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

Кваліфікаційна робота (НаУКМА, ФІ, БП4, ІПЗ 2023)

Виконав: Макаренко Данило Олександрович

Науковий керівник: ст. викл. Борозенний Сергій Олександрович

Що таке реверс-інжиніринг?

Реверс-інжиніринг – це процес зворотної розробки при якому за допомогою різноманітних методів аналізується застосунок із метою виявлення принципів його роботи.

Алгоритм модифікації застосунку:

1. Завантаження
2. Декомпіляція
3. Модифікація .dex файлів
4. Рекомпіляція

Типи реверс-інжинірингу:

1. Статичний
2. Динамічний

Цілі реверс-інжинірингу

1. Пошук вразливостей та/або шкідливих функцій в застосунку
2. Відновлення початкового коду застосунку, якщо останній було втрачено
3. Навчання

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. Отримання інформації про взаємодію з серверною частиною застосунку для подальших атак
5. Безкоштовні версії платних застосунків
6. Крадіжка інтелектуальної власності

1. Зменшення прибутку
2. Репутаційні збитки для компанії
3. Юридичні проблеми

Актуальність

[2022] Опитування трьохсот двох мобільних розробників - Osterman Research&Approv

- 78% опитаних мають сумніви, що їх мобільні застосунки мають достатній рівень захисту
- близько 50% респондентів не впевнені в безпеці застосунків від реверс- інжинірингу

[2016] Дослідження Open Worldwide Application Security Project (OWASP)

- реверс-інжиніринг входить до десяти найбільш популярних вразливостей мобільних застосунків

[2014] «State of Mobile App Security» - Arxan Technologies

- 97% найпопулярніших платних додатків мали безкоштовних клонів

Мета

Розробка десктопного застосунку для підвищення захисту додатків на операційній системі Android від статичного реверс-інжинірингу

Набір інструментів

1. Compose Desktop
2. Kotlin Coroutines
3. Apktool
4. Apksigner
5. JADX
6. Koin
7. Java / Kotlin
8. IntelliJ IDEA та Android Studio
9. Material UI

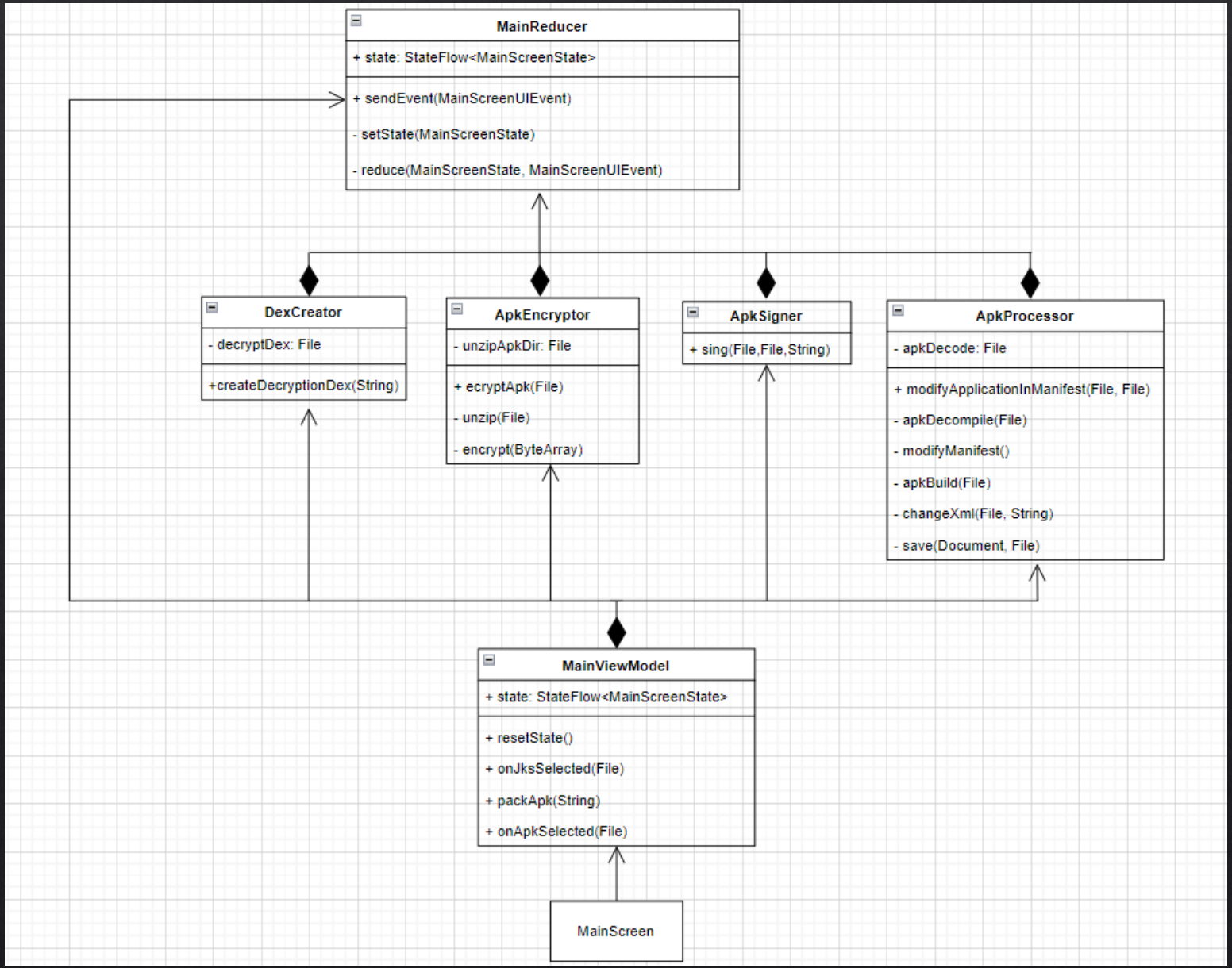
Алгоритм

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 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.

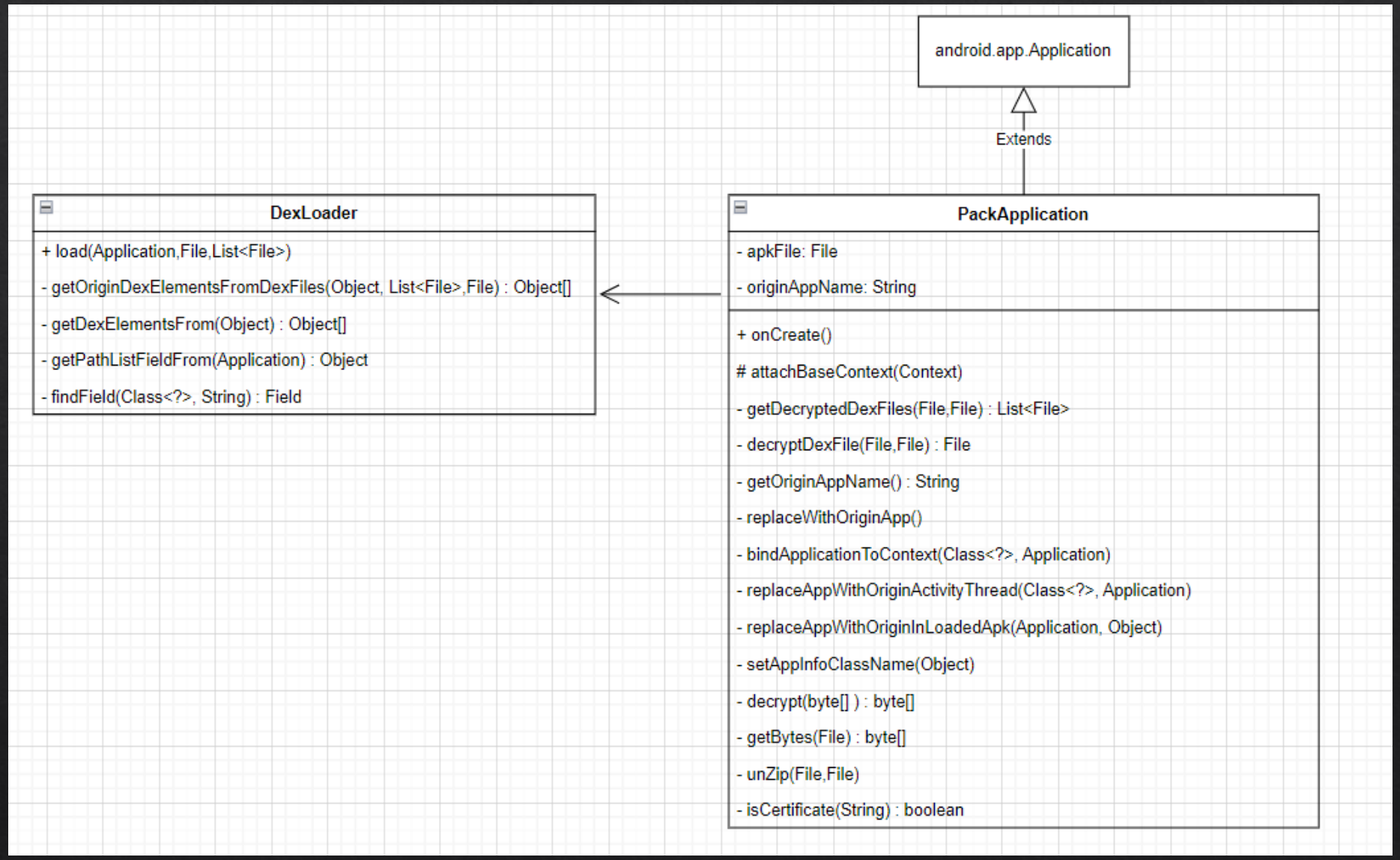
Недоліки існуючих рішень

1. Усі dex файли після шифрування поєднуються в один dex файл, що включає в себе логіку дешифрування.
2. Недостатня автоматизація та зручність використання.
3. Більшість готових рішень не містить документації або просто навіть опису : як цим користуватись?
4. У середньому останні зміни в проєктах проводились 5 років назад
5. Не підтримує роботу з AAPT2 (Android Asset Packaging Tool)
6. Відсутність обфускації логіки дешифрування
7. Відсутність інформації про недоліки та особливості використання

Структура проєкту

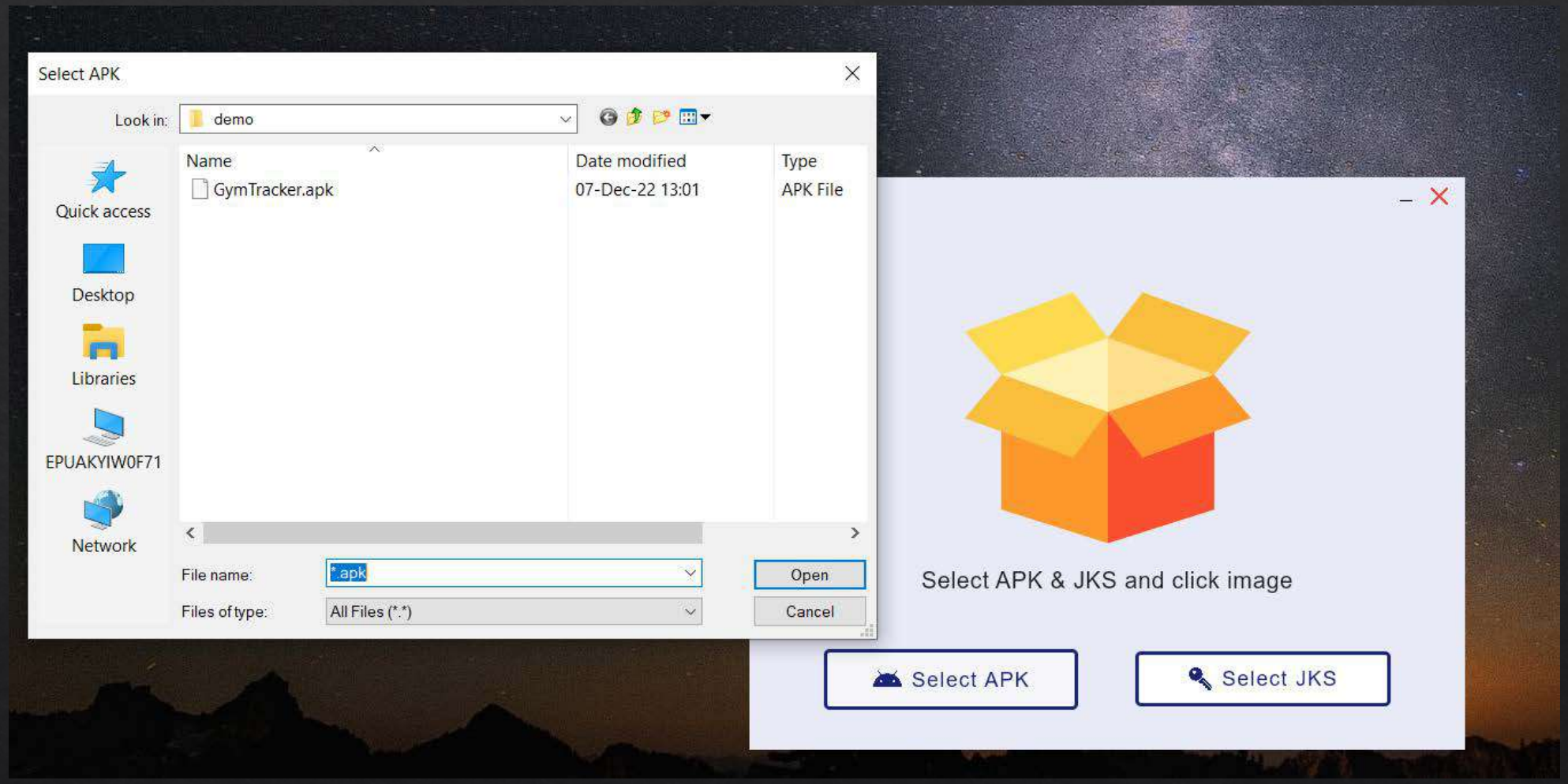


Структура проєкту-дешифрувальника

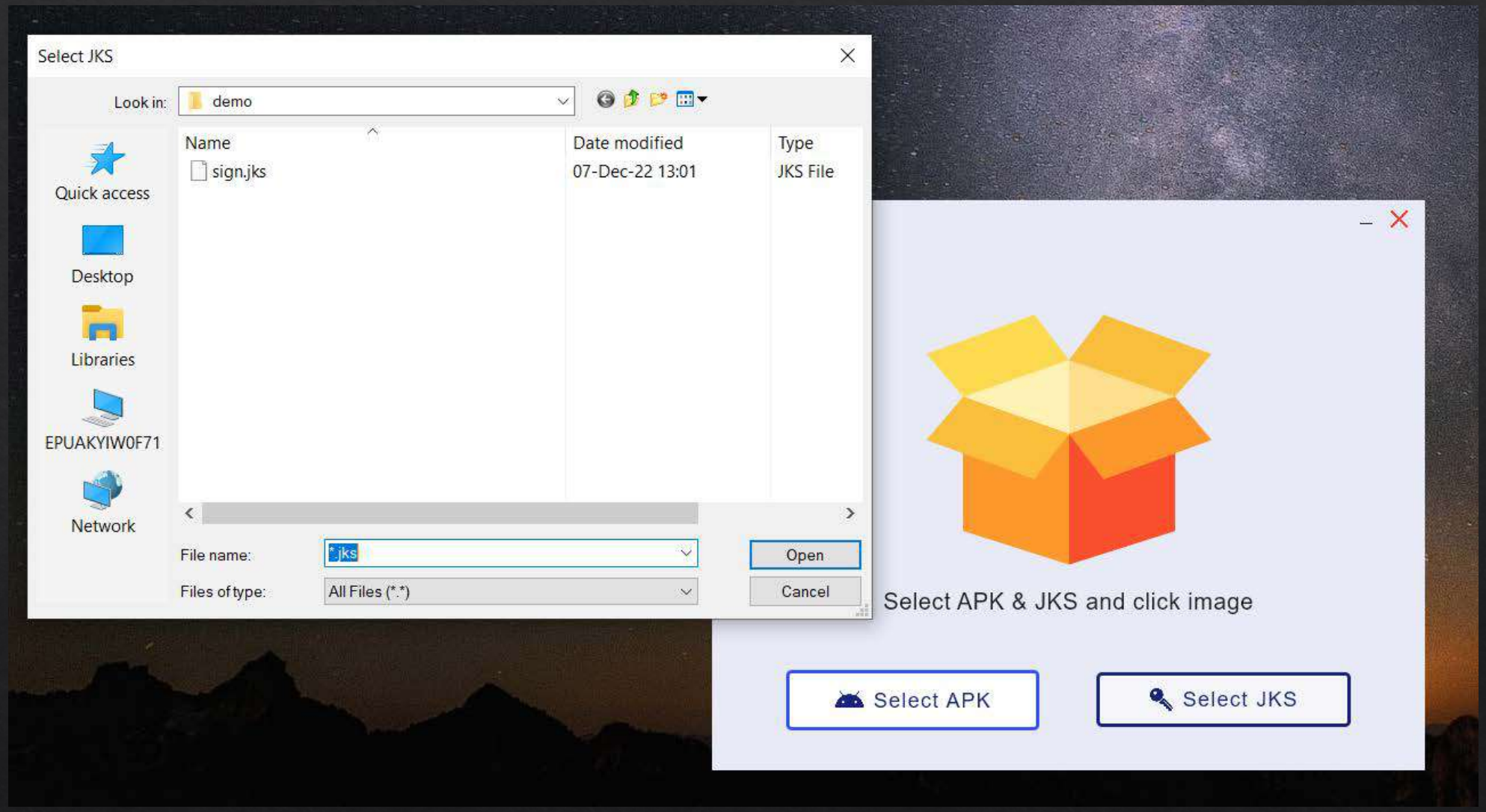


Демонстрація – Головний екран

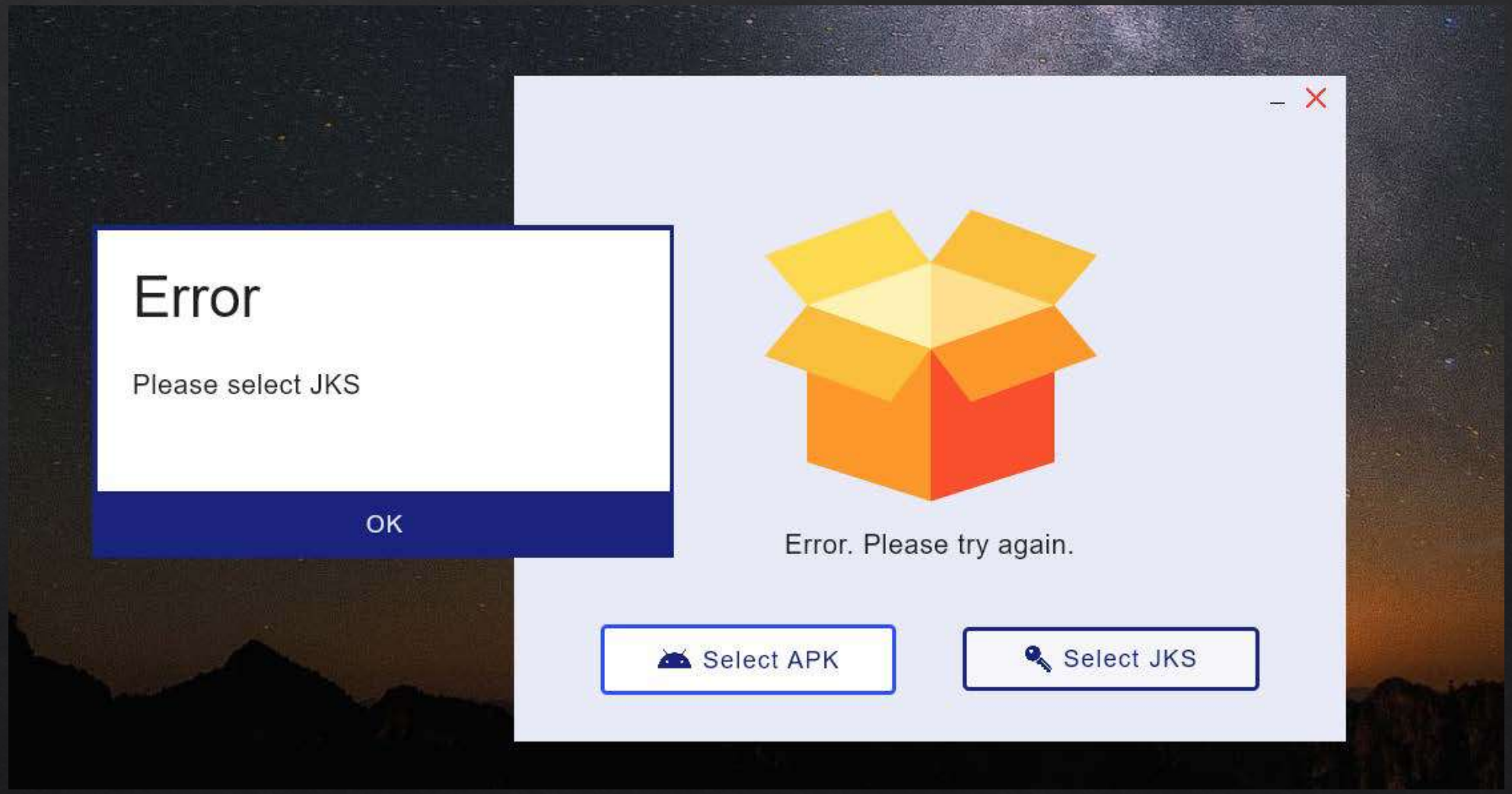
Демонстрація – вибір APK



Демонстрація – вибір JKS

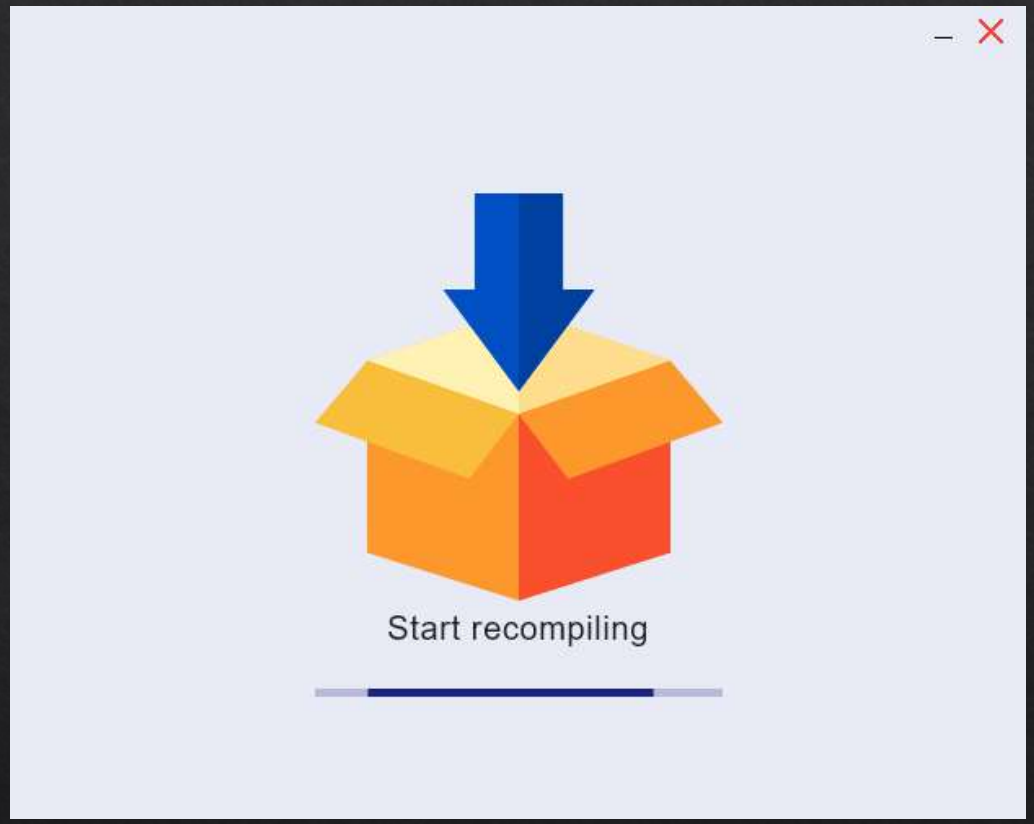


Демонстрація – обробка помилок



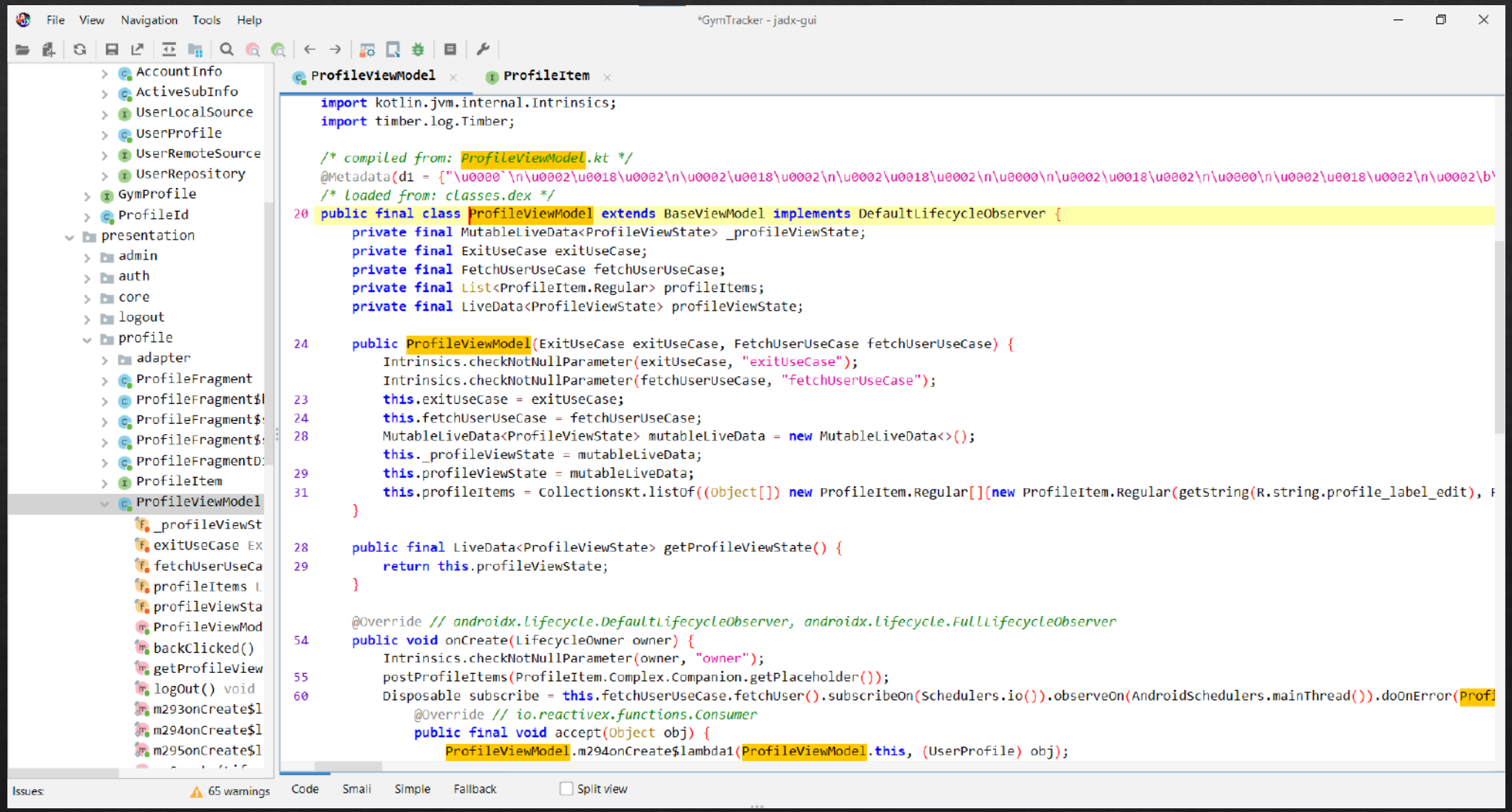
Демонстрація – Надання паролю JKS

Демонстрація – Процес пакування



Демонстрація – Успішне виконання

Аналіз початкового коду застосунку до шифрування



Аналіз початкового коду застосунку після шифрування - Помилка

File View Navigation Tools Help

*modified-GymTracker - jadx-gui

modified-GymTracker.apk
- Source code
  - a
    - a
      - a(Class<?>, String) Fi
      - a(Object, List<File>,
  - com
    - example.gymtracker
      - R
    - makarenko.naukma.packlib
      - PackApplication
  - kotlin.coroutines.jvm.inter
    - DebugProbesKt
- Resources
- APK signature
- Summary

```
package a;

import jav
import jav
import jav
import jav
import jav
import jav
import org

/* Loaded
public fin
    @NotNu
    public
        fo
4
9
        }
15      th
    }
15  public
        wh
16
17
20
        }
    }
23  throw new Exception("Unable to find such field");
```

Log Viewer

Log level: ERROR

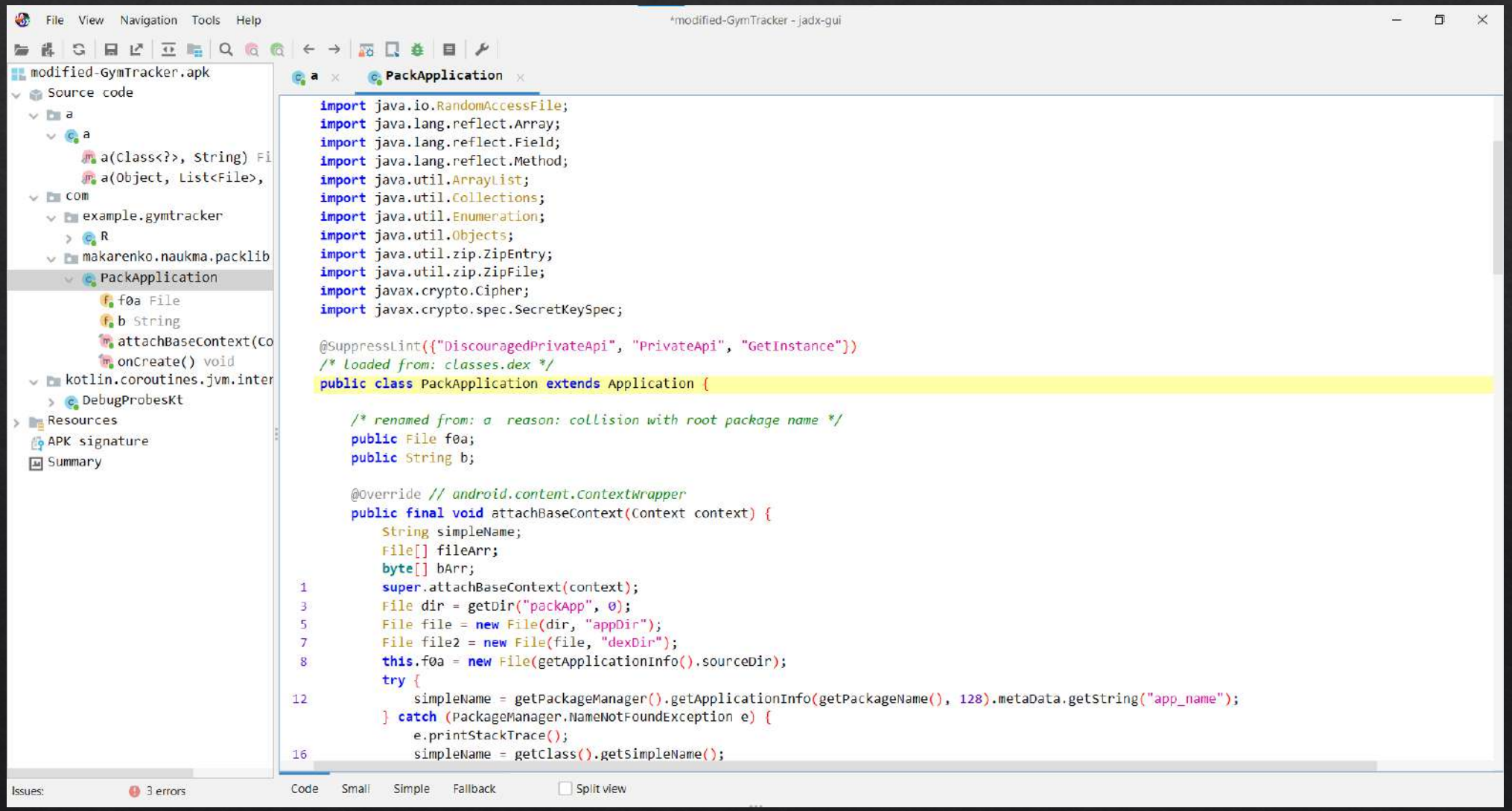
```
ERROR: Failed to read zip entry: error-classes.dex
jadx.plugins.input.dex.DexException: Bad checksum: 0x6fd5fbcd, expected: 0xe7c370c3
    at jadx.plugins.input.dex.utils.DexCheckSum.verify(DexCheckSum.java:22)
    at jadx.plugins.input.dex.DexFileLoader.loadDexReader(DexFileLoader.java:82)
    at jadx.plugins.input.dex.DexFileLoader.load(DexFileLoader.java:67)
    at jadx.plugins.input.dex.DexFileLoader.lambda$collectDexFromZip$2(DexFileLoader.java:92)
    at jadx.api.plugins.utils.ZipSecurity.lambda$readZipEntries$0(ZipSecurity.java:124)
    at jadx.api.plugins.utils.ZipSecurity.visitZipEntries(ZipSecurity.java:103)
    at jadx.api.plugins.utils.ZipSecurity.readZipEntries(ZipSecurity.java:121)
    at jadx.plugins.input.dex.DexFileLoader.collectDexFromZip(DexFileLoader.java:90)
    at jadx.plugins.input.dex.DexFileLoader.load(DexFileLoader.java:73)
    at jadx.plugins.input.dex.DexFileLoader.loadDexFromFile(DexFileLoader.java:50)
    at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:195)
    at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:195)
    at java.base/java.util.ArrayList$ArrayListSpliterator.forEachRemaining(ArrayList.java:1655
    at java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:484)
    at java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:474)
    at java.base/java.util.stream.ReduceOps$ReduceOp.evaluateSequential(ReduceOps.java:913)
    at java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:234)
    at java.base/java.util.stream.ReferencePipeline.collect(ReferencePipeline.java:578)
    at jadx.plugins.input.dex.DexFileLoader.collectDexFiles(DexFileLoader.java:45)
    at jadx.plugins.input.dex.DexInputPlugin.loadFiles(DexInputPlugin.java:37)
    at jadx.plugins.input.dex.DexInputPlugin.loadFiles(DexInputPlugin.java:33)
    at jadx.api.JadxDecompiler.loadInputFiles(JadxDecompiler.java:133)
    at jadx.api.JadxDecompiler.load(JadxDecompiler.java:117)
```

Close

Issues: 3 errors

Code Smali Simple Fallback Split view

Аналіз початкового коду застосунку після шифрування - Дешифрування



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

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.

Висновки

1. Актуальність проблеми стійкості мобільних застосунків на операційній системі Android була чітко доведена на основі декількох досліджень.
2. Було розроблено сучасний десктопний застосунок для підвищення захисту додатків на ОС Android від статичного реверс-інжинірингу
3. Застосунок враховує та виправляє всі недоліки, які було знайдено під час ґрунтовного аналізу існуючих рішень.
4. Додатково до розробки застосунку неведено практичні рекомендації щодо його застосування про важливість комплексного підходу у вирішенні проблеми протидії реверс-інжинірингу, без яких використання застосунку майже не мало б сенсу.
5. Додатково досліджено та висвітлено загальні недоліки застосунків-шифрувальників.
6. Отримані знання про інструменти та методи для проведення та захисту від реверс-інжинірингу дозволять мені писати безпечніші застосунки.