКІВ НА ОПЕРАЦІЙНІЙ СИСТЕМІ ANDROID ВІД РЕВЕРС-ІНЖИНІРИНГУ

Текстова частина до дипломної роботи
за спеціальністю «Інженерія програмного забезпечення»

Керівник дипломної роботи
ст. викладач Борозенний С. О.

_____
(підпис)
“____” _____ 2023 р.

Виконав студент 4 р. н.
Макаренко Д. О.
“____” _____ 2023 р.

Київ – 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем

доцент, к.ф.-м.н.

О. П. Жежерун

_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту 4-го курсу факультету інформатики Макаренку Данилу Олександровичу

Розробити Застосунок для підвищення захисту додатків на операційній системі Android від реверс-інжинірингу

Зміст ТЧ до дипломної роботи:

Зміст

Вступ

Теоретичні відомості

Реверс-інжиніринг

Розробка застосунку для захисту від статичного реверс-інжинірингу

Висновки

Список літератури

Дата видачі “___” _____ 2023 р. Керівник _____

Завдання отримав _____

Тема: Розробка застосунку для підвищення захисту додатків на операційній системі Android від реверс-інжинірингу

Календарний план виконання роботи:

№	Етап	Термін виконання
1.	Отримання завдання на дипломну роботу.	06.10.2022
2.	Огляд технічної літератури за темою роботи.	24.12.2022
3.	Аналіз існуючих методів вирішення поставленої задачі.	15.01.2023
4.	Розробка програми для протидії статичному реверс-інжинірингу.	30.03.2023
5.	Написання текстової частини роботи.	18.04.2023
6.	Створення слайдів для доповіді та написання доповіді.	30.04.2023
7.	Аналіз отриманих результатів з керівником, остаточне оформлення доповіді.	05.05.2023
8.	Захист дипломної роботи.	01.06.2023

Студент Макаренко Данило Олександрович

Керівник Борозенний О. С.

“_____”_____

Зміст

Вступ

Інформаційні технології щоденно все більше проникають в наше життя, роблячи його простішим, зручнішим та цікавішим. Величезну популярність серед них мають саме мобільні пристрої.

Мобільні пристрої мають цілий ряд переваг, що буде тільки збільшувати попит на них із часом :

1. Компактність. Завжди під рукою через невеликі розміри.
2. Доступ до інтернету.
3. Спілкування.
4. Камера. Не потрібно завжди носити із собою фотокамеру, щоб зафіксувати важливу інформацію.
5. Банкінг. Можливість оплати без готівки чи навіть без банківської карти.
6. Багато застосунків. Деякі з них, такі як «Дія», несуть невичерпну користь та зручність.
7. Розваги. Ігри, «YouTube» та інші мають особливо великий вплив на популярність мобільних пристроїв.
8. Доступність. Значна кількість мобільних пристроїв коштує відносно недорого, що дозволяє майже кожному поринути у світ інформаційних технологій.

Велика кількість користувачів мобільних пристроїв побудила бізнеси створювати мобільні застосунки. Хоча й деякі застосунки є лише доповнення до вже існуючих бізнесів : «Нова пошта», «Сільпо», «Приват24», існують самодостатні застосунки такі як : «Quizlet», різноманітні ігри та месенджери.

Об’єднує ці дві категорії мобільних застосунків той факт, що усі вони в тій чи інший мірі потребують захисту : конфіденційної інформації користувачів; від втручання в роботу застосунку; коду застосунку та використаних алгоритмів. Справа в тому, що такий попит на мобільні пристрої серед людей зацікавив не лише власників бізнесів, а й зловмисників. Розробники операційних систем дбають у першу чергу про безпеку ОС, а розробники застосунків несуть відповідальність за безпеку свого продукту.

Особливо актуальними для мобільних застосунків є атаки на стороні клієнта, адже користувач застосунку має доступ до оперативної та довготривалої пам’яті, а також до коду, що виконується на пристрої. Використовуючи засоби реверс-інжинірингу, зловмисник може отримати доступ до логіки роботи застосунку та матиме можливість змінити її. Зрозуміло, що отримання початкового коду хакерами може нанести значний репутаційний удар бізнесу, зменшити прибуток від продажу платних версій застосунку, зменшити кількість переглядів реклами.

Компанія Arxan Technologies у 2014 році провела дослідження «State of Mobile App Security» [1]. Найцікавішими фактами дослідження є :

1. 97% найпопулярніших платних додатків мали безкоштовних клонів.
2. Загалом станом на квітень 2014 року було виявлено 890 482 застосунків, отриманих в результаті реверс-інжинірингу.
3. Основною проблемою було не те, що підробки привласнювали собі інтелектуальну працю, а те, що більш ніж 50% модифікованих застосунків були шкідливим ПЗ.
4. Більш ніж 65% модифікованих застосунків змінювали роботу з рекламою, тим самим зменшуючи прибутки розробників додатків.

У 2022 році компанія Osterman Research спільно з компанією Approv провели опитування [2] серед трьохсот двох розробників мобільних додатків із США та Великої Британії. 48% опитаних працюють у компаніях до 500 співробітників, 42% - у компаніях, де від 501 до 4999 працівників, та 10% - у організаціях із понад 5000 співробітників. Дане дослідження не тільки демонструє ріст важливості мобільних застосунків для бізнесів (рисунок Д.1), а й наявність проблем у безпеці мобільних застосунків.

Відповідно до даного дослідження, 78% опитаних мають сумніви, що їх мобільні застосунки мають достатній рівень захисту. Зрозуміло, що йдеться не лише про захист від реверс-інжинірингу, а й від інших видів атак. Однак близько 50% респондентів не впевнені в безпеці застосунків від реверс- інжинірингу. Важливим також є те, що незважаючи на розуміння того, що безпека мобільних застосунків має велике значення, компанії часто не приділяють цьому значної уваги, та розглядають такі питання як низько пріоритетні.

Актуальність проблеми протидії реверс-інжинірингу підтверджується ще одним дослідженням 2016 року, яке провели в компанії Open Worldwide Application Security Project (OWASP) [3]. Відповідно до цього дослідження, реверс-інжиніринг входить до десяти найбільш популярних вразливостей мобільних застосунків.

Тема протидії реверс-інжинірингу надзвичайно мене зацікавила, адже я працюю розробником мобільних додатків під операційну систему Android більше року, проте майже ніколи не чув серед колег розмов щодо протидії реверс-інжинірингу. Подібних питань я не помічав і на співбесідах. Єдиним засобом протидії реверс-інжинірингу, який я знав, була обфускація коду. Почавши проводити моє дослідження, я знайшов в інтернеті досить багато

готових методів боротьби з реверс-інжинірингом, проте детальної аналітики щодо їх ефективності мені знайти не вдалося. Чи потрібно використовувати всі існуючі? Чи можу я винайти нові чи покращити вже існуючі методи протидії реверс-інжинірингу? Цікавим методом для покращення став для мене застосунок-пакувальник Android додатків. Ідея полягає в тому, щоб зашифрувати всю логіку Android застосунку задля унеможливлення статичного реверс-інжинірингу, а в процесі виконання додатку на пристрої розшифровувати її.

Отже, метою моєї кваліфікаційної роботи є розробка застосунку для підвищення захисту додатків на операційній системі Android від реверс-інжинірингу.

Основна частина

1. Теоретичні відомості

1.1. Android

Android – операційна система для мобільних телефонів із відкритим кодом на базі ядра Linux створена компанією Google в 2008 році. На момент написання кваліфікаційної роботи, операційна система Android для мобільних телефонів є найбільш популярною у світі. Станом на серпень 2022 відповідно до [4], 71.54% мобільних телефонів використовують саме операційну систему Android.

Деталі реалізації ОС Android та історія її розвитку є окремими цікавими темами для дослідження. Реверс-інжиніринг пов’язаний саме із застосунками, тому інформації про те, як Android працює із застосунками буде достатньо для повного висвітлення теми.

Застосунки для ОС Android найчастіше пишуть мовою Java чи Kotlin. Їх виконання забезпечує ART(Android Runtime), що є наступником віртуальної машини Dalvik. Процес підготовки застосунків до виконання виглядає наступним чином.

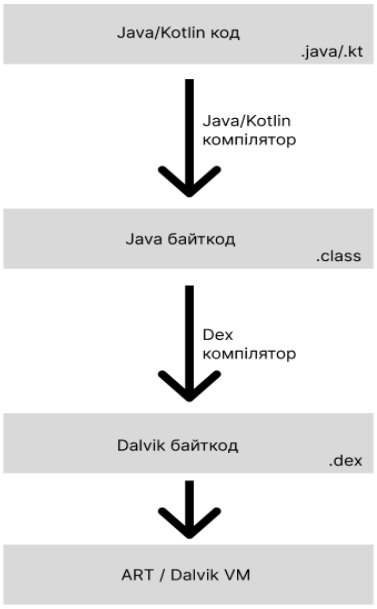

Рисунок 1.1 – Процес компіляції мобільних застосунків для ОС Android

1.2 Що таке APK?

Android **P**ac**k**age – це формат файлу, який використовується для поширення та інсталяції застосунків для ОС Android. APK є підмножиною формата архівів ZIP : він не шифрується та включає в себе всі файли необхідні для інсталяції застосунку.

Пакет Android застосунку включає :

1. Файли формату .dex (Dalvik Executable) – скомпільований код застосунку, що буде виконуватись віртуальною машиною Dalvik або ART(Android Runtime)
2. Каталог META-INF – метадані, що включають підпис застосунку, контрольні суми ресурсів та іншу службову інформацію.
3. Каталог res – ресурси застосунку : файли розмітки екранів, зображення, стилі, кольори, тестові рядки тощо.

4. AndroidManifest.xml
   4.1. основні компоненти застосунку
   4.2. необхідні дозволи – наприклад дозвіл на використання камери, доступ до геолокації чи файлової системи телефону
   4.3. необхідні апаратні та програмні особливості – наприклад наявність компасу, камери чи Bluetooth.
5. Каталог assets – для різноманітних JSON, MP3 та інших форматів файлів.
6. resources.arsc – таблиця відповідності ресурсів та їх унікальних ідентифікаторів.

Насправді APK включає й інші каталоги та файли, проте вони не є корисними в контексті дослідження реверс-інжинірингу.

1.3 Google Play

Google Play – офіційна крамниця, що станом на 2022 рік містить понад 2 мільйони Android застосунків [5]. Даний сервіс дозволяє розробникам публікувати, а користувачам ОС Android – завантажувати розміщені застосунки. Google Play – єдине офіційне джерело застосунків. Перед публікацією Google Play перевіряє кожен застосунок на наявність шкідливих функцій. Спочатку Google використовувала Bouncer для захисту від шкідливого ПЗ, проте певні види вразливостей не відстежувались [6], то ж наразі компанія використовує Google Play Protect[7]. Однак ОС Android не забороняє інсталяцію застосунків із будь-яких інших джерел, проте попереджає, що це може бути небезпечно. Справа в тому, що завантаження застосунків зі сторонніх джерел часто виконують з метою отримання платних версій ігор чи застосунків безкоштовно чи з метою відключення реклами. Цим користуються зловмисники, додаючи до змінених застосунків крадіжку чи шифрування даних пристрою та інші шкідливі функції.

1.4 Підпис застосунків

Кожен застосунок, що встановлюється на пристрій чи завантажуються в Google Play обов'язково має бути підписаним цифровим ключем Java KeyStore (JKS)[8]. Головною метою використання підпису є ідентифікація автора застосунку. Відповідальність за генерацію та безпечне зберігання цифрового ключа покладається на розробників застосунку. Підпис гарантує, що навіть якщо зловмисник отримав облікові дані від Google Play Console, то без цифрового ключа він не зможе від імені розробників завантажити підроблену версію застосунку в Google Play. Підпис також використовується пакетним менеджером мобільного пристрою для верифікації того, що оновлена версія застосунку підписана тим же ключем, що й вже існуюча, таким чином гарантуючи, що застосунок не був підроблений.

2 Реверс-інжиніринг

2.1 Що таке реверс-інжиніринг?

Реверс-інжиніринг (Reverse Engineering) – це процес зворотної розробки при якому за допомогою різноманітних методів аналізується застосунок із метою виявлення принципів його роботи. Загалом реверс-інжиніринг складається із завантаження застосунку, декомпіляції, проведення необхідних змін та рекомпіляції. Найчастіше реверс-інжиніринг націлений на застосунки, для яких інтелектуальна власність є головною метою бізнесу. Так автор статті [9] за допомогою реверс-інжинірингу Android застосунку популярної платформи для електронних публікацій Medium зробив усі статті розміщені на ресурсі безкоштовними.

Застосунки для ОС Android найчастіше пишуться мовами Java чи Kotlin. При компіляції двох останніх отримується байт-код, який зберігає достатньо інформації, щоб при наявності необхідних навичок змінити логіку роботи застосунку без доступу до початкового коду. Іноді для написання певних частин застосунку використовують мови C/C++, для яких провести реверс-інжиніринг набагато складніше, адже C/C++ одразу компілюється в машинний код, що не зберігає жодної інформації про назви змінних, функцій та класів. Важливо також, що машинний код не містить умовних операторів та циклів, що є додатковою перешкодою для проведення реверс-інжинірингу, адже потрібно будувати граф потоку керування програмою.

2.2 Цілі реверс-інжинірингу

Загалом реверс-інжиніринг використовується для багатьох цілей :

1. Пошук вразливостей та/або шкідливих функцій в застосунку.

2. Відновлення початкового коду застосунку, якщо останній було втрачено.
3. Навчання. Проводячи зворотню розробку готового застосунку, можна покращити свої навички звичайної розробки. Виявлені архітектурні та структурні підходи, рішення цікавих проблем можуть стати в пригоді в майбутньому. Цікавий та корисний спосіб використання оберненої розробки для навчання наведено в роботі [10] : реверс-інжиніринг використовується для побудови UML діаграм.

Однак реверс-інжиніринг найчастіше проводять не з такими добрими намірами. Негативними прикладами застосування реверс-інжинірингу є :

1. Отримання прибутку від реклами. Існуюча реклама заміняється на рекламу зловмисників.
2. Видалення реклами із застосунку.
3. Модифікація застосунку шкідливими функціями : шпигунство, крадіжка та шифрування даних, тощо.
4. Безкоштовні версії платних застосунків.
5. Отримання інформації про взаємодію з серверною частиною застосунку для подальших атак.
6. Аналіз дозволів застосунку та його компонентів із метою виявлення слабких місць для крадіжки даних через інші застосунки на простої.

Таким чином, зловмисний реверс-інжиніринг має наступні наслідки для розробників застосунку :

1. Зменшення прибутку.
2. Крадіжка інтелектуальної власності.
3. Репутаційні збитки для компанії.

Реверс-інжиніринг буває двох типів : статичний та динамічний. Під час статичного аналізу, зловмисник аналізує код без запуску застосунку.

Динамічний реверс-інжиніринг включає в себе запуск та взаємодію із застосунком із метою аналізу його функціональності.

2.3 Статичний реверс-інжиніринг

Під час статичного реверс-інжинірингу зловмисник намагається отримати вихідні коди програми зі скомпільованого готово застосунку без запуску останнього. Застосунки для ОС Android поширюються в архівах APK, що містять ресурси, скомпільований код, маніфест, підпис і т.д. Уся логіка застосунку зберігається в .dex файлах. Першочерговим завданням зловмисника є зрозуміти логіку застосунку та за необхідності змінити її. Dalvik байт-код призначений для віртуальної машини, а не для людини, тому для останньої було створено smali / baksmali, що є асемблером / дизасемблером для Dalvik байт-коду.

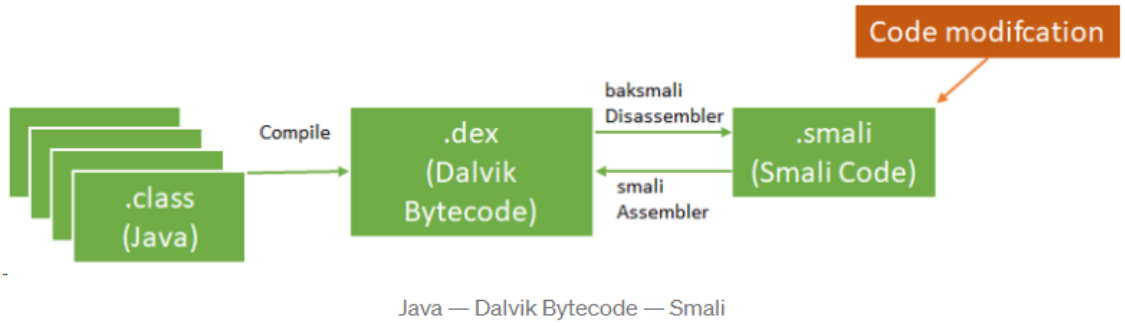

Рисунок 2.1 – Java – Dalvik Bytecode – Smali [11]

Якщо необхідно провести модифікацію функціональності застосунку, то це потрібно робити в .smali файлах (рисунок 2.1), адже неможливо отримати повністю працюючий вихідний код застосунку мовою Java із .dex файлів.

Інструменти для статичного реверс-інжинірингу

1. Apktool – програма, що використовується для декомпіляції та рекомпіляції мобільних застосунків. На вхід програма отримує APK файл, а на вихід видає всі ресурси застосунку, файл маніфесту та .smali

файли. Після внесення необхідних змін у .smali файли, застосунок можна рекомпілювати та отримати модифікований APK.

2. Dex2Jar / JadX / ByteCode Viewer – програми, що дозволяють отримати Java код із .dex файлів. Отриманий Java код буде не ідеальним, адже більшість розробників використовують базові засоби боротьби з реверс-інжинірингом, такі як, наприклад, обфускація. Проте отриманий код мовою програмування Java все одно зручніше аналізувати, ніж мовою Smali. Таким чином, зловмисник аналізує логіку застосунку мовою Java, а модифікації проводить у .smali файлах.

2.4 Динамічний реверс-інжиніринг

Динамічний реверс-інжиніринг включає запуск застосунку на емуляторі чи реальному смартфоні, пошук та аналіз вразливостей та можливу модифікацію роботи застосунку.

Справа в тому, що деякі шкідливі функції та вразливості, такі як : небезпечне зберігання даних користувача; шифрування даних пристрою, складно виявити під час статичного аналізу. До того ж часто статичний аналіз ускладнюють:

1. Застосування різноманітних обфускаторів.
2. Динамічне завантаження коду.
3. Динамічне дешифрування стрічок під час виконання застосунку.

Кожен застосунок перед публікацією в Google Play проходить через динамічний аналіз : програму запускають у віртуальному середовищі та спостерігають за її виконанням. Якщо застосунок містить шкідливі функції, то його подальша публікація неможлива.

2.5 Стандарт захисту мобільних застосунків

На момент написання кваліфікаційної роботи головним стандартом захисту мобільних застосунків є MASVS [12] (Mobile Application Security Verification Standart), розроблений компанією OWASP (Open Worldwide Application Security Project). Загалом компанія займається безпекою веб-застосунків, проте має підрозділ пов’язаний саме з безпекою мобільних застосунків - MAS (Mobile Application Security). Цей стандарт визнаний Google Android, CREST, Національним інститутом стандартів і технологій (США), Федеральним управлінням з інформаційної безпеки (Німеччина), тому вартий уваги та довіри.

Стандарт MASVS складається з трьох рівнів :

1. MASVS – L1 : Стандартна безпека. Перший рівень безпеки мають реалізовувати всі без виключень мобільні застосунки. Він пов’язаний з безпечним зберіганням чутливої інформації та загалом дотримання найкращих практик безпеки мобільних застосунків.
2. MASVS – L2 : Поглиблена безпека. Другий рівень є розширенням першого та слугує для застосунків, що працюють із дуже чутливою інформацією, для яких безпека є першочерговим завданням. Додатки пов’язані з документообігом, банківською та медичними справами є прикладами таких застосунків.
3. MASVS – R : Протидія реверс-інжинірингу та підробкам. Даний рівень жодним чином не є заміною двом попереднім : він націлений на створення додаткового захисту застосунків, що вже відповідають двом попереднім рівням стандарту.

У контексті даної кваліфікаційної робити цікавим є лише MASVS – R, адже я розробляю застосунок для боротьби саме з реверс-інжинірингом, а не з усіма можливими видами атак та вразливостей.

Таблиця 2.1 - Вимоги стандарту MASVS – R

Номер	MSTG-ID	Опис
1	MSTG-RESILIENCE-1	Застосунок припиняє роботу чи повідомляє користувача при виявленні прав суперкористувача.
2	MSTG-RESILIENCE-2	Застосунок запобігає відлагодженню та/або виявляє та реагує на наявність відлагоджувача. Усі доступні протоколи відлагодження мають бути враховані.
3	MSTG-RESILIENCE-3	Застосунок виявляє та реагує на модифікацію виконуваних файлів.
4	MSTG-RESILIENCE-4	Застосунок виявляє та реагує на наявність широко використовуваних інструментів реверс-інжинірингу на мобільному пристрої.
5	MSTG-RESILIENCE-5	Застосунок виявляє та реагує на виконання на емуляторі.
6	MSTG-RESILIENCE-6	Застосунок виявляє та реагує на модифікацію свого коду та даних в оперативній пам’яті.

7	MSTG-RESILIENCE-7	Застосунок реалізує декілька механізмів захисту для кожної категорії (1 – 6). Важливо, що на стійкість до реверс-інжинірингу впливає кількість, різноманітність та оригінальність використовуваних засобів.
8	MSTG-RESILIENCE-8	Захисні механізми виявлення мають бути прихованими, відкладеними та різноманітними.
9	MSTG-RESILIENCE-9	Обфускація застосована до механізмів, що протидіють динамічному реверс-інжинірингу.
10	MSTG-RESILIENCE-10	Застосунок реалізує прив’язку до пристрою шляхом формування та порівняння підписів.
11	MSTG-RESILIENCE-11	Усі виконувані файли застосунку зашифровані. Статичний реверс-інжиніринг не дозволяє отримати важливу інформацію.
12	MSTG-RESILIENCE-12	У випадку, якщо першочерговою метою обфускації є захист якихось важливих алгоритмів, то вона повинна захищати від ручного тестування, автоматизованих деобфускаторів та враховувати актуальні тематичні дослідження.

2.6 Засоби протидії реверс-інжинірингу

У таблиці 2.1 наведено вимоги стандарту MASVS-R про те, що потрібно зробити для того, щоб захистити мобільний застосунок від реверс-інжинірингу. У цьому розділі я би хотів приділити увагу : чому та як потрібно реалізовувати вимоги стандарту? Даний розділ не має лістингів коду про імплементацію конкретних методів протидії реверс-інжинірингу, адже, по-перше, головною метою моєї кваліфікаційної роботи є саме розробка застосунку-шифрувальника для підвищення захисту мобільних додатків від статичного реверс-інжинірингу, і цей розділ має на меті більш дослідницький характер, а, по-друге, відповідно до стандарту MASVS-R кожен розробник мобільного додатку має самостійно більш-менш унікально реалізовувати наведені методи, адже скопійовані рішення зі StackOverflow чи інших ресурсів буде легше знайти в коді та модифікувати реверс-інженеру.

Методи протидії реверс-інжинірингу :

1. Виявлення та блокування виконання застосунку на пристрої із правами суперкористувача (MASVS-R1).

   Чому?

   Суперкористувачі можуть переглядати та редагувати будь-які файли на пристрої : як системні так і всі файли застосунків, що інстальовані на пристрої. Загалом кожен додаток може зберігати дані у внутрішньому сховищі пристрою, до якого не має доступу ніхто, крім розробника цього застосунку. Однак, якщо власник пристрою має права суперкористувача, жодні системні захисти не працюють, і користувач може отримати доступ до всіх файлів пристрою. Вимогою MASVS-R1 є як раз блокування роботи застосунку на пристрої, що має права суперкористувача.

Як?

- Play Integrity API[13] – допомогає визначити : чи застосунок виконується на звичайному пристрої без прав суперкористувача, чи застосунок завантажений з Google Play, чи початковий код застосунку не було редаговано.
- Перевірка наявності файлів, характерних для пристроїв, що мають права суперкористувача. Наприклад, рутовані пристрої містять директорію su, яка може знаходитись за різними системними шляхами, такими як :
  /system/xbin/busybox
  /system/sd/xbin/su
  /system/bin/su
  /sbin/su
  /system/xbin/su
  /data/local/su
  /system/bin/failsafe/su
  /data/local/bin/su
  /data/local/xbin/su
- Перевірка можливості редагування системних файлів. На звичайному пристрої неможливо редагувати системні файли. Якщо така можливість є – це свідчить про наявність на пристрої прав суперкористувача.

Важливо, що всі ці методи мають бути розміщені в різних частинах застосунку задля ускладнення пошуку та нейтралізації методів захисту реверс-інженером, тобто це не може бути якась бібліотека або один

клас. Методи перевірки можна також реалізовувати мовами C/C++, що додатково збільшить складність для реверс-інженера.

2. Виявлення та блокування роботи застосунку при виявлені відлагоджувача (MASVS-R2).

   Чому?

   Динамічний аналіз роботи застосунку реверс-інженери також проводять за допомогою відлагоджувачів, які дозволяють виконувати програму крок за кроком, аналізувати та модифікувати значення змінних.

   Як?

   - Перевірка значення ApplicationInfo.FLAG_DEBUGGABLE, яке завжди має бути false для APK, що завантажуються в Google Play.
   - Перевірка за допомогою системної функції android.os.Debug. isDebuggerConnected().

3. Виявлення на блокування роботи застосунку при виявленні модифікації виконуваних файлів (MASVS-R3).

   Чому?

   Звичайний користувач не буде редагувати виконувані файли застосунку, а отже, щоб ускладнити процес реверс-інжинірингу для зловмисника, потрібно виявляти та блокувати роботу застосунку при виявленні будь-яких модифікацій.

   Як?

   Найчастіше перевірка цілісності файлів відбується за допомогою порівняння хешу або контрольної суми для вибраних файлів. Перевіряти всі файлі застосунку може бути витратно, тому загалом перевіряють лише AndroidManifest.xml та файли, які містять критичну

для роботи застосунку логіку, у тому числі реалізації методів для протидії реверс-інжинірингу.

4. Виявлення та блокування роботи застосунку при виявленні інструментів для проведення реверс-інжинірингу, таких як Frida, Xposed та Substrate (MASVS-R4).

   Як?

   Засоби реверс-інжинірингу постійно розвиваються, тому потрібно стежити за актуальними способами виявлення таких інструментів. Інструмент Frida, наприклад, можна виявити по відкритому TCP порту 27042. Часто такі програми вимагають наявності девайсу з правами суперкористувача, то ж якщо мобільний застосунок виявляє та реагує на рутований девайс, то інструмент для реверс-інжинірингу автоматично не буде працювати. Аналогічна ситуація з використанням відлагоджувачів.

5. Виявлення та блокування роботи застосунку на емуляторі (MASVS-R5).

   Чому?

   Звичайні користувачі рідко коли використовують емулятор, а ось зловмисники часто, адже це значно простіше, зручніше та дешевше за використання справжнього пристрою.

   Як?

   Виявлення чи застосунок виконується на емуляторі полягає в перевірці деяких системних значень, до яких належать : Build.DEVICE, Build.MANUFACTURER та TelephonyManager.getDeviceId().

6. Використання декількох різноманітних механізмів захисту з MASVS-R1 по MASVS-R6 ускладнює реверс-інжиніринг, адже зловмиснику необхідно буде знайти в різних частинах застосунку код,

відповідальний за захист та змінити його. Чим більше методів та чим вони різноманітніші – тим складніше буде реверс-інженеру.

7. Використання обфускації коду (MASVS-R9).

Чому?

Обфускація – це процес модифікації початкового коду, при якому останній залишається повністю функціональним, проте складним для сприйняття людьми. Код стає важко читати, адже всі назви класів, інтерфейсів, констант та змінних перетворюються на беззмістовний набір символів.

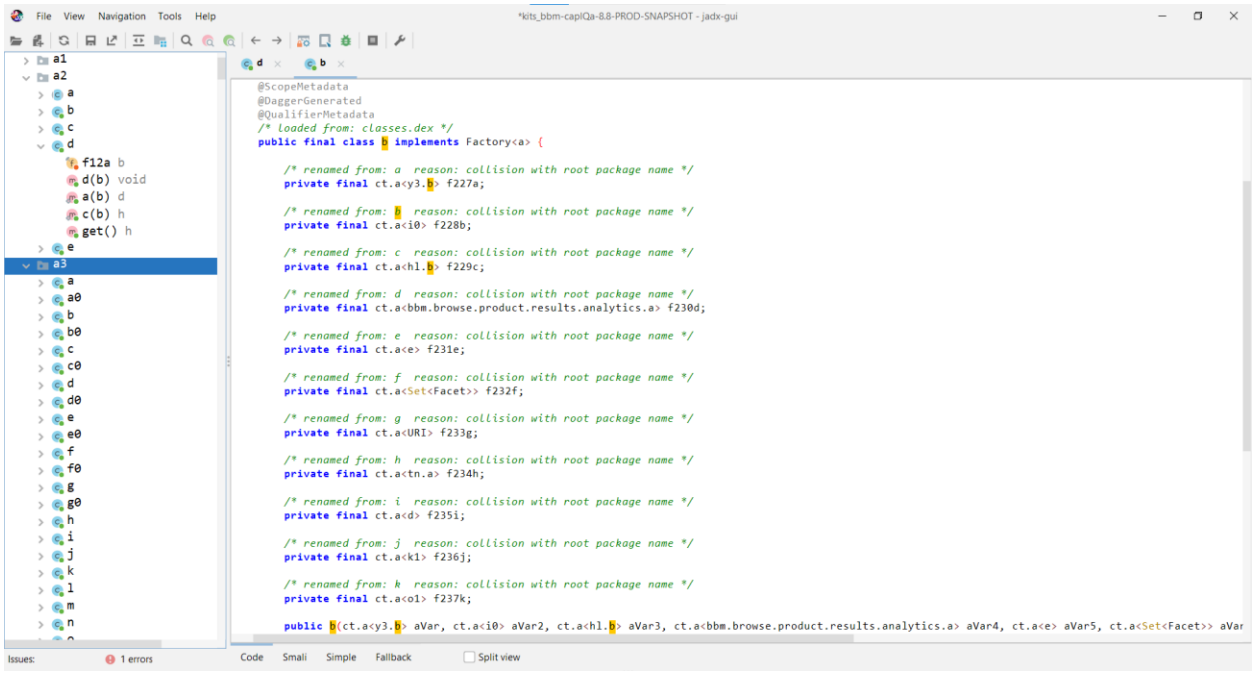

Рисунок 2.2 – Приклад обфускованого коду

Найчастіше для протидії реверс-інжинірингу програмісти використовують саме цей метод через його простоту та ефективність.

Як?

- ProGuard – інструмент, що використовується для обфускації, оптимізації та видалення невикористовуваного коду мовами Java, Kotlin та іншими JVM мовами. Постачається разом із середовищем розробки Android Studio.
- DexGuard – є покращеною платною версією ProGuard. Якщо головною метою ProGuard є оптимізація та видалення невикористовуваного коду, то DexGuard націлений саме на захист застосунків від реверс-інжинірингу. Він включає в себе шифрування, декілька видів обфускації, виявлення прав суперкористувача та перевіряє наявність програм для проведення реверс-інжинірингу.

8. Шифрування логіки застосунку для протидії статичному реверс-інжинірингу (MASVS-R11).

   Чому?

   Якщо обфускація ускладнює реверс-інжиніринг, то шифрування логіки застосунку унеможливлює проведення статичного реверс-інжинірингу, адже логіка застосунку розшифровується лише під час виконання застосунку.

   Як?

   Практична частина моєї кваліфікаційної роботи присвячена саме розробці такого додатку, який дозволив би шифрувати dex файли, що зберігають всю логіку роботи застосунку. Алгоритм детально описаний в розділі 3.1.

9. Важливий код має бути розміщений на сервері.

Чому?

Якщо код містить високу інтелектуальну цінність, то він обов’язково має бути розміщений на сервері. Зловмисник має доступ лише до коду, що виконується на пристрої, то ж переміщення критичної логіки на сервер унеможливить реверс-інжиніринг.

3. Розробка застосунку для захисту від статичного реверс-інжинірингу

3.1 Опис методу

Застосунки для ОС Android поширюються в форматі APK та зберігають всю логіку застосунку в dex файлах. За допомогою перелічених інструментів зловмисник може отримати наближений до початкового Java код із dex файлів. Зрозуміло, що задача кожного розробника – максимально ускладнити цей процес аби зберегти свою інтелектуальну власність та запобігти втраті чутливої інформації.

Одним із методів такого ускладнення є використання так званих пакувальників [14]. Ідея полягає в тому, щоб зашифрувати логіку застосунку, а під час виконання на пристрої користувача – розшифрувати її. Таким чином, зловмисник, що хоче провести статичний реверс-інжиніринг не зможе це зробити існуючими інструментами та буде вимушений докладати додаткових зусиль. Як я вже зазначив, не існує методу, який би захистив застосунок на 100% від реверс-інжинірингу, адже потенційний зловмисник має фізичний доступ до пристрою на якому виконується застосунок. Однак, використання даного та/чи інших методів значно скорочує коло реверс-інженерів, адже вимагає від останнього додаткових навичок, аніж просте використання автоматизованих інструментів.

3.2 Алгоритм пакувальника

1. На вхід подається APK файл та JKS файл для повторного підпису застосунку, який потрібно зашифрувати.
2. Розархівування APK файлу.

3. Модифікація AndroidManifest.xml : підміна оригінального Application [15] класу застосунку на дешифратор та зберігання шляху до первинного Application класу в метаданих маніфесту.
4. Шифрування всіх dex файлів застосунку.
5. Додавання логіки дешифрування до APK файлу.
6. Рекомпіляція застосунку.
7. Повторний підпис застосунку.

Логіка дешифрування dex файлів зберігається в окремому Application класі. Для розуміння назвімо його DecryptApplication. Кожен застосунок містить AndroidManifest.xml, який містить дані про Application клас цього додатку. Application – це клас, із якого починається виконання всіх застосунків для операційної системи Android.

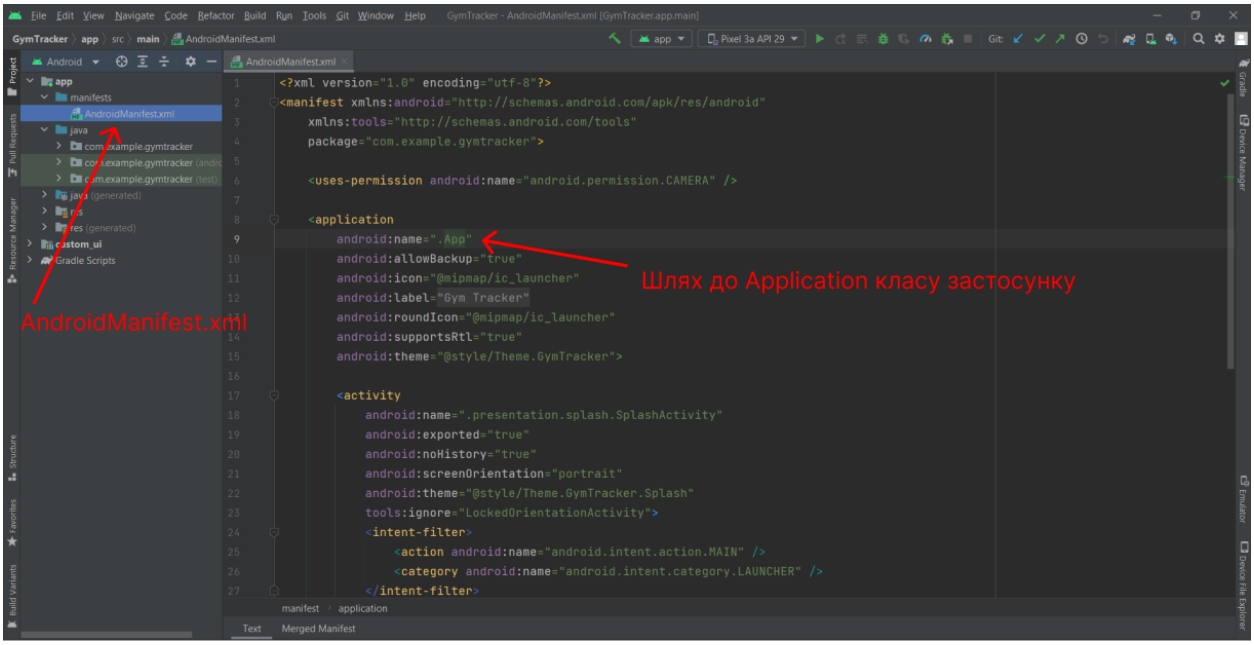

Рисунок 3.1 - Application клас в AndroidManifest.xml на прикладі реального застосунку

Після виконання алгоритму AndroidManifest.xml має мати наступний вигляд.

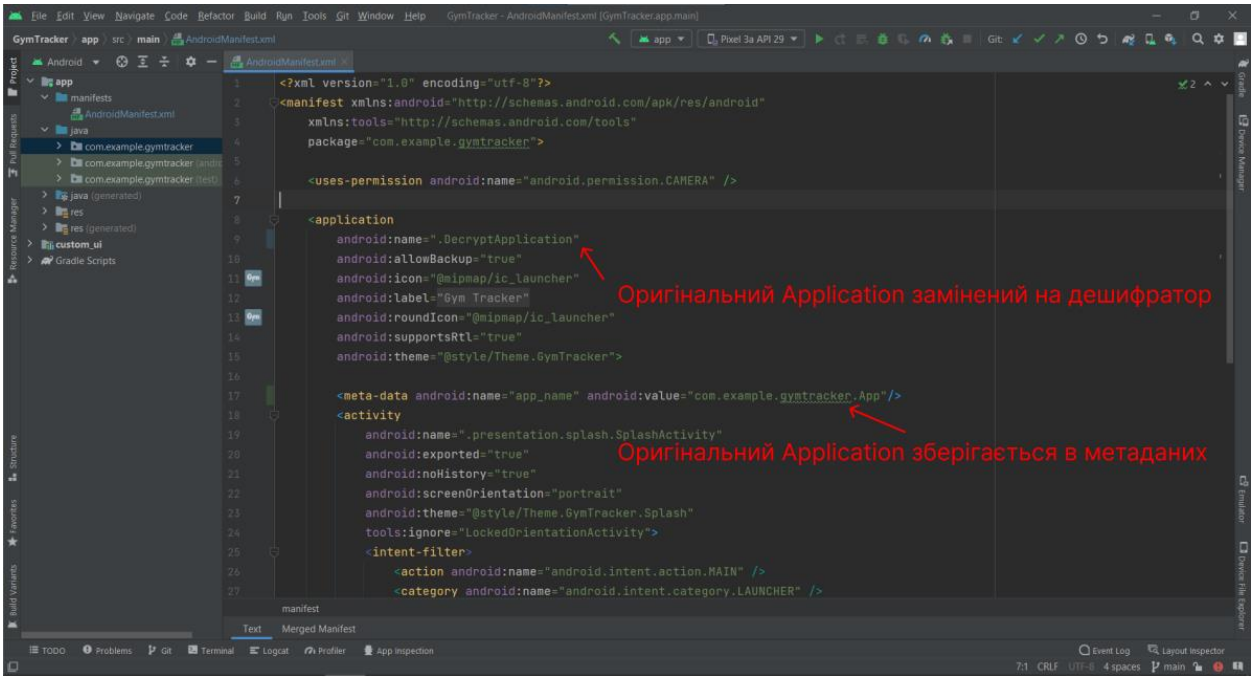

Рисунок 3.2 - AndroidManifest.xml файл після виконання алгоритму

Під час виконання застосунку на пристрої DecryptApplication розшифровує та зберігає у внутрішнє сховище девайсу всі dex файли. Після цього DecryptApplication завантажує розшифровані dex файли в оперативну пам’ять пристрою та передає виконання із використанням засобів рефлексії оригінальному Application класу, який він отримує з метаданих AndroidManifest.xml.

3.3 Аналіз існуючих методів вирішення поставленої задачі

Статей та науковий робіт українською чи англійською мовами мені знайти не вдалося : існуючі часткові рішення поставленої задачі були представлені лише на GitHub. Варто також зазначити, що всі знайдені мною рішення на GitHub були створені розробниками з Китаю : про це свідчать коментарі в коді та опис їх проєктів. Отже, ми можемо зробити висновок, що наразі серед англомовних та україномовних розробників ще ніхто поставлену задачу не реалізував.

Загалом я проаналізував 13 рішень поставленої задачі. Рішення [16-26] майже ідентичні та мають спільні наступні недоліки :

1. Усі dex файли після шифрування поєднуються в один dex файл, що включає в себе логіку дешифрування. Згідно з офіційною документацію для ОС Android[27] один dex файл може містити до 65536 методів.

When your app and the libraries it references exceed 65,536 methods, you encounter a build error that indicates your app has reached the limit of the Android build architecture:

```
trouble writing output:
Too many field references: 131000; max is 65536.
You may try using --multi-dex option.
```

Older versions of the build system report a different error, which is an indication of the same problem:

```
Conversion to Dalvik format failed:
Unable to execute dex: method ID not in [0, 0xffff]: 65536
```

Рисунок 3.3 – Обмеження на 65536 методів для одного dex файлу [27]

Може скластись враження, що дана проблема не є актуальною для більшості Android застосунків, адже 65536 методів – це досить багато, проте dex файли включають в себе також методи бібліотек, які використовуються в застосунку, тому, насправді, для більшості сучасних популярних Android застосунків існує потреба у використанні декількох dex файлів - multidex.

2. Недостатня автоматизація та зручність використання. Розробнику для того, щоб захистити свій застосунок необхідно клонувати проєкт та відредагувати початковий код : змінити шлях до APK файлу, змінити пароль та шлях до JKS файлу. Для більшості розробників це не буде складно зробити, проте все одно це є недоліком, адже потрібно витрачати час на ознайомлення з вихідним кодом.
3. Більшість готових рішень не містить документації або просто навіть опису : як цим користуватись? Як я вже зазначив, проєкти створені китайськими розробниками, тому коментарі в коді та опис також написані китайською мовою, що значно ускладнює розуміння для англомовних людей. Якщо для опису проєкту на GitHub ще зручно використовувати засоби автоматичного перекладу, такі як Google Translate, то ось коментарі в коді доведеться перекладати один за одним. Додамо до цього проблеми з автоматичним перекладом китайської на англійську й отримуємо досить суттєвий недолік.
4. У середньому останні зміни в проєктах проводились 5 років назад. Можемо припустити, що існуючі рішення є покинутими. Розробникам, які регулярно публікують нові версії застосунків необхідні надійні рішення, адже деякі залежності в проєктах-пакувальниках могли чи можуть застаріти більш ніж за 5 років.

5. Не підтримує роботу з AAPT2 (Android Asset Packaging Tool)[28]. Це суттєво скорочує кількість застосунків, які можна запакувати, адже більшість сучасних застосунків використовують, наприклад, Android Jetpack Navigation Component[29], який вимагає використання саме AAPT2.

Серед проаналізованих рішень відрізняється[30]. Проєкт на відміну від інших:

- було розроблено у вересні 2022 року
- підтримує роботу з multidex
- містить документацію хоча б китайською мовою

Однак, додатково до всіх інших спільних недоліків із попередніми рішеннями(AAPT2 та недостатня зручність використання), цей проєкт має ще один : не містить логіки дешифрування. Наскільки я зрозумів із опису, програма отримує на вхід для дешифрування Android Archive(AAR), конвертує його в JAR, а потім у dex. Знайти інформацію щодо доцільності цього рішення мені не вдалося, проте так як логіка дешифрування завжди одна, то логічніше та швидше одразу зберігати файл дешифрування у форматі dex.

Іншим рішенням для аналізу стало[31]. Розробник з Китаю створив пакувальник у вигляді Android додатку та завантажив його у Google Play.

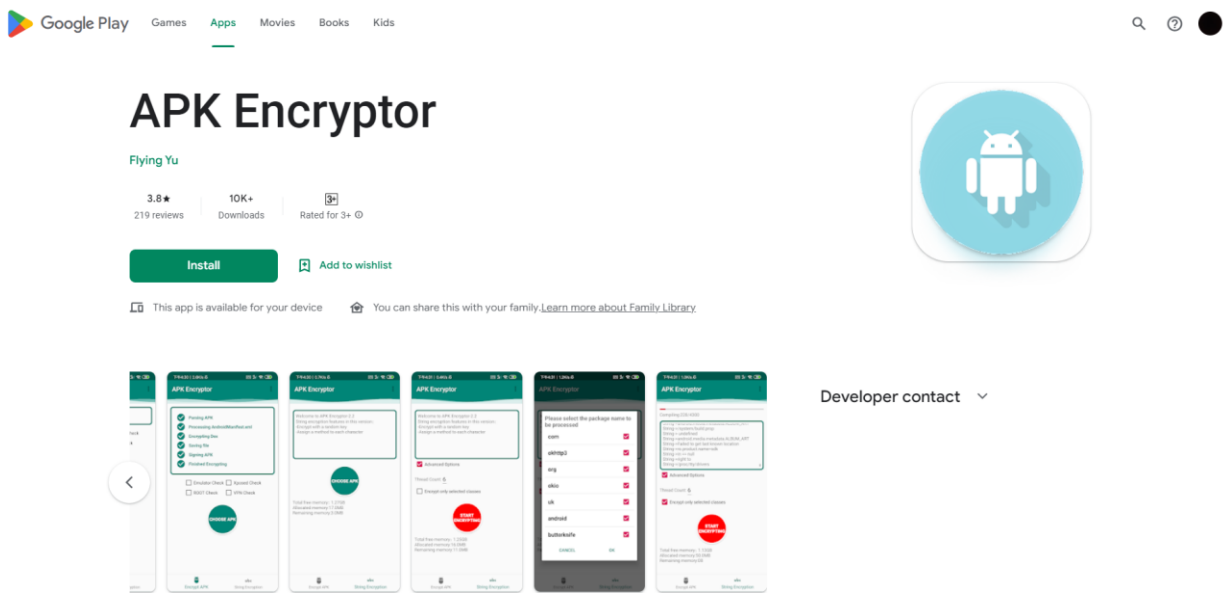

Рисунок 3.4 - Приклад Android застосунку-пакувальника у Google Play

Хоча застосунок має понад 10 тисяч завантажень, особисто мені, як Android розробнику, було б незручно ним користуватись : публікація застосунку в Google Play відбувається найчастіше з комп’ютера, а не з мобільного пристрою, але, щоб опублікувати додаток, користуючись наведеним рішенням, потрібно спочатку надіслати його на телефон, там зашифрувати, а потім знов перенести на комп’ютер. Набагато зручніше в даному випадку був би веб чи десктоп застосунок.

Ще одним недоліком є те, що пакування здійснюється не на клієнті, а на сервері. Відправлення паролю та підпису до свого застосунку на якийсь невідомий сервер є небезпечним, адже ніхто не гарантує, що даний розробник не виявиться зловмисником, який, додатково отримавши ваші дані від облікового запису Google Play, зміг би видалити ваш застосунок, завантажити свій додаток замість вашого або публікувати свої застосунки від вашого імені.

Отже, відповідно до мого дослідження, станом на час його проведення не існує готових рішень, які б поєднували в собі підтримку роботи з multidex, підтримку роботи з AAPT2, містили б зрозумілі вихідні коди та опис для англомовних чи україномовних людей та були достатньо автоматизованими та зручними у використанні.

3.4 Опис застосунку

3.4.1 Ціль

На основі аналізу недоліків, які мали рішення поставленої задачі я вирішив розробити застосунок, який би дозволяв кожному Android розробнику підвищити рівень захисту свого додатку від статичного реверс-інжинірингу. Мій застосунок мав вирішувати всі існуючі проблеми, а саме :

1. Зручність та достатня автоматизація, щоб заощадити час Android розробникам, які будуть його використовувати. Існуючі рішення не надають розробникам готового до використання продукту.
2. Підтримка роботи з mutlidex та AAPT2, що є невід'ємними складовими майже кожного популярного додатку.
3. Прозорість. Кожен розробник має мати можливість за необхідності ознайомитись із вихідним кодом та деталями реалізації, щоб чітко усвідомлювати : який рівень захисту від статичного реверс-інжинірингу надає використання застосунку.

3.4.2 Реалізація

Попередній аналіз існуючий рішень показав, що використання Android додатку для пакування інших застосунків не є зручним, тому я вирішив написати десктоп версію пакувальника.

Набір інструментів :

1. Середовища розробки IntelliJ IDEA та Android Studio. Останнє використовувалось для написання логіки дешифрування та отримання файлу classes.dex.
2. Compose Desktop[32]. Я обрав цей фреймворк для відображення користувацького інтерфейсу, адже він надає можливість

використовувати декларативний підхід для написання логіки відображення під операційні системи Windows, Linux та macOS. Сьогодні декларативний стиль програмування набуває все більшої популярності, адже він дозволяє писати менше коду та пришвидшує розробку програмного продукту. Відомими фреймворками є також JetPack Compose, React Native, SwiftUI та Flutter, які є повною протилежністю Swing та JavaFX, які використовують імперативний стиль програмування.

3. Kotlin Coroutines [33]. Корутина – це блок коду, що може виконуватись паралельно з іншим кодом та призупиняти своє виконання. Корутина відрізняється від звичайного потоку тим, що потік може містити багато корутин, які не прив’язані до нього та можуть призупиняти своє виконання. При написанні застосунків із користувацьким інтерфейсом кожен розробник так чи інакше стикається з асинхронністю та/чи паралельними обчисленнями. Пакування застосунку займає певний час, тому, щоб користувач міг бачити прогрес виконання, процес пакування я вирішив зробити асинхронним. Бізнес-логіка та логіка відображення написані мовою Kotlin, тому для мене вибір Kotlin Coroutines став очевидним. Існують й інші рішення для забезпечення асинхронності та паралельних обчислень, такі як, наприклад, RxJava[34], проте Kotlin Coroutines мають простіші API і оператори та загалом більше підходять для мого проєкту.
4. Я використовую Apktool для декомпіляції та рекомпіляції APK, який потрібно запакувати. На вихід з програми я отримую готовий до редагування AndroidManifest.xml. Після внесення необхідних змін проводиться рекомпіляція APK.

5. Apksigner[35] - стандартний інструмент для підпису Android застосунків від компанії Google. Використовувався для повторного підпису модифікованого застосунку.
6. Koin[36]. Ін'єкцію залежностей використовують для того, щоб код було легко підтримувати, читати, тестувати, модифікувати та перевикористовувати. У своєму застосунку я реалізував цей механізм за допомогою бібліотеки Koin (від Kotlin Injection). Іншою популярною бібліотекою є Dagger[37], проте він більше підходить для багатомодульних великих проєктів, ніж для мого застосунку.
7. MVI (Model-View-Intent). Я використав MVI архітектуру для побудови свого застосунку, адже вона, як на мене, на сьогоднішній день є найбільш зручною для написання додатків за допомогою Compose Desktop фреймворку.

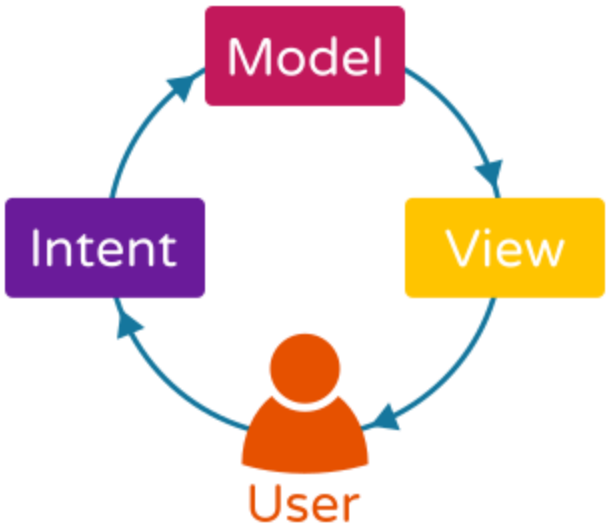

Рисунок 3.5 – Схематичне зображення MVI-архітектури [38]

Intent – сутність, що являє собою намір користувача. Наприклад, натискання користувача на кнопку, щоб запакувати APK – це його намір.

Model – відповідає за обробку намірів користувача(intents). На вхід отримує Intent від користувача, на вихід видає певний стан застосунку, який має відобразити View.

View – відповідає за реакцію на дії користувача та відображення отриманого від Model стану.

Характерними для MVI є також такі сутності як : reducer (відповідає за створення нового стану) та state (незмінний об’єкт, що містить необхідні для відображення дані).

8. Material Design[39] – це дизайн система, яку розробила компанія Google у 2014 році. Я хотів, щоб мій застосунок не лише був написаний із використанням останніх технологій, а й мав сучасний вигляд, тому вирішив використати цю дизайн систему.

Структура проєкту

Рисунок 3.6 – Структура проєкту

Директорія *input* включає classes.dex файл, який містить логіку дешифрування.

Директорія *libs* включає :

- apksigner.jar – для повторного підпису запакованого APK.
- apktool_2_6_1.jar – декомпіляція та рекомпіляція APK.

Пакет di включає KoinModules.kt, що відповідає за ін’єкцію залежностей.

Пакет *domain* відповідає за бізнес-логіку застосунку та включає в себе :

1. ApkEncryptor – отримує на вхід APK із модифікованим AndroidManifest.xml файлом та шифрує всі dex файли APK.
2. ApkProcessor – відповідає за модифікацію AndroidManifest.xml :
    a. декомпілює отриманий APK
    b. додає метадані про оригінальний Application клас до маніфесту та змінює початковий Application клас на дешифратор
    c. рекомпілює APK
3. ApkSigner – відповідає за підпис рекомпільованого APK за допомогою apksigner.jar.
4. DexCreator – відповідає за додавання логіки дешифрування до APK.

Пакет ui відповідає за логіку відображення та включає в себе :

1. Main.kt – описує користувацький інтерфейс за допомогою фреймворку Compose Desktop та відповідає за взаємодію з користувачем.
2. MainReducer.kt – допоміжний клас для реалізації MVI архітектури. На вхід отримує старий стан застосунку та дію користувача, на вихід видає оновлений стан застосунку.
3. MainScreenEntity.kt – містить сутності MainScreenState та MainScreenUIEvent. Перша відповідає за стан головного екрану, остання – за події, які можуть відбутися в застосунку, наприклад, початок пакування застосунку, помилка та інші.
4. MainViewModel – зберігає стан, який відображається в Main.kt, та відповідає за взаємодію з бізнес-логікою застосунку.
5. StylingState – допоміжний клас, який відповідає за колір кнопок на головному екрані залежно від стану застосунку.

Пакет utils містить :

1. Constants – на момент написання кваліфікаційної роботи фреймворк Compose Desktop не підтримує локалізації, проте задля простішого додавання український мови користувацького інтерфейсу в майбутньому я вирішив зберігати всі строкові літерали та константи в одному місці.
2. Extensions.kt – зберігає функції-розширення (Kotlin Extension Functions[40])
3. FileUtils – містить методи для зручної роботи з файлами.
4. ZipUtils – інкапсулює логіку архівування та розархівування.

Директорія resources містить іконки, які відображаються в застосунку.

Логіка дешифрування :

Проєкт містить директорію *input*, яка включає файл classes.dex, що зберігає всю логіку дешифрування. Для того, щоб отримати цей файл, я створив окремий проєкт у середовищі розробки Android Studio (рисунок 3.7).

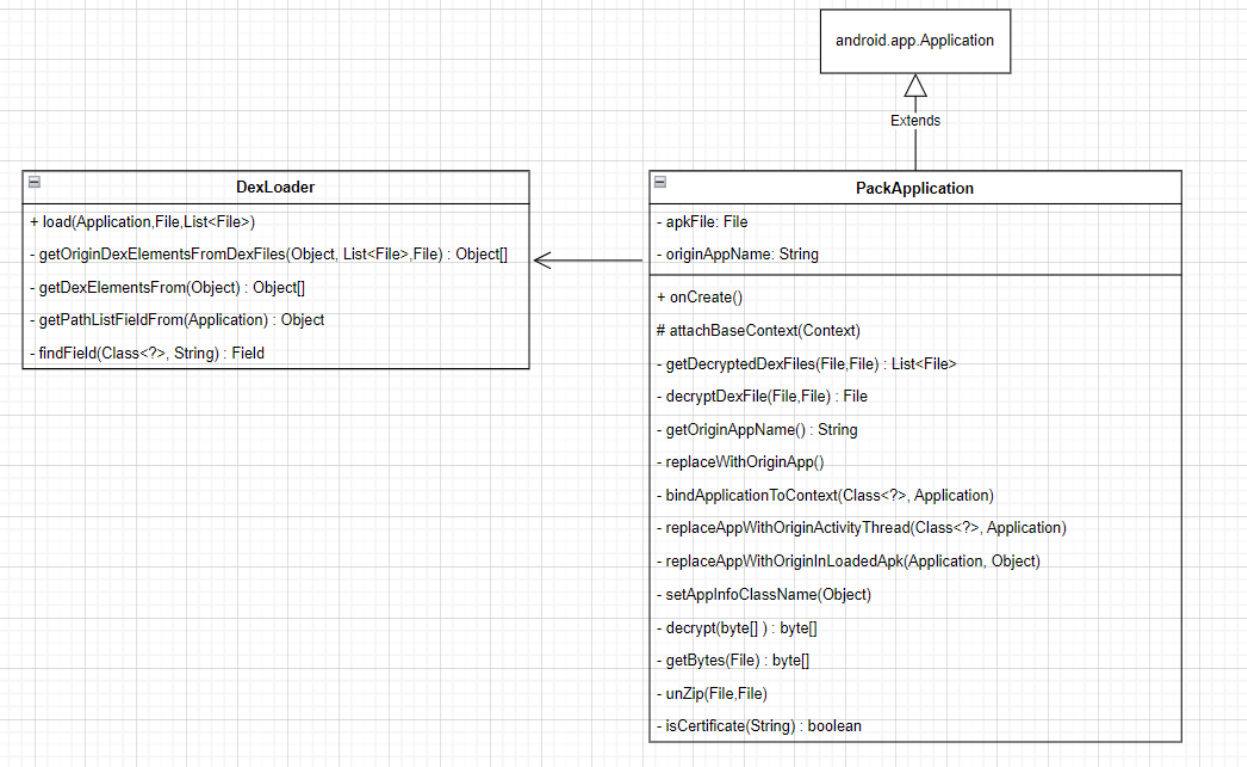

Рисунок 3.7 – Структура проєкту-дешифрувальника

Допоміжний проєкт складається з 2 класів :

1. DexLoader – відповідальний за завантаження дешифрованих dex файлів в оперативну пам'ять.
2. PackApplication – є нащадком системного класу Application. Цей клас дешифрує всі dex файли застосунку та, використовуючи DexLoader, завантажує їх в оперативну пам’ять. Після цього за допомогою засобів рефлексії передає виконання оригінальному Application класу, який містився в початковому APK.

Після компіляції допоміжного проєкту на виході отримуємо AAR(Android Archive) файл, який необхідно додатково скомпілювати, щоб отримати dex файл. Для цього я використав інструмент командного рядка d8[41], розроблений компанією Google.

3.4.3 Демонстрація

Для демонстрації роботи застосунку я обрав свою минулорічну курсову роботу : застосунок для Android телефонів для обліку та автентифікації відвідувачів спортивної зали.

Перед використанням застосунку користувач має заздалегідь підготувати APK файл, який необхідно запакувати та JKS файл, яким буде підписано зашифрований застосунок.

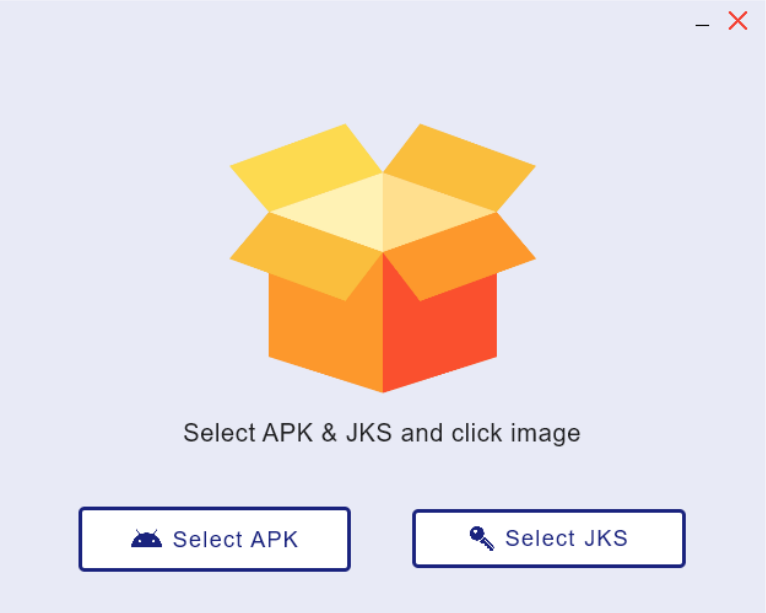

Рисунок 3.8 – Головний екран застосунку

На головному екрані застосунку користувач повинен вибрати APK та JKS.

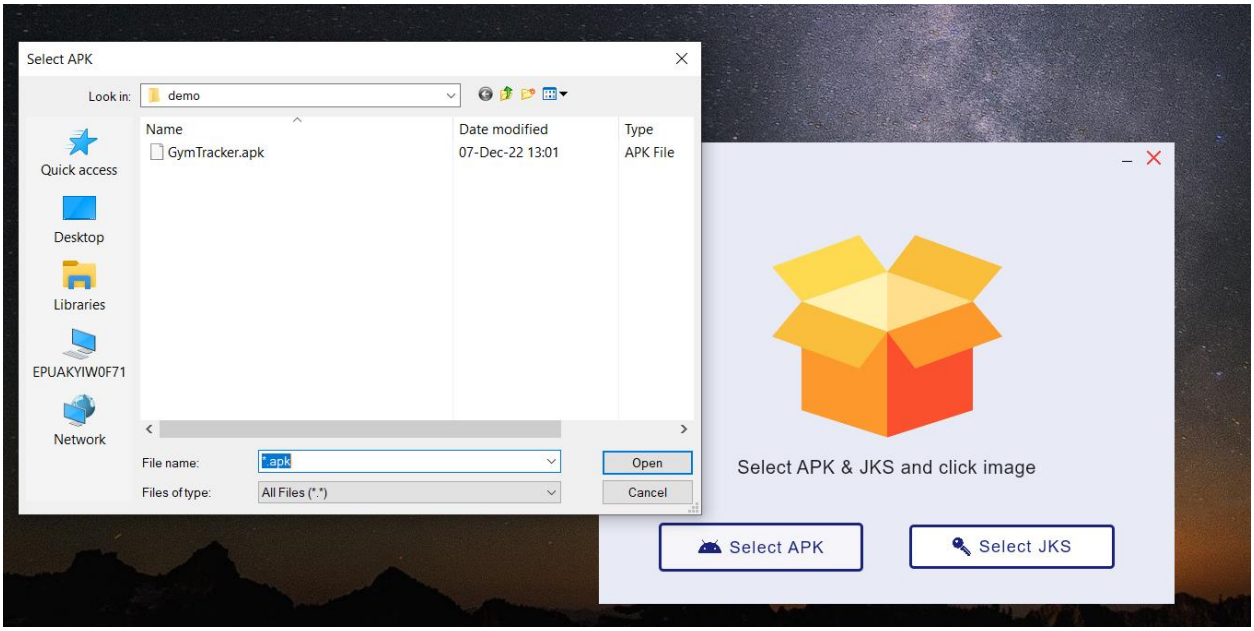

Рисунок 3.9 – Процес вибору APK

Після вибору APK колір кнопки змінюється, повідомляючи таким чином користувача про внесені зміни.

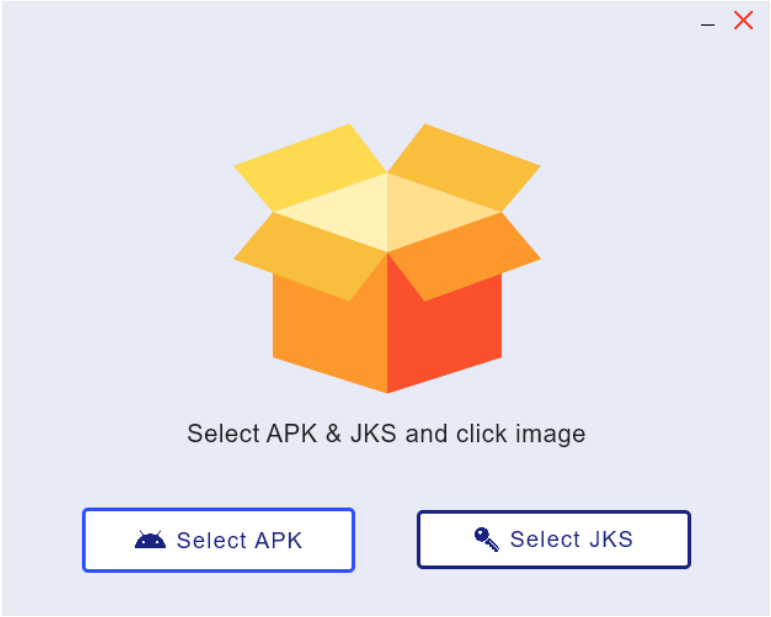

Рисунок 3.10 – Зміна користувацького інтерфейсу після вибору APK

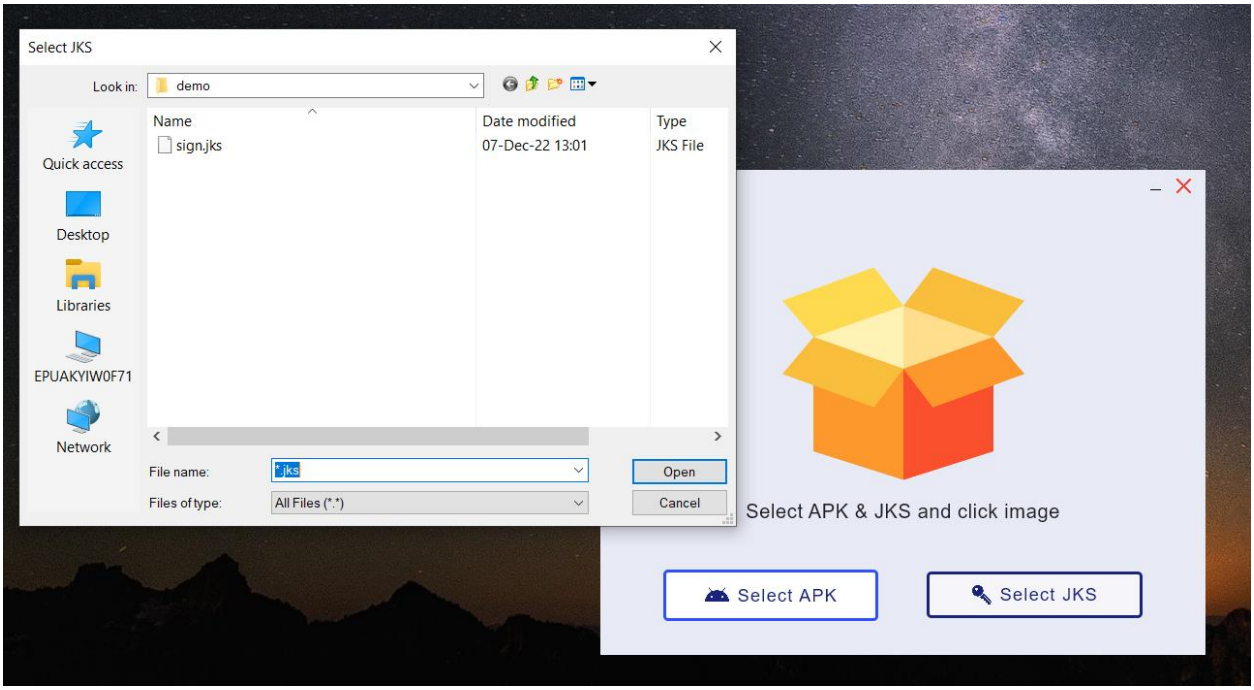

Рисунок 3.11 – Процес вибору JKS

Якщо користувач не обере JKS або APK файл та закриє вікно, він побачить повідомлення про помилку.

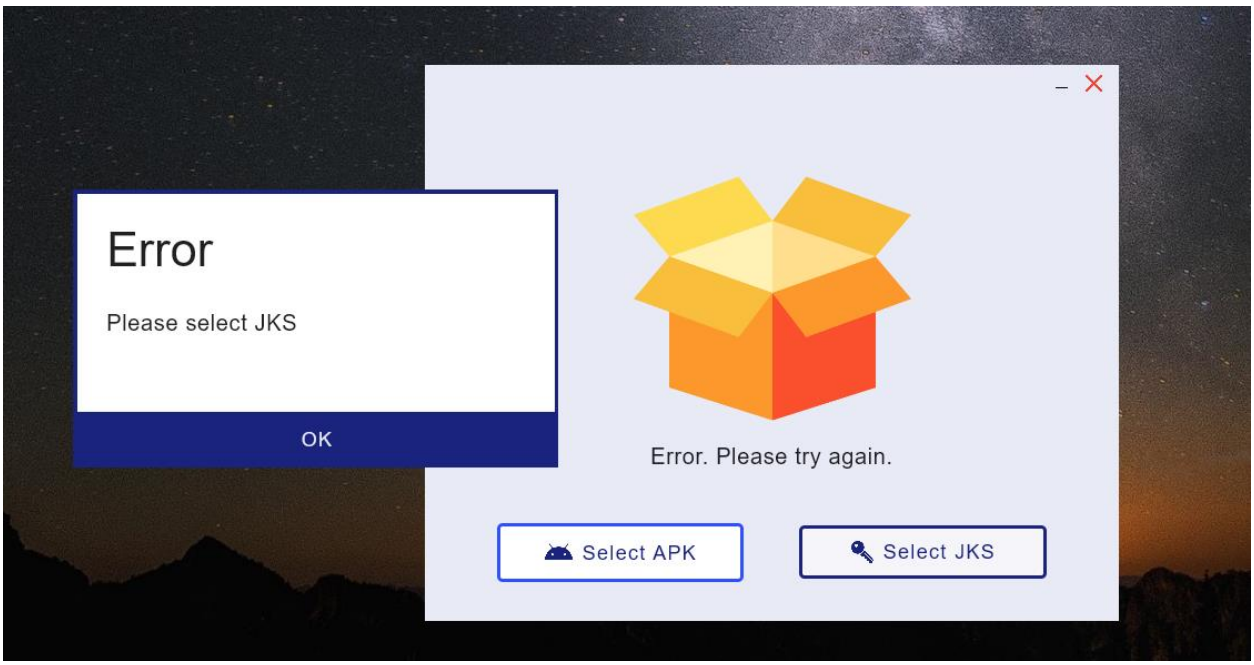

Рисунок 3.12 - Повідомлення про помилку при скасуванні вибору JKS

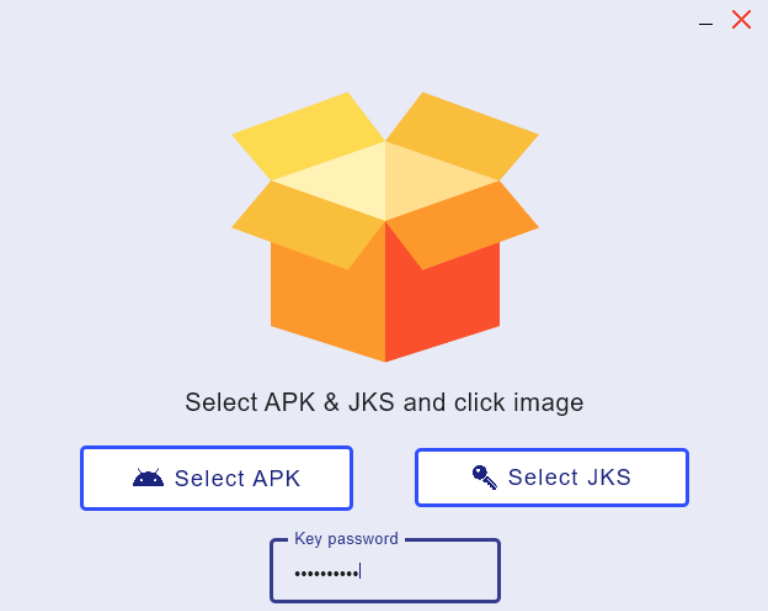

Рисунок 3.13 - Надання паролю користувачем для JKS

Після вибору APK та JKS користувач має надати пароль для JKS і натиснути за зображення коробки, щоб почати процес пакування.

Рисунок 3.14 - Процес пакування застосунку

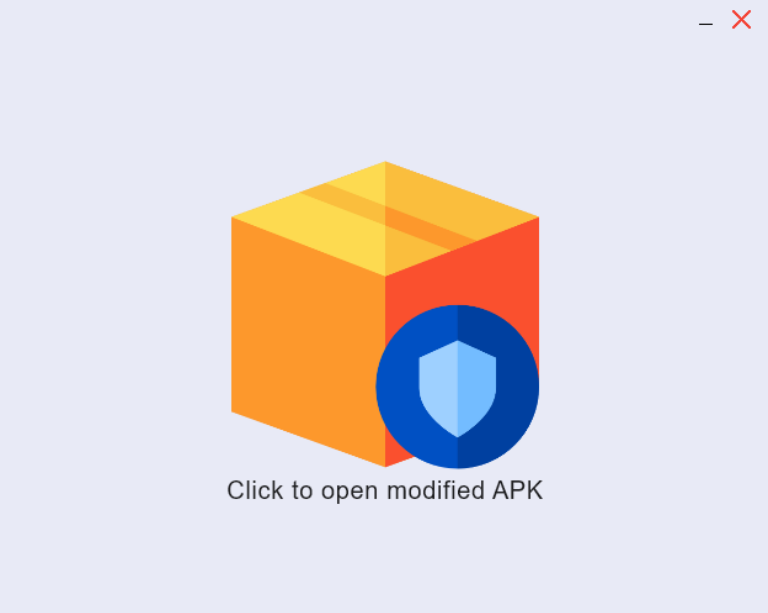

Рисунок 3.15 - Результат успішного пакування застосунку

Якщо в процесі пакування застосунку не виникло помилок, то користувач після натискання на іконку коробки опиниться в директорії з оригінальним APK, яка тепер додатково містить модифікований APK файл.

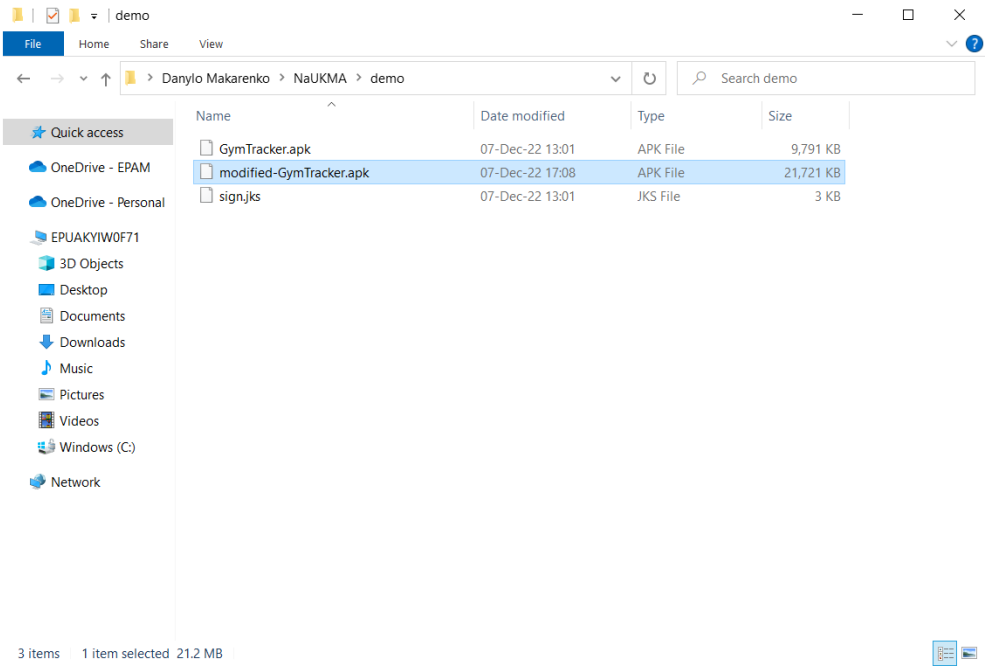

Рисунок 3.16 - Модифікований APK файл

Після завершення процесу пакування зашифрований APK файл готовий до використання. Процес інсталяції на телефон чи емулятор запакованого APK файлу не відрізняється від встановлення звичайного APK файлу. Єдиною особливістю є те, що перший запуск застосунку займає трохи довше часу, адже додатково розшифровуються всі dex файли застосунку.

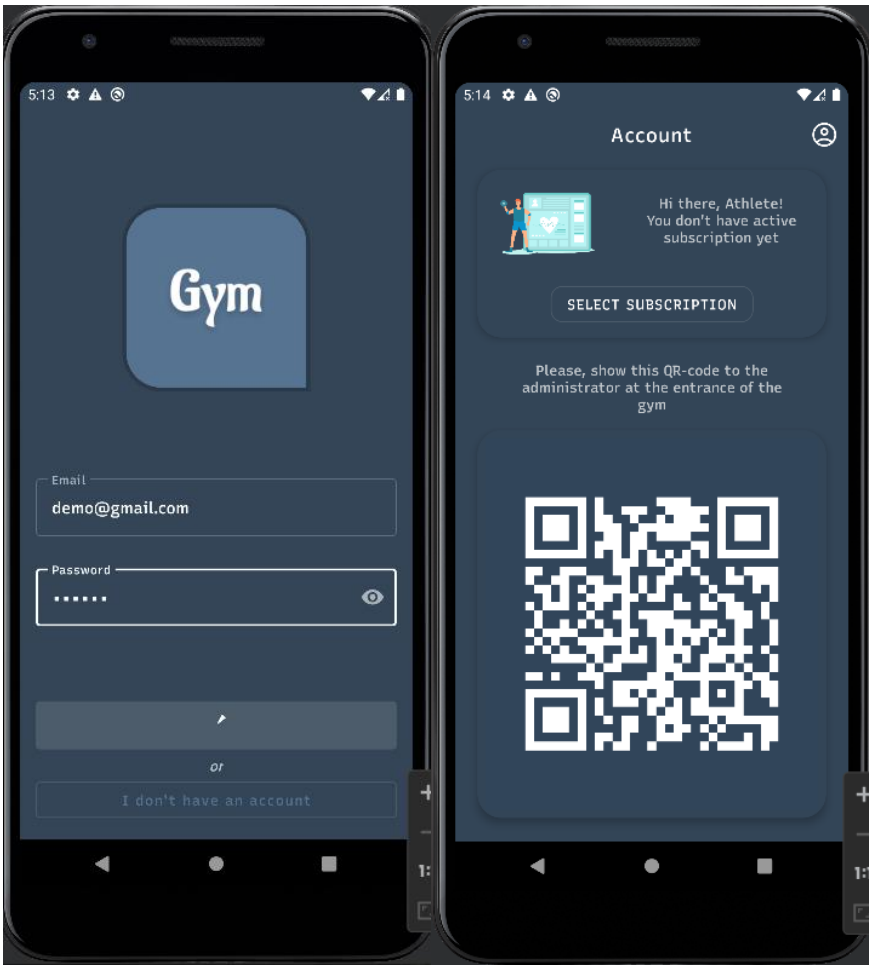

Рисунок 3.17 - Застосунок успішно встановлюється на девайс та працює після пакування.

Особливістю застосування пакувальника є також збільшення розміру APK. Як ми можемо побачити на рисунку 3.18, оригінальний застосунок займає 9.6 МБ. Відповідно classes.dex(3.4 МБ), classes2.dex(2.3 МБ), classes3.dex(2.2 МБ).

Зашифрований APK файл займає значно більше місця – 21.2 МБ (рисунок 3.19). Відповідно error-classes.dex(8.3 МБ), error-classes2.dex(6.1 МБ), error-classes3.dex(5.8 МБ). Сама ж логіка дешифрування classes.dex займає всього 3.8 КБ.

На кінцевому пристрої користувача оригінальний додаток займає 34.08 МБ, а зашифрований – 76.42 МБ (рисунок 3.20).

Таким чином, загальний розмір APK збільшився у 2.2 рази, classes.dex на у 2.44 рази classes2.dex у 2.65 рази, classes3.dex у 2.64 рази. На телефоні інстальований зашифрований застосунок займає у 2.12 разів більше за оригінальний застосунок.

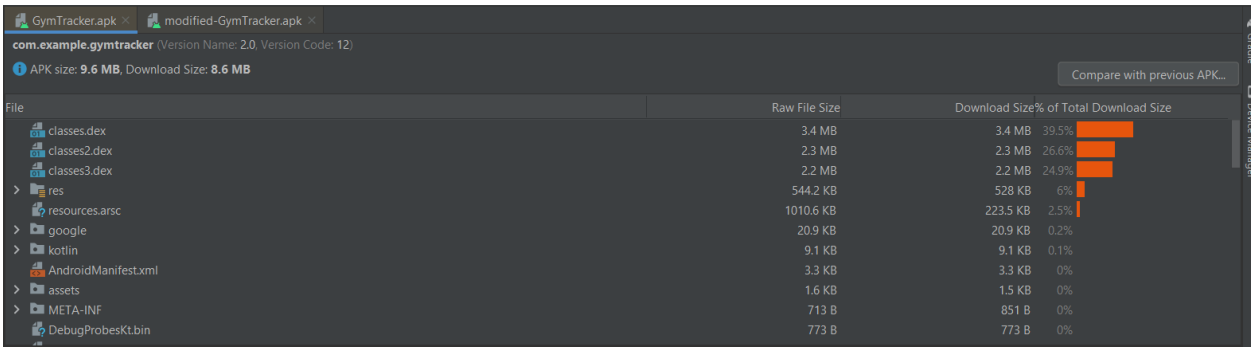

Рисунок 3.18 – Аналіз розміру оригінального APK

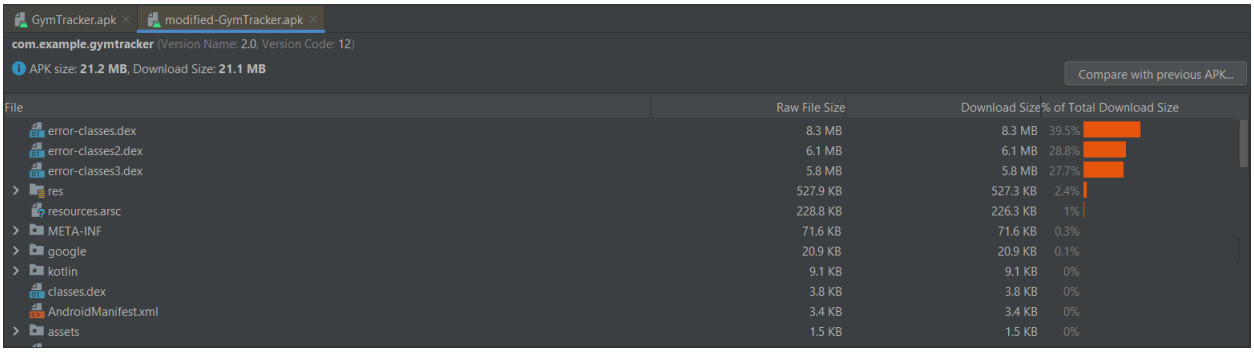

Рисунок 3.19 – Аналіз розміру зашифрованого APK

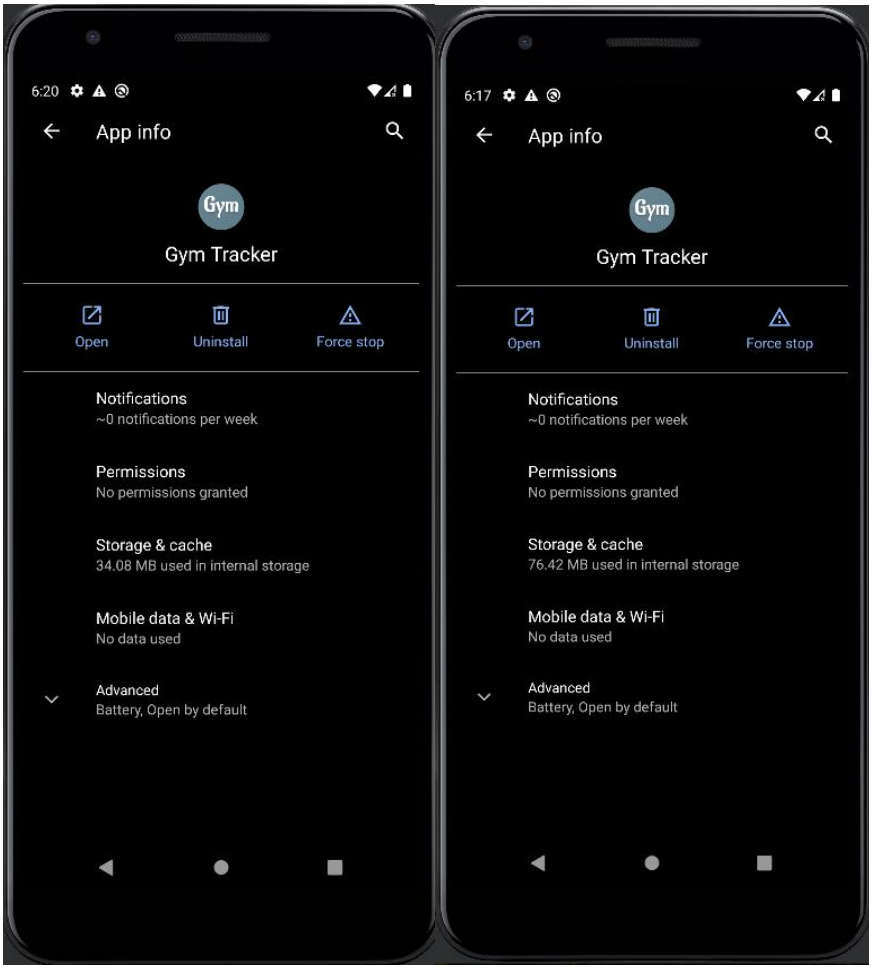

Рисунок 3.20 - Порівняння розмірів оригінального та зашифрованого APK на пристрої користувача

За допомогою програми JADX [42] або подібних легко аналізувати початковий код додатку (рисунок 3.21). Після використання застосунку-пакувальника аналіз початкового коду додатку стає неможливим, адже вся логіка зашифрована та JADX видає помилку при спробі проведення статичного реверс-інжинірингу (рисунок 3.22). Звичайно, зловмисник усе ще може частково переглянути логіку дешифрування (рисунок 3.23), але я додатково використав обфускацію коду дешифрування, що значно ускладнить задачу для реверс-інженера.

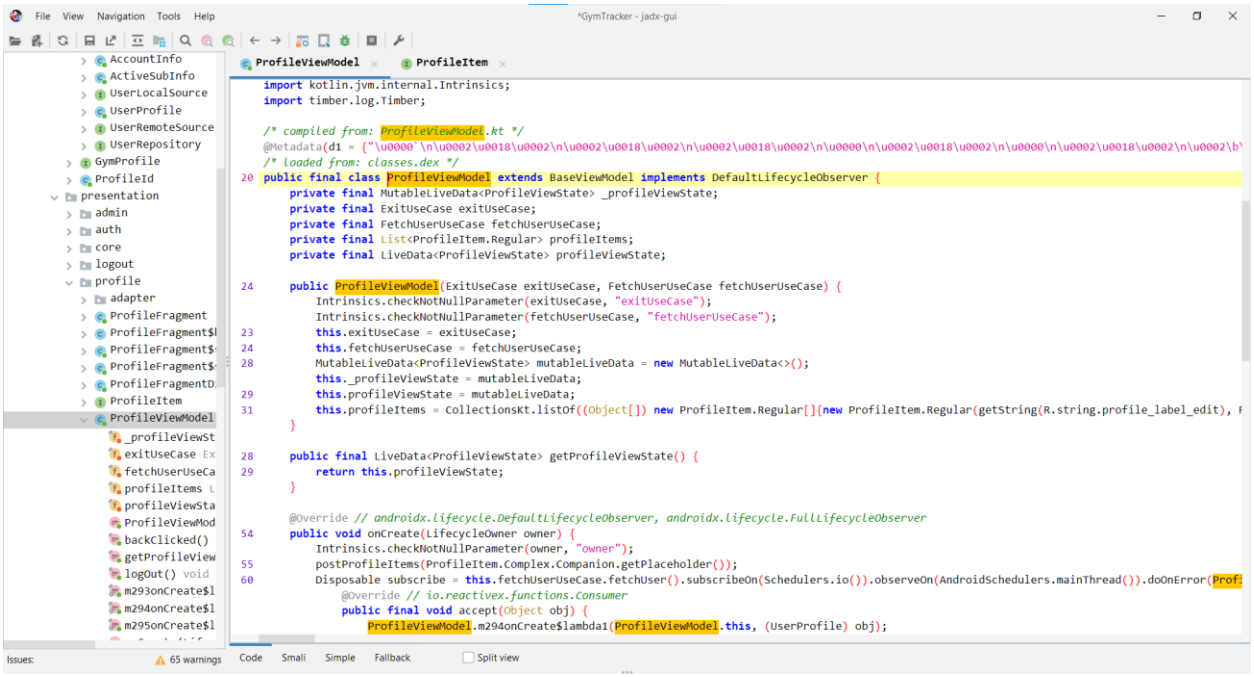

Рисунок 3.21 - Аналіз початкового коду оригінального застосунку

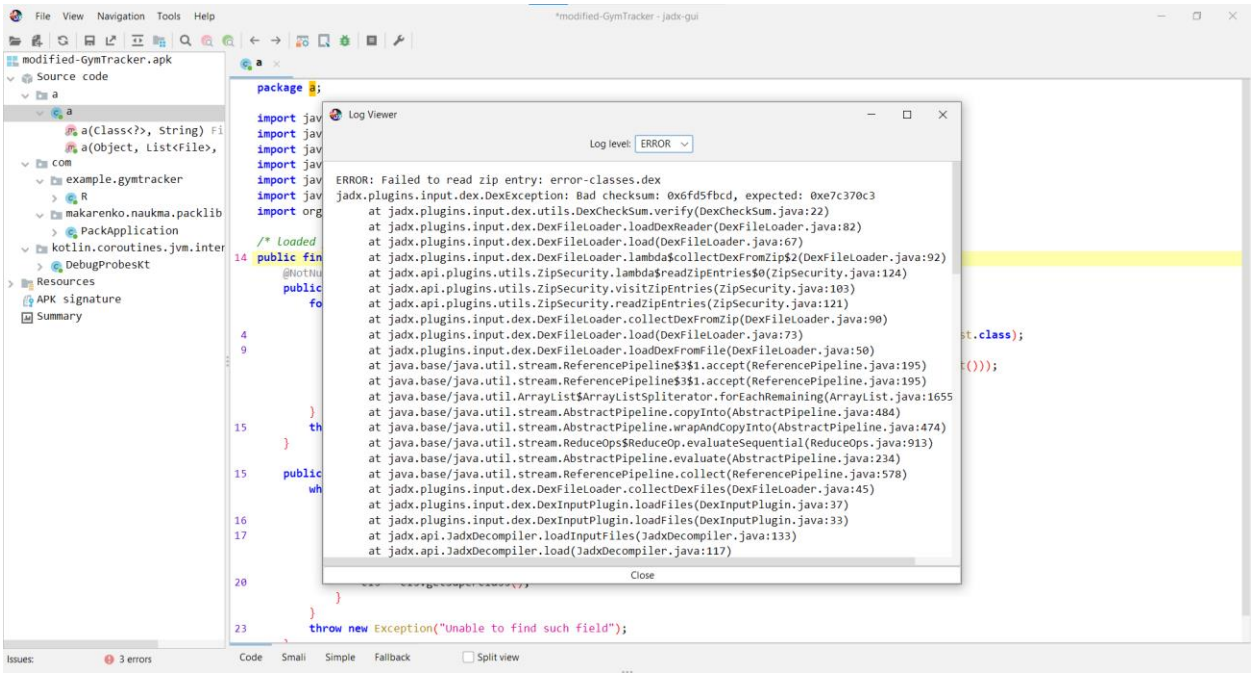

Рисунок 3.22 - Повідомлення про помилку при спробі використання JADX задля аналізу початкового коду зашифрованого застосунку

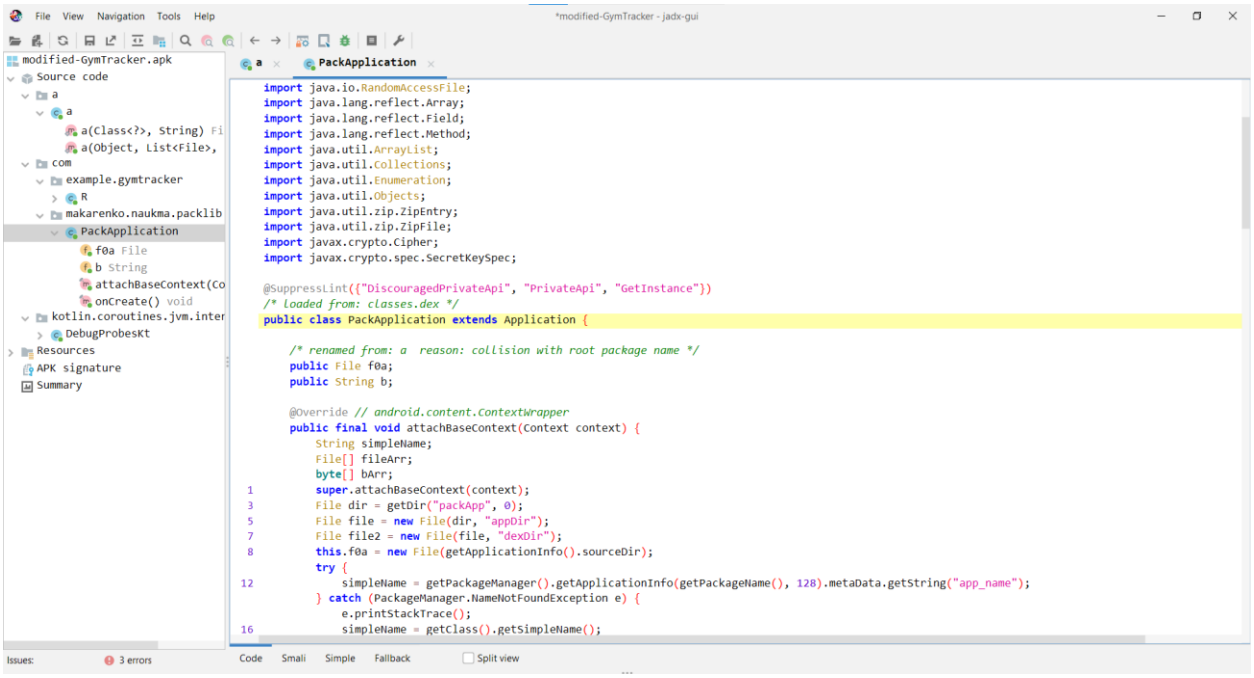

Рисунок 3.23 - Аналіз початкового коду зашифрованого застосунку

3.5 Недоліки використання застосунку

Існує 3 недоліки використання розробленого застосунку-пакувальника :

1. Збільшення розміру APK. Причиною збільшення розміру APK є те, що .dex файли модифікованого застосунку зашифровані, та кінцевий APK містить логіку розшифрування.
2. Збільшення розміру інстальованого додатку. Причини є ідентичними до пункту 1.
3. Перший запуск мобільного додатку займає більше часу. Під час першого запуску відбувається дешифрування, що потребує додаткового часу.

Для демонстрації наведених недоліків я обрав 3 Android застосунки : MyDaily Planner[43], Loop Habit Tracker[44] та GymTracker. Останній є результатом моєї минулорічної курсової роботи. Результати дослідження наведено в таблиці 3.1 :

1. Розмір APK в середньому збільшився у 1.71 разів.
2. Розмір інстальованого додатку на пристрої збільшився в середньому у 2.35 разів.
3. Час першого запуску додатку збільшився у 1.5 рази.

Таблиця 3.1 – Недоліки використання розробленого застосунку

Android додаток	Daily Planner			Habits Tracker			Gym Tracker		
Пристрій	Pixel 3A	Pixel 4	Pixel 5	Pixel 3A	Pixel 4	Pixel 5	Pixel 3A	Pixel 4	Pixel 5
Розмір APK, МБ	10.36			5.23			12.11		
Розмір модифікованого APK, МБ	15.93			8.37			24.13		
Розмір на пристрої, МБ	25.59			13.62			34.08		
Розмір на пристрої з використанням пакувальника, МБ	57.47			34.81			76.42		
Час першого запуску, с	3.88	3.39	3.42	2.17	1.44	1.16	2.87	2.65	1.78
Час першого запуску з використання пакувальника, с	5.42	5.38	5.17	5.42	3.89	2.88	4.15	3.84	3.11

Отже, використання застосунку-пакувальника має 3 недоліки : збільшення розміру APK, збільшення розміру інстальованого додатку та збільшення часу першого запуску. Перелічені недоліки можуть погіршити зручність користування додатком, тому кожен розробник і власник бізнесу мають чітко усвідомлювати необхідність використання застосунку-пакувальника у кожному конкретному випадку.

3.6 Практичні рекомендації

Виявлення відлагоджувача, прав суперкористувача, інструментів для проведення реверс-інжинірингу або факту виконання на емуляторі є

обов’язковими, адже без дотримання базових вимог до стійкості мобільних застосунків від реверс-інжинірингу використання пакувальника не має сенсу.

Наприклад, якщо зловмисник має права суперкористувача або використовує емулятор, то він має доступ до всіх файлів, що зберігаються на пристрої. Так як уся розшифрована логіка застосунку зберігається у внутрішньому сховищі пристрою, доступ до якого за звичайних умов не має ніхто крім самого розробника додатку, реверс-інженер зможе легко переглядати всі dex файли, адже права суперкористувача або емулятор надають таку можливість. Можливим рішення проблеми могло б стати розшифрування логіки застосунку про кожному запуску додатку, проте це може погіршити користувацький досвід, адже дешифрування займає певний час залежно від розміру застосунку.

Водночас, обфускація не є обов’язковою, проте ефективно доповнює роботу пакувальника : навіть якщо зловмиснику вдасться розшифрувати логіку застосунку, то на виході він отримає обфускований код, який надзвичайно складно читати людині.

Таким чином, важливою особливістю використання застосунку-пакувальника є те, що він є доповненням, а не альтернативою до перелічених методів протидії реверс-інжинірингу.

Висновки

Актуальність проблеми стійкості мобільних застосунків на операційній системі Android була чітко доведена на основі декількох досліджень. Існування стандарту захисту мобільних застосунків Mobile Application Security Verification Standard, розробленого компанією OWASP, та існування інструменту Play Integrity для протидії реверс-інжинірингу, розробленого компанією Google, додатково підтверджують важливість проблеми.

Отримані впродовж написання кваліфікаційної роботи знання про інструменти та методи для проведення та захисту від реверс-інжинірингу неодмінно стануть мені в пригоді в майбутньому та дозволять писати безпечніші застосунки.

Мені вдалося реалізувати мету своєї роботи : розробити сучасний застосунок для підвищення захисту додатків на операційній системі Android від реверс-інжинірингу. Застосунок враховує та виправляє всі недоліки, які було знайдено під час ґрунтовного аналізу існуючих рішень. Додатково до розробки застосунку я навів практичні рекомендації щодо його застосування про важливість комплексного підходу у вирішенні проблеми протидії реверс-інжинірингу, без яких використання застосунку майже не мало б сенсу.

Можливими покращеннями мого застосунку є :

1. Використання DexGuard для підвищення захисту логіки дешифрування мобільних додатків.

У своєму застосунку я використав ProGuard для обфускації коду дешифрування, проте, як я зазначив у своїй роботі, існує DexGuard, що є покращеною, проте платною версією ProGuard. Інтеграція DexGuard є

дуже схожою до ProGuard, то ж за необхідності таке покращення можна легко зробити в майбутньому.

2. Розробка веб-версії застосунку.

Веб-версія застосунку могла б додатково підвищити зручність використання для розробників.

3. Підтримка локалізації.

Локалізація наразі не підтримується Compose Desktop фреймворком, проте імплементація даного покращення також не буде складною.

Таким чином, мені вдалося розробити повністю готовий до використання застосунок. За підвищення захисту від реверс-інжинірингу доводиться платити збільшенням розміру APK та розміру інстальованого додатку, а також уповільненням першого запуску, тому кожен розробник має чітко усвідомлювати доцільність використання застосунку.

Додаток А

(довідниковий)

Рисунки для наведених досліджень

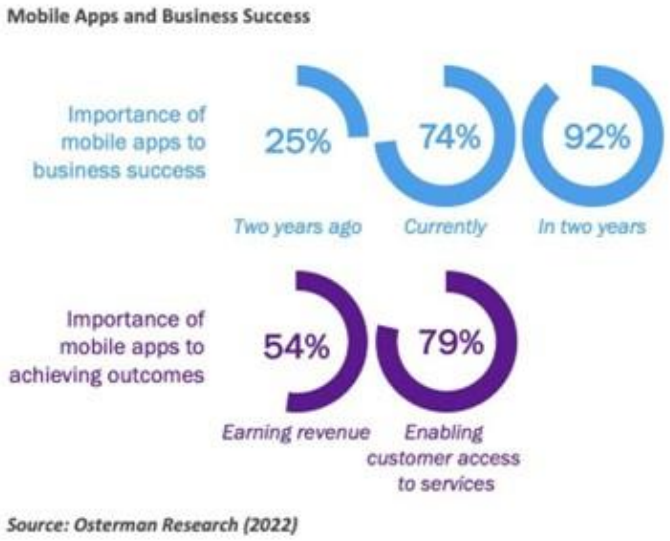

Рисунок Д.1 – Важливість мобільних додатків для бізнесів

Список літератури

1. «State of Mobile App Security» [Електронний ресурс] : https://dokumen.tips/documents/state-of-mobile-app-security-arxan.html?page=1
2. «State of mobile app security 2022» [Електронний ресурс] : https://approov.io/for/state-of-mobile-app-security-2022/
3. «Top 10 Mobile Risks» [Електронний ресурс] : https://owasp.org/www-project-mobile-top-10/
4. «Mobile Operating System Market Share Worldwide» [Електронний ресурс] : https://gs.statcounter.com/os-market-share/mobile/worldwide
5. «Number of available applications in the Google Play Store from December 2009 to September 2022» [Електронний ресурс] : https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/
6. «Dissecting the Bouncer and Malware Detection in Google Play» [Електронний ресурс] : https://www.jetir.org/view?paper=JETIR1808169
7. Google Play Protect [Електронний ресурс] : https://support.google.com/googleplay/answer/2812853
8. Creating a KeyStore in JKS Format [Електронний ресурс] : https://docs.oracle.com/cd/E19509-01/820-3503/ggfen/index.html
9. «Reverse Engineering The Medium App» [Електронний ресурс] : https://medium.com/hackernoon/dont-publish-yet-reverse-engineering-the-medium-app-and-making-all-stories-in-it-free-48c8f2695687
10. «SMART: Static and Dynamic Analysis to Reverse Engineer Android Applications» [Електронний ресурс] : https://repositorio-aberto.up.pt/bitstream/10216/136085/2/494622.pdf
11. «Reverse engineering and modifying an Android game» [Електронний ресурс] : https://medium.com/swlh/reverse-engineering-and-modifying-an-android-game-apk-ctf-c617151b874c
12. OWASP MASVS (Mobile Application Security Verification Standard) [Електронний ресурс] : https://mas.owasp.org/MASVS/

13. «Overview of the Play Integrity API» [Електронний ресурс] : https://developer.android.com/google/play/integrity/overview
14. «Android очима хакера» - Євген Зобнін [Електронний ресурс] — с.105
15. Application клас [Електронний ресурс] : https://developer.android.com/reference/android/app/Application
16. Реалізація застосунку-пакувальника(1) [Електронний ресурс] : https://github.com/huyuxin95/DexShellProject
17. Реалізація застосунку-пакувальника(2) [Електронний ресурс] : https://github.com/oncealong/apk_dex_shell
18. Реалізація застосунку-пакувальника(3) [Електронний ресурс] : https://github.com/dileber/DexShellTools
19. Реалізація застосунку-пакувальника(4) [Електронний ресурс] : https://github.com/Jocerly/DexShellTools
20. Реалізація застосунку-пакувальника(5) [Електронний ресурс] : https://github.com/KouChengjian/DexShellTools
21. Реалізація застосунку-пакувальника(6) [Електронний ресурс] : https://github.com/longtaoge/AndroidShell
22. Реалізація застосунку-пакувальника(7) [Електронний ресурс] : https://github.com/iDeMonnnnnn/Shell
23. Реалізація застосунку-пакувальника(8) [Електронний ресурс] : https://github.com/vbanqi/shellApk
24. Реалізація застосунку-пакувальника(9) [Електронний ресурс] : https://github.com/Herrrb/DexShell
25. Реалізація застосунку-пакувальника(10) [Електронний ресурс] : https://github.com/ray-tianfeng/dex-shell
26. Реалізація застосунку-пакувальника(11) [Електронний ресурс] : https://github.com/Po1lux/DexShell-Java
27. «About the 64K reference limit» [Електронний ресурс] : https://developer.android.com/studio/build/multidex#about
28. Android Asset Packaging Tool [Електронний ресурс] : https://developer.android.com/studio/command-line/aapt2
29. Android Jetpack Navigation Component [Електронний ресурс] : https://developer.android.com/guide/navigation
30. Реалізація застосунку-пакувальника(12) [Електронний ресурс] : https://github.com/openJK-dev/AppJiagu-java
31. Реалізація застосунку-пакувальника(13) [Електронний ресурс] : https://github.com/FlyingYu-Z/ApkEncryptor

32. Compose for Desktop [Електронний ресурс] : https://www.jetbrains.com/lp/compose-desktop/
33. Coroutines [Електронний ресурс] : https://kotlinlang.org/docs/coroutines-overview.html
34. RxJava [Електронний ресурс] : https://reactivex.io/documentation
35. Apksigner [Електронний ресурс] : https://developer.android.com/studio/command-line/apksigner
36. Koin [Електронний ресурс] : https://insert-koin.io/
37. Dagger [Електронний ресурс] : https://dagger.dev/
38. Getting started with MVI Architecture on Android [Електронний ресурс] : https://proandroiddev.com/getting-started-with-mvi-architecture-on-android-b2c280b7023
39. Material Design [Електронний ресурс] : https://m2.material.io/design/introduction
40. Kotlin Extensions [Електронний ресурс] : https://kotlinlang.org/docs/extensions.html
41. d8 [Електронний ресурс] : https://developer.android.com/studio/command-line/d8
42. JADX [Електронний ресурс] : https://github.com/skylot/jadx
43. My Daily Planner [Електронний ресурс] : https://play.google.com/store/apps/details?id=com.time_management_studio.my_daily_planner
44. Loop Habit Tracker [Електронний ресурс] : https://play.google.com/store/apps/details?id=org.isoron.uhabits