Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідуюча кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«14» червня 2022р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього ступеня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Засоби та механізми захисту мобільних додатків на базі операційної системи Android

Виконавець: студент III (за скороченим терміном навчання) курсу, групи КБ-43мс

_____ **Развій Артем Олександрович**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Іван ПАРХОМЕНКО	

Нормоконтроль	Сергій ДАКОВ	

Київ 2022

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідуюча кафедри кібербезпеки
та захисту інформації
_____Наталія ЛУКОВА-ЧУЙКО
«01» листопада 2021 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студентові **КБ-43мс** (група) **Развію Артему Олександровичу** (прізвище ім'я по-батькові)

Тема дипломної роботи Засоби та механізми захисту мобільних додатків на базі операційної системи Android

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №5 від 29.10.2021 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структури, архітектури, засоби функціонування додатків на базі операційної системи Android

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Технології та модель захисту операційної системи Android, цілі захисту додатків, основні вразливості веб-додатків на базі операційної системи Android, засоби захисту вихідного коду, захист рядків, захист від модифікацій додатку, захист від дослідження логіки роботи додатку

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Поєднання механізмів захисту додатків на базі операційної системи Android.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29.10.2021 року

Завдання видав	(підпис)	Іван ПАРХОМЕНКО (ім’я, прізвище)
Завдання прийняв до виконання	(підпис)	Артем РАЗВІЙ (ім’я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.10.2021 – 28.01.2022	*виконано*
2	Аналіз літератури	29.01.2022 – 11.02.2022	*виконано*
3	Розгляд технологій операційної системи та додатків Android	12.02.2022 – 11.03.2022	*виконано*
4	Дослідження основних вразливостей	12.03.2022 – 07.04.2022	*виконано*
5	Розгляд засобів та механізмів захисту вихідного коду та рядків додатку	08.04.2022 – 20.04.2022	*виконано*
6	Розгляд засобів та механізмів захисту від модифікацій	21.04.2022 – 05.05.2022	*виконано*
7	Розгляд засобів та механізмів захисту від дослідження додатку	06.05.2022 – 20.05.2022	*виконано*
8	Впровадження розглянутих засобів та механізмів	21.05.2022 – 02.06.2022	*виконано*
9	Оформлення пояснювальної записки	03.06.2022 – 06.06.2022	*виконано*
10	Підготовка до захисту	07.06.2022 – 13.06.2022	*виконано*

Завдання видав	(підпис)	Іван ПАРХОМЕНКО (ім’я, прізвище)
Завдання прийняв до виконання	(підпис)	Артем РАЗВІЙ (ім’я, прізвище)

Термін подання дипломної роботи до ЕК 06 червня 2022 року

РЕФЕРАТ

Пояснювальна записка дипломної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел та додатків. Основний текст займає 51 сторінки, включає в себе зміст, вступ, три розділи дипломної роботи, висновки та список джерел. У пояснювальній записці дипломної роботи міститься 4 рисунки.

Метою роботи є реалізація засобів захисту мобільних додатків на базі операційної системи Android.

Для досягнення зазначеної мети поставлено наступні завдання:

– розглянути загальну структуру, модель захисту операційної системи Android та цілі захисту додатків;

– описати найбільш поширені вразливості, характерні для додатків на базі операційної системи Android;

– реалізувати певний набір механізмів та засобів для захисту додатків на базі операційної системи Android.

Об’єктом дослідження є процес захисту мобільних додатків на базі операційної системи Android.

Предметом дослідження є набір механізмів, що реалізують засоби захисту мобільних додатків на базі операційної системи Android.

Практичною цінністю отриманих результатів є поєднання механізмів захисту додатків на базі операційної системи Android.

Ключові слова: мобільний додаток, Android, вразливості, захист персональних даних, повідомлення Intent, Activity, файл-маніфест.

ЗМІСТ

ВСТУП

В наш час мобільні пристрої набули великої популярності та значення серед людей. За допомогою них ми користуємось безліччю різноманітних сервісів. Сервіси можуть бути представлені у вигляді web сайтів, якими можна користуватись за допомогою вбудованого в операційну систему браузеру, або у вигляді додатків, так звані «нативні» додатки, які безпосередньо інсталюються на пристрій користувача. Одна із переваг «нативних» додатків, це те, що вони оптимізовані під конкретні операційні системи, тому вони можуть працювати коректно і швидко, на відміну від web сайтів. Також вони мають доступ до апаратної частини пристроїв, тобто можуть використовувати в своєму функціоналі камеру пристрою, його мікрофон, геолокацію, адресну книгу та інше. «Нативні» додатки інсталюються на мобільні пристрої безпосередньо виробником, також вони можуть бути завантажені зі сторонніх джерел.

На сьогоднішній день Android – одна з найпопулярніших в світі мобільних операційних систем, на якій працюють безліч пристроїв: телефони, планшети, телевізори, розумні годинники, та інше [1]. Швидкий розвиток даної платформи викликав необхідність досліджень у сфері захисту інформації, адже з ростом популярності платформи також зросла кількість потенційних загроз.

Під час розробки додатку на персональний мобільний пристрій слід мати на увазі, що дані, з якими працює цей додаток, можуть представляти певний інтерес для третіх осіб. Ступінь цінності даних дуже різниться, але навість просту приватну інформацію, наприклад, пароль входу в додаток, потрібно захищати. Це особливо важливо через поширеність мобільних додатків у всіх сферах електронних послуг, включаючи фінансові та банківські операції, зберігання і передачу персональних даних, тощо. Додаткам часто довіряють чутливі дані користувача: паролі, токени автентифікації, контакти, комунікаційні записи та інше [2].

РОЗДІЛ 1
ТЕХНОЛОГІЇ ОПЕРАЦІЙНОЇ СИСТЕМИ ТА ДОДАТКІВ ANDROID

1.1 Складові операційної системи Android

Існує щонайменше п'ять технологій, які роблять Android саме тим, чим ми звикли його бачити: віртуальна машина, система багатозадачності, сервіси Google, механізм взаємодії між процесами Binder та ядро Linux. Кожна з цих підсистем наділяє операційну систему Android фірмовими рисами [3].

Вважають, що в основі операційної системи Android лежить мова програмування Java. Java, а також Kotlin – офіційна мова програмування Android. У випадку настільної Java додатки компілюються в проміжний байткод, який потім виконує віртуальна машина. Ранні версії віртуальної машини робили це шляхом інтерпретації: віртуальна машина зчитувала байткод, аналізувала записані в ньому інструкції і виконувала їх. Це повільний метод виконання, адже одній інструкції віртуальної машини могли відповідати десятки або сотні машинних інструкцій. Через це з’явився JIT-компілятор. Це компілятор, за допомогою якого віртуальна машина може під час виконання конвертувати байткод в машинні інструкції, що істотно підвищує швидкість його виконання, але збільшує витрати оперативної пам'яті, оскільки в пам'яті тепер необхідно зберігати не тільки байткод, але і отримані з нього машинні інструкції разом з метаданими, що дозволяють виконувати оптимізації [4].

В перших версіях Android використовувалась регістрова віртуальна машина, яка мала назву Dalvik. Така віртуальна машина була невимогливою до оперативної пам’яті, а також дозволяла виконувати додатки досить швидко навіть в режимі інтерпретації. У Android версії 2.2 було реалізовано JIT-компілятор, а у версії 5.0 було замінено Dalvik компілятор на AOT-компілятор під назвою ART. AOT-компілятор дозволяє позбутися від віртуальної машини як сутності, і переводити додаток в машинні інструкції ще на етапі його установки [4]. Але не весь байткод

можна конвертувати в інструкції процесора і результуючий код в результаті може містити як машинні інструкції, так і байткод віртуальної машини Dalvik. Через те, що AOT-компілятор не має інформації про принцип та особливості роботи додатку, адже необхідно запустити його, то він має недоліки в оптимізації перед JIT-компілятором. Також AOT-компілятор суттєво сповільнює встановлення додатків та перший запуск операційної системи. З Android версії 7.0 було створено гібридний JIT/AOT-компілятор, який уникав цих недоліків. Одразу після встановлення додатку він використовує JIT-компіляцію, але під час процесу зарядки смартфон перетворює додаток в машинні інструкції за допомогою AOT-компілятора, який може враховувати інформацію, накопичену в процесі виконання додатку.

Система Android з самого свого початку була створена з системою багатозадачності, тобто можна запустити безліч різних додатків і всі вони можуть працювати у фоновому режимі. Система сама закриває додатки, коли вільна оперативна пам’ять закінчується або якщо вільної пам’яті не вистачить на відкриття іншого додатку, пріоритет на закриття додатку визначається за частотою його використання. Такий механізм називається lowmemorykiller [5].

Важливий елемент системи багатозадачності – сервіси. Це особливі компоненти додатків, які в ранніх версіях Android могли працювати при абсолютно будь-яких умовах: увімкнений екран або вимкнений, згорнуто додаток або розгорнуто, запущено чи ні батьківський додаток. Сервіс просто робив запит ресурсів процесора і система їх йому надавала. Такий запит до системи називається wakelock. Такий підхід призвів до серйозної проблеми, з'явилася велика кількість додатків, які створювали сервіси, котрі постійно виконували роботу і не давали пристрою перейти в режим очікування. Встановивши на смартфон сотню додатків, користувач отримував кілька десятків сервісів, кожний з яких періодично щось виконував.

В Android версії 6.0 було створено механізм Doze. Він полягає в тому, що якщо пристрій неактивний протягом однієї години, то він переходить в спеціальний енергозберігаючий режим. Одна з особливостей цього режиму — заборона на

wakelock, коли ні додатки, ні сервіси не можуть вивести смартфон з режиму очікування, щоб виконати будь-яку роботу [6].

З версії операційної системи Android 8.0 з’явилась заборона роботи фонових сервісів, але з деякими винятками. У окремих випадках, наприклад, коли додаток знаходиться на екрані, він може запускати сервіси, але система припинить їхній процес після переходу додатку в режим сну. Сервіси, які видимі для користувача, є дозволеними. Вони називаються foreground сервісами, які видно в панелі повідомлень і мають іконку в статусбарі.

З версії Android 9 з’явилось розширення механізму Doze – App Standby Buckets. Його ідея полягає в тому, щоб розділити всі встановлені на пристрої додатки на категорії в залежності від того, як часто вони використовуються. Всього існує п'ять основних категорій [7]:

– Active – додаток використовується в цей момент або використовувався нещодавно;

– Working set – додатки, які часто використовуються;

– Frequent – додатки, які використовуються регулярно, але не обов’язково кожен день;

– Rare – додатки, які використовуються рідко;

– Never – додатки, які встановлені, але жодного разу не запускались.

Система Android групує додатки за категоріями, ґрунтуючись на часі останнього запуску, але виробник пристрою може використовувати інші способи угруповування. Наприклад, з цією метою пристрої Google Pixel застосовують нейронну мережу. Залежно від групи, система застосовує до додатків різні обмеження, включаючи обмеження на запуск фонових завдань, спрацьовування таймерів, доступність мережевих функцій і push-повідомлень. Завдяки такій системі, в тому випадку, якщо вся взаємодія користувача з додатком буде зводитися тільки до прочитання і змахування повідомлень, то через кілька днів додаток перейде в групу Rare і буде серйозно урізаний в можливостях. Додаток не буде урізаний в правах, якщо він доданий в список винятків системи енергозбереження або якщо телефон заряджається.

Система Android з найперших версій використовує “пісочниці” для ізоляції додатків. Кожен додаток запускається від імені окремого користувача Linux і, таким чином, має доступ тільки до свого каталогу всередині /data/data. Один з одним і операційною системою додатки можуть спілкуватися через механізм міжпроцесорної взаємодії – Binder, який вимагає авторизацію на виконання тієї чи іншої дії. В системі Android Binder використовується для всього: запуску додатків, виклику функцій операційної системи, відкриття конкретного екрану в додатку та інше [8].

Для того, щоб використати функції Binder’у потрібно отримати посилання на необхідний об’єкт-сервіс і викликати один з його методів. Фреймворк перетворює цей виклик в повідомлення Binder і відправляє його в ядро через файл-пристрій /dev/binder. Це повідомлення перехоплює Service manager, який знаходить у своєму каталозі потрібний сервіс, перевіряє повноваження додатку на відповідний сервіс і, якщо воно має всі необхідні права, передає повідомлення цьому сервісу. Після отримання та обробки повідомлення сервіс відправляє відповідь, використовуючи той самий Binder. Крім стандартних перевірок повноважень Service manager перевіряє чи має додаток право на створення сервісу. Це здійснюється за допомогою перевірки значення ідентифікатора користувача процесу, який викликається. Якщо ідентифікатор користувача більше 10000 – додаток не має права реєструвати нові сервіси. Усі сторонні додатки в системі Android отримують ідентифікатор користувача більше 10000. Цей механізм застосовується і для інших цілей: з його допомогою система сповіщає додатки про системні події, а саме: вхідний дзвінок, вхідне повідомлення, підключення зарядки і т. д.

Система сповіщень базується на повідомленнях Intent – спеціальному механізмі, реалізованому поверх Binder. Intent’и призначені для обміну інформацією між додатками або операційною системою і додатками, а також запуску компонентів додатків. За допомогою Intent’ів можна сповіщати додатки про події, зробити запит до системи для того, щоб відкрити додаток, для обробки певних типів даних або просто запустити компонент того чи іншого додатку. В операційній

системі Android запуск додатків здійснюється не безпосередньо, а за допомогою повідомлень Intent.

Компанія Google надає вихідний код операційної системи Android та вважає це однією з його головних переваг. Саме через це існує велика кількість нестандартних версій Android. Кожен розробник може зібрати власну версію системи, але така система не буде мати двох важливих складових. Перша складова – це драйвери пристроїв, вихідні коди яких є комерційною таємницею. Другою складовою є сервіси Google, які потрібні в першу чергу для отримання доступу до власного аккаунту, магазину додатків Google Play і хмарного резервного копіювання. Вони також відповідають за багато інших речей: підтримка push-повідомлень, сервіс Google Maps, визначення місця розташування по стільникових вишках і Wi-Fi роутерах, механізм Smart Lock і багато іншого [9].

Операційна система Android заснована на ядрі Linux. Ядро управляє ресурсами пристрою, а також доступом до заліза, оперативною і постійною пам'яттю, запуском, зупинкою і перенесенням процесів між ядрами процесора і багатьма іншими завданнями. Ядро – це центральна частина Android, без якої інші компоненти не працювали би один з одним.

Наявність ядра Linux, а також частково сумісної зі стандартом POSIX середовища виконання, робить Android сумісним з додатками для Linux. Наприклад, система аутентифікації WPA supplicant, яка застосовувалась для підключення до Wi-Fi мереж, була така сама, як в будь-якому дистрибутиві Linux. Ранні версії Android використовували стандартний Bluetooth-стек Linux під назвою bluez, але пізніше його замінили на реалізацію від Qualcomm під назвою bluedroid.

1.2 Цілі захисту додатків

Перш ніж розглянути вразливості додатків, необхідно провести аналіз цілей їхнього захисту. Розгляд цих цілей важливий, адже дозволяє зрозуміти правильність принципів при доступі до безпеки додатку. Крім цього, це робить аудит безпеки

додатків більш зручним: перевірка існування цих елементів управління, а потім розробка та впровадження згаданих елементів при їх відсутності.

Перша ціль захисту – це захист даних користувача. Додаткам часто довіряють чутливі дані, пов’язані з користувачем, наприклад: паролі, токени автентифікації, контакти, комунікаційні записи, IP адреси та доменні імена конфіденційних сервісів.

Зберігання даних користувача до кешу може бути передбачено додатком і часто може зберігати дані користувача безпосередньо у базах даних, XML-файлах або в будь-якому іншому форматі дискового сховища. Розробник додатку може використовувати будь-який формат файлу або механізм зберігання, який йому потрібен. Важливо оцінювати безпеку цих сховищ даних з тією ретельністю, яка використовується для оцінки та аудиту хмарних баз даних та механізмів зберігання інформації через те, що інформація, яка зберігається в додатку, може впливати на безпеку веб-сайтів та інших хмарних сервісів. Наприклад, якщо зловмисник отримує облікові дані для аутентифікації у хмарному сервісі з додатку, то він одразу отримає доступ до самого хмарного сервісу. Додатки онлайн-банкінгу зазвичай отримують та зберігають токени двохфакторної автентифікації у папці вхідних повідомлень, що є не досить безпечним.

Наступною ціллю є захист додатків один від одного: ізоляція, поділ привілеїв. Додатки захищені за допомогою “пісочниці”, це означає, що кожному додатку присвоюється ідентифікатор користувача і він має доступ тільки до власних ресурсів [10]. Даний механізм ізоляції додатків в Android запозичений з Linux. Система Android ввела декілька власних механізмів захисту для того, щоб додатки не зловживали компонентами та даними один одного, таким прикладом є система дозволів Android, яка працює на рівні додатків та реалізується проміжним програмним забезпеченням додатків. Це існує для того, щоб перекласти механізм управління доступом Linux на рівень додатків. Це означає, що кожен раз, коли додаток отримує дозвіл, то конкретному ідентифікатору користувача присвоюється відповідний ідентифікатор групи. Наприклад, коли додаток запрошує дозвіл на використання інтернету, то він переміщується в групу inet.

Додатки часто складаються з безлічі екземплярів сервісів, джерел даних, екранів Activity і приймачів. Для того, щоб захистити ці компоненти від шкідливого або будь-якого ненавмисного шкідливого впливу, важливо, щоб розробники додатків повідомляли і знижували ризик, який їхні додатки несуть для користувача щодо сервісів і даних, до яких вони можуть отримати доступ. Розробники додатків також повинні поважати цілісність цих ресурсів. Ці два принципи безпечної розробки можуть бути реалізовані за допомогою системи дозволів, гарантуючи, що вони запитують тільки необхідні дозволи. Ключовим моментом тут є забезпечення того, щоб розробники дотримувалися принципу найменших привілеїв.

Ще однією важливою ціллю захисту додатків є захист передачі конфіденційної інформації. Розробники додатків також повинні звертати увагу на те, як передається інформація, яка зберігається в додатку. Наприклад, якщо додаток надійно зберігає дані користувача, але дозволяє передати їх третім особам це створює вразливість. Через це зв’язок повинен реалізуватись безпечно. Зв’язок може забезпечуватись кількома способами:

– Міжкомпонентний зв’язок. Додаткам потрібно передавати інформацію між своїми компонентами, наприклад, між двома Activity. Враховуючи, що взаємодія може реалізовуватись через повідомлення Intent або Intent filter то існує ймовірність того, що неавторизовані додатки можуть перехопити цю взаємодію різними способами [11];

– Зв’язок між додатками. Обмін даними між додатками повинен забезпечуватись таким чином, щоб запобігти несанкціонованому втручанню, перехопленню або отримання доступу до цих даних неавторизованими додатками;

– Зв’язок з іншими пристроями. Можливо, додатки будуть використовувати засоби зв'язку NFC, Bluetooth, GMS або Wi-Fi для передачі конфіденційних даних. Розробники повинні вживати належних запобіжних заходів для забезпечення конфіденційності та цілісності даних, які передаються таким чином.

Таким чином, під час аудиту додатку на предмет помилок зв’язку важливо шукати засоби контролю, які забезпечують наступне:

– аутентифікацію між стороною, яка приймає та додатком, який ініціює;

– контроль доступу, що запобігає несанкціонованими сторонами або додатками отримати доступ до даних, які передаються або управління потоку даних.

1.3 Модель безпеки операційної системи Android

Як і більшість систем, модель безпеки Android також використовує переваги функцій безпеки, які пропонує ядро Linux. Операційна система Linux використовує модель із багатьма користувачами, і ядро може ізолювати ресурси користувачів один від одного так само, як воно ізолює процеси. У системі Linux один користувач не може отримати доступ до файлів іншого користувача, якщо не надано дозвіл для цього, і кожен процес виконується з ідентифікатором користувача, який його запустив, якщо у відповідному файлі, що виконується, не встановлені set-user-ID або set-group-ID.

Система Android використовує переваги такої ізоляції користувачів, але ставиться до них інакше, ніж система Linux. У системі Linux ідентифікатор користувача присвоюється або фізичному користувачеві, або системному сервісу. На відміну від системи Android, яка спочатку була розроблена для мобільних пристроїв і, оскільки вони є персональними пристроями, не було необхідності реєструвати в системі різних фізичних користувачів. Через це ідентифікатор користувача застосовується для розмежування додатків та є неявним. Саме це являє собою основу «пісочниці» додатків в операційній системі Android.

У «пісочниці» додатків Android автоматично присвоюється унікальний ідентифікатор користувача, який часто називають ідентифікатором додатку, кожному додатку під час встановлення та виконує цей додаток в окремому процесі, який працює під цим ідентифікатором. Крім цього, кожному додатку надається виділений каталог даних, де дозвіл на їхнє читання і запис має тільки він. Таким чином, додатки ізольовані як на рівні процесів, так і на рівні файлів, це створює ізольоване середовище, яке застосовується до всіх додатків.

Оскільки додатки Android ізольовані, вони можуть отримати доступ лише до власних файлів та будь-яких ресурсів цього пристрою. Тому система може надавати

додаткові, більш деталізовані, права доступу до додатків, щоб забезпечити більш повну функціональність. Вони називаються дозволами і можуть керувати доступом до апаратних пристроїв, підключення до Інтернету, даних або сервісів операційної системи. Додатки можуть запитувати дозволи, визначаючи їх у файлі маніфесту. Під час встановлення додатку операційна система Android перевіряє список запитів на дозвіл та вирішує надавати їх чи ні. Після надання дозволу вони не можуть бути відкликані і доступні додатку без будь-якого іншого підтвердження. Крім того, для таких функцій, як доступ до облікового запису, контактів, повідомлень та інше, потрібно явне підтвердження користувача для кожного такого об'єкту [12]. Деякі дозволи можуть бути надані тільки додаткам, які є частиною операційної системи Android або тим додаткам, які підписані тим же ключем, що й операційна система. Також додатки можуть створювати власні дозволи та встановлювати рівні захисту дозволів, обмежуючи таким чином доступ до сервісів та ресурсів додатків, які створені одним тим самим автором.

Всі додатки операційної системи Android повинні бути підписані їхніми розробниками, включаючи системні додатки. Оскільки APK-файли є розширенням формату пакету Java JAR8, то використовується метод підпису коду, який заснований на підписі JAR. Система Android використовує підпис APK файлів, щоб переконатись, що оновлення для додатка надходять від одного і того ж автора, а також для встановлення довірчих відносин між додатками. Такі функції безпеки реалізуються шляхом порівняння сертифіката підпису додатку, встановленого на даний момент, з сертифікатом оновлення або пов’язаного додатку [13].

Системні додатки підписуються кількома ключами платформи. Різні системні компоненти можуть спільно використовувати ресурси і виконуватися всередині одного і того ж процесу, якщо вони підписані одним і тим же ключем платформи. Ключі платформи генеруються і контролюються тим, хто підтримує версію Android, встановлену на конкретному пристрої: виробниками пристроїв, операторами зв'язку або розробниками власних версій Android які були створені власноруч.

Висновки за розділом 1

У першому розділі було розглянуто технології операційної системи Android, проведено аналіз цілей, яких слід дотримуватись при розробці мобільних додатків та розглянуто модель безпеки, яка використовується у операційній системі Android.

Основними технологіями операційної системи Android, які наділяють її фірмовими рисами, є: віртуальна машина Dalvik, система багатозадачності, Google сервіси, механізм взаємодії між процесами Binder та ядро Linux.

При проектуванні та подальшій розробці додатків необхідно провести аналіз їхніх цілей захисту. Було виділено три цілі захисту додатків в операційній системі Android:

– захист даних користувача;
– захист додатків один від одного;
– захист конфіденційних даних при їхній передачі.

Захист даних користувача полягає у оцінці сховищ даних, в якому зберігається конфіденційна інформація користувача та в якому вигляді.

Захист додатків один від одного полягає у використанні правильної системи взаємодії між додатками. Система Android дозволяє використання обміну повідомленнями Intent між різними компонентами додатків, через це неправильна обробка таких повідомлень може призводити до витоку інформації.

Виділяють три види зв’язків, які реалізуються у системі Android:

– зв’язок між компонентами;
– зв’язок між різними додатками;
– зв’язок з іншими пристроями.

Крім захисту даних додаток повинен реалізовувати захист процесу передачі конфіденційних даних користувача.

Операційна система Android побудована на ядрі Linux. Ядро управляє ресурсами пристрою, в тому числі доступом до заліза, оперативною і постійною пам'яттю, запуском, зупинкою і перенесенням процесів між ядрами процесора і багатьма іншими завданнями. Система Android має певні переваги функцій безпеки,

які їй пропонує ядро, але деякі з них використовує інакше. З перших версій операційна система використовувалась для персональних мобільних пристроїв, отже не мала в собі необхідності використання декількох користувачів, через це, ідентифікатор користувача присвоюється до окремих запущених додатків. Кожен додаток запускається в окремому процесі, йому надається його унікальний ідентифікатор користувача та окремий каталог даних на пристрої до якого тільки він має дозвіл на читання та запис. Це створює ізольоване середовище, яке застосовується до всіх додатків. При цьому спілкування між додатками можливе за допомогою механізму міжпроцесорної взаємодії Binder.

При інсталяції додатку на пристрій користувача необхідно щоб додаток мав власний підпис розробника. Інсталяція додатку завжди відбувається за допомогою APK-файлу. Під час інсталяції система перевіряє чи наявний вже у системі інстальований пакет з тим самим ім’ям, перевіряє версію пакету, вона повинна бути більшою за вже інстальовану, та перевіряє підпис поточного встановленого додатку з завантаженим і, якщо він збігається, то система дає дозвіл на оновлення додатку, в інакшому випадку необхідно видалити додаток зі всіма його даними та інсталювати завантажений пакет. Ця процедура необхідна задля безпеки даних користувача, адже завантажений APK-файл зі сторонніх ресурсів мав би доступ до збережених даних користувача у додатку і при цьому міг би відправляти їх третім особам.

РОЗДІЛ 2
ВРАЗЛИВОСТІ ДОДАТКІВ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

Документ OWASP Top Ten, створений Open Web Application Security Project, виділяє наступні 10 найбільш поширених вразливостей для мобільних додатків за 2016 рік, що до сьогодні є актуальними [14]:

1. Обхід архітектурних обмежень. Ця вразливість охоплює зловживання особливостями платформи, обходу обмежень або невикористання систем контролю управління безпеки платформи, прикладом є сховище Keystore;

2. Небезпечне зберігання даних. До цієї вразливості відноситься небезпечне зберігання і ненавмисні витоки даних;

3. Небезпечна передача даних. Недостатнє підтвердження достовірності джерел зв'язку, невірні версії SSL, передача конфіденційних даних у відкритому вигляді і тому подібне;

4. Небезпечна аутентифікація. Ця вразливість відноситься до аутентифікації кінцевого користувача або неправильне управління сеансами. Включає в себе відсутність вимог перевірки ідентифікації користувача та відсутність перевірки контролю сеансу;

5. Недостатня криптостійкість. Застосування криптостійких алгоритмів для передачі інформації. Використання криптоалгоритмів може бути недостатнім в окремих випадках. Ця вразливість описує варіанти неналежного використання криптографічних елементів, слабкої або недостатньої криптостійкості;

6. Небезпечна авторизація. Ця вразливість описує недоліки авторизації, перевірка на стороні клієнта;

7. Контроль вмісту клієнтських додатків. Ця вразливість розглядає контроль за вхідними даними додатку та якого якістю;

8. Модифікація коду. Ця категорія описує зміну виконуваних файлів, локальних ресурсів, перехоплення викликів сторонніх процесів, підміна методів під час виконання і динамічну модифікацію пам'яті. Після інсталяції додатку, його код

залишається на пристрої. Це дозволяє зловмисному додатку змінювати код, вміст пам'яті, змінювати або замінювати системні методи API, змінювати дані та ресурси програми. Це може забезпечити зловмиснику можливість маніпулювання сторонніми додатками для здійснення нелегітимних дій, крадіжки даних або отримання іншої фінансової вигоди;

9. Аналіз вихідного коду. Ця вразливість включає в себе аналіз бінарних файлів для визначення вихідного коду, бібліотек, алгоритмів і тому подібне. Інструменти реверс-інженірингу можуть дати уявлення про внутрішню роботу додатку. Це може бути використано для пошуку вразливостей додатку, вилучення критичної інформації, такої як ключів шифрування або інтелектуальної власності.

10. Прихований функціонал. Часто розробники включають в код додатків приховані функціональні можливості, механізми, функціональність яких призначена для загального використання. Під цю категорію вразливостей підходить відоме визначення security through obscurity. Розробник може випадково залишити пароль як коментар у додатку. Або це може бути відключення двофакторної аутентифікації під час його тестування.

2.1 Вразливість використання незахищеного з’єднання

Велика частина додатків, які розроблені для мобільних операційних систем, побудовані за клієнт-серверною архітектурою. Через це більшість додатків використовує з’єднання із сервером через мережу Інтернет. Додаток може робити запити на будь-які адреси, в цьому немає обмежень. Важливо те, щоб додаток використовував захищені протоколи з’єднання. Адже завантаження даних через незахищені протоколи, наприклад HTTP, дозволяють контролювати мережу третім особам задля заміни, видалення або впровадження шкідливого коду. Приклад використання незахищеного протоколу [15]:

```
val url = URL("http://uk.wikipedia.org/ ")
val urlConnection: URLConnection = url.openConnection()
val inputStream: InputStream = urlConnection.getInputStream()
```

Починаючи з Android версії 9, всі додатки, які використовують незахищені протоколи http, smtp, ftp, не зможуть виконувати запити до мережі, якщо явно не відключити цей механізм захисту. Але все ще існує можливість використання незахищених протоколів у більш ранніх версіях системи Android.

Також використання незахищених протоколів, наприклад HTTP, призводить до вразливості при передачі чутливих даних через мережу, які передаються в параметрах запиту або у його тілі. При використанні аутентифікації у додатку, деякі розробники створюють власні методи аутентифікації, що веде за собою низку вразливостей, які можуть бути використані для перехоплення чутливих даних користувача. Тому аутентифікація у додатку повинна бути реалізована за допомогою загального, перевіреного на безпечність методу, наприклад OAuth [16].

2.2Вразливість зберігання чутливих даних в незахищених сховищах

Додатки можуть зберігати в собі різні типи даних: паролі, комунікаційні записи, токени, текстові данні та інше в різних типах сховищ: бази даних SQL, інтегровані сховища даних, XML-файли або просто у вигляді змінних всередині коду. Використання незахищених локальних сховищ веде за собою велику небезпеку, адже незахищені локальні сховища легко піддаються зламу, та зазвичай не потребує особливих навичок у зловмисника. Аналогічно зберігання конфіденційних даних у коді у вигляді констант створює вразливість, адже додаток легко декомпілюється і при цьому зчитуються дані. Додатки, які зберігають в собі сторонні токени, наприклад ключі доступу API або токени аутентифікації до різних сторонніх сервісів можуть бути вийняті з додатку та використані зловмисником в різних цілях. При захисті токенів стороннього сервісу необхідно відноситись з тією ж ретельністю що й і до захисту персональних даних користувача.

Android-додатки зберігають дані у власному приватному каталозі, який не призначений для спільного доступу [17]. Зазвичай тільки додаток, якому належить цей каталог, має до нього доступ, але при не правильному проектуванні є низка вразливостей, коли є можливість використання вразливостей компонентів додатку

для отримання доступу до даних, які зберігаються в приватному каталозі. Наприклад, наступний компонент Activity має вразливість отримувати шлях до файлу та шлях завантаження файлу ззовні:

```
class FileUploaderActivity: Activity() {
   …
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      val path = intent.extras?.getString("path");
      val url = intent.extras?.getString("url");
      uploadFile(path, url);
   }
   …
}
```

Існує вразливість додатку, коли не перевіряється шлях до каталогу при збережені файлів, що призводить до ураження перезаписування раніше збереженого файлу на файл зловмисника [18]. Необхідно обмежити користувачів у заданні власного шляху для збереження файлів. Приклад коду, який наводить таку вразливість:

```
var outputStream = FileOutputStream(PUBLIC_DIRECTORY + "/Download/" + filename)
```

Крім того, пристрої Android з підвищеними правами користувача, так званими root-правами, мають можливість компрометації даних користувача при втраті пристрою. Це означає, що якщо конфіденційні облікові дані користувача будуть зберігатись на пристрої в не зашифрованому вигляді, то вони легко можуть скомпрометувати обліковий запис користувача сервісу. Через це необхідно конфіденційні дані зберігати в зашифрованому вигляді та використовувати компонент Keystore [19].

2.3Вразливості компонентів додатку

Також при описі вразливостей додатків операційної системи Android необхідно розглянути вразливості компонентів, які використовуються при їхній розробці.

Наприклад, коли створюється екран Activity у додатку, необхідно її оголосити у файлі маніфесту AndroidManifest.xml і, якщо розробник позначить таку Activity як експортовану, то з’явиться можливість запускати її іншим додатками на пристрої, що може порушувати робочий процес користувача, в тому числі, порушуючи впроваджені процеси безпеки додатку. Activity стає експортованою, якщо до неї додати атрибут android:exported [20]. Також, якщо до створеної Activity додати фільтр Intent’ів, то така Activity автоматично вважається експортованою і може запускатись іншими додатками пристрою:

```
<activity android:name="fit.knu.MainActivity ">
...
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
...
</activity>
```

Щоб уникнути проблеми дозволу запуску Activity з інших додатків необхідно явно позначити android:exported="false" у всіх Activity, де це не використовується. Також можна конфігурувати додаток так, що тільки компоненти, які підписані ключем додатку, зможуть відкривати компоненти, які позначені як експортовані. Для цього необхідно у файлі маніфесту вказати атрибут: android:protectionLevel="signature".

Під час розробки додатку часто використовується компонент сервісів для виконання різних цілей у фоновому режимі: процесу завантаження файлів, авторизації користувача, перевірки місцерозташування пристрою та інше. Так само, як і компонент Activity, сервіси необхідно оголошувати та налаштовувати у файлі маніфесту AndroidManifest.xml. І функція з експортуванням компоненту так само працює і з сервісами. Тобто якщо ми позначимо сервіс при його розробці як android:exported="true" або застосуємо до нього <intent-filter>, то такий сервіс можна буде запускати будь-якому додатку на пристрої.

Додатки, які приймають та запускають довільні повідомлення Intent із зовнішніх джерел, можуть дозволити шкідливому програмному забезпеченню непрямим способом запускати внутрішні компоненти додатку або отримувати доступ до захищених content:// посилань, які використовуються для безпечного обміну файлами, пов'язаних з додатком. Приклад вразливого коду, який дозволяє використати наведену вразливість [21]:

```
fun onHandleIntent(intent: Intent?) {
   if (intent != null) {
      val privateIntent = Intent(this, parseIntentClass(intent));
      startActivity(privateIntent);
   }
}
```

Для того, щоб не допустити даної вразливості, необхідно перевіряти всі Intent повідомлення, які використовуються у функції startActivity(), повідомлення Intent повинні створюватись з даних, які є захищеними або які є не конфіденційними. Якщо необхідно використовувати дані, які є незахищеними або якщо вони мають певний рівень конфіденційності, необхідно перевіряти ім'я та пакет кінцевого компоненту або цифровий підпис додатку, котрий володіє кінцевим компонентом, якому надсилається дане повідомлення Intent.

Висновки за розділом 2

У другому розділі в загальному було розглянуто види вразливостей, які присутні у додатках, які розробляються на базі операційної системи Android.

Виділяють 10 найбільш поширених категорій вразливостей: обхід архітектурних обмежень, небезпечне зберігання даних, небезпечна передача даних, небезпечна аутентифікація, недостатня криптостійкість, небезпечна авторизація, контроль вмісту клієнтських додатків, модифікація коду, аналіз вихідного коду, прихований функціонал. Усі наведені категорії несуть в собі певні вразливості, які можуть бути використані зловмисниками для досягнення різноманітних цілей, тому

при розробці мобільних додатків необхідно мати на увазі, що існують певні вразливості.

Більшість мобільних додатків побудовані на клієнт-серверній архітектурі, таким чином вони виконують певні запити до мережі Інтернет для відображення певних даних користувачу. Якщо розробник додатку буде виконувати запити, використовуючи незахищені протоколи з'єднання, то такі запити можуть легко бути перехоплені зловмисниками.

Більшість користувачів зберігають на своєму мобільному пристрої чутливі дані: текстові дані, паролі, токени, комунікаційні записи та інше. Розробник не обмежується у використанні сховищ для зберігання цих даних. Через це виникає низка вразливостей, яких можуть припуститися розробники додатків. При неправильній побудові додатку він може надавати доступ до конфіденційної інформації на запит від іншого додатку або може робити певні модифікації файлів, які вже присутні на пристрої.

Система Android з кожною новою версією покращує свої механізми безпеки та вводить низку обмежень на використання певних функцій пристрою, збільшуючи при цьому безпеку даних користувача. Але в цей час відбувається підтримка більш старих версій систем Android, також не всі пристрої підтримують останні версії системи. Компоненти, на яких будуються додатки, не можуть забезпечити стовідсотковий захист, якщо розробник буде припускатися певних вразливостей при їх використанні.

Компонент Activity, який відображає певний екран, може бути запущений будь-яким іншим додатком на пристрої, якщо його позначити як exported. Це означає, що може бути порушений робочий процес користувача, в тому числі, це може порушувати впроваджені процеси безпеки додатку.

Також часто при розробці додатків використовується компонент сервісів, для якого виконується те саме, що і для компоненту Activity. Оскільки сервіси використовуються для виконання різноманітних дій у фоні, наприклад авторизація користувача, завантаження файлів, перевірки місцерозташування пристрою та інше,

то цей компонент також потребує певних обмежень на запуск з інших додатків, адже сервіс приймає параметри на запуск ззовні.

Необхідно розуміти, що якщо додаток не передбачає взаємодію з іншими додатками інших розробників або одного розробника, то можливе використання атрибуту android:protectionLevel="signature" у файлі-маніфесту, до якого входить перевірка підпису при використанні компонентів іншими додатками.

Повідомлення Intent використовується в розробці для того, щоб компоненти додатків запитували певний функціонал у інших компонентів. Intent дозволяє взаємодіяти з іншими компонентами з тих самих додатків або з компонентами, створеними іншими додатками. Додатки можуть реагувати на такі повідомлення, надаючи певний функціонал або відкриваючи певний екран. Якщо розробник передбачить обробку вхідних повідомлень Intent без перевірки джерела їх надходження, то така реалізація передбачає низку вразливостей, якими може скористатись зловмисник. Зловмисник може вказати необхідний компонент Activity та вказати певні параметри і надіслати його додатку.

РОЗДІЛ 3
ЗАСОБИ ЗАХИСТУ ДОДАТКІВ НА БАЗІ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

3.1 Приховування вихідного коду

Найкращий засіб захисту вихідного коду від дослідження – це його обфускація. Обфускація, або заплутування, коду – приведення початкового коду або виконуваного програмного коду до вигляду, який зберігає його функціональність, але ускладнює аналіз, розуміння алгоритму роботи і модифікації при декомпіляції. Існує декілька способів це зробити. Найбільш ефективний інструмент для обфускації коду, який входить в інтегроване середовище розробки Android Studio – R8 [22]. Для його активації необхідно у файл build.gradle додати наступний код:

```
android {
…
   buildTypes {
      release {
         minifyEnabled true
      }
   }
…
}
```

Після цього Android Studio буде використовувати обфускацію при побудові APK-файлів додатку. Це також призведе до того, що за рахунок видалення коду, який не використовується, додаток стане більш компактним, а також отримає деякий рівень безпеки від його реверсу. Під час обфускації замінюються імена всіх внутрішніх класів, методів та змінних на одно- та двобуквені сполучення, це суттєво затрудняє розуміння декомпільованого коду.

За замовчування R8 перейменовує класи, методи та змінні, використовуючи англійський алфавіт: перший перейменовується в “a”, наступний в “b” і так далі.

Такий підхід має недоліки, адже тому, хто зламує додаток, необхідно буде розібратись в обфускованому коді лише один раз і в наступних версіях додатку йому легко буде знайти необхідну ділянку коду, адже його ім’я, скоріш за все, буде таким самим. Для того, щоб позбавитись цього недоліку, ми маємо можливість використовувати словник функцій, класів та змінних при кожній наступній версії додатку. Це зробить генерацію імен непередбачуваною. До файлу налаштувань R8 proguard-rules.pro необхідно додати параметри та вказати :

```
-obfuscationdictionary dictionary.txt
-classobfuscationdictionary dictionary.txt
-packageobfuscationdictionary dictionary.txt
```

3.2 Приховування рядків

В більшості випадків, дослідження додатку починається з пошуку рядків. У вигляді рядків у додатку можуть бути збережені ключі шифрування, ключі доступу API, рядки інтерфейсу, які можуть полегшити подальше дослідження коду. Саме через це приховування рядків – це один з головних засобів захисту додатку.

Однією з найпростіших реалізацій засобу приховування рядків є механізм збереження рядків у файлі strings.xml. Його суть полягає в тому, щоб замість розміщення рядків усередині коду у вигляді констант, що призведе до їх виявлення після звичайного пошуку по файлах декомпільованого додатку, розмістити їх у файлі ресурсів: res/values/strings.xml:

```
<resources>
   <string name="password">412510731</string>
</resources>
```

Після цього ми можемо отримати наступний рядок з коду:

```
val password = resources.getString(R.string.password)
```

Більшість інструментів для дослідження додатків дозволяють переглядати вміст файлу strings.xml, через це необхідно використовувати назву рядку, яка не пов’язана з його змістом. Також ми можемо додати до рядку символи, розділити

його за допомогою функції split та використати необхідну нам частину рядка. Наприклад, рядок «Event 412510731 started»:

```
val string = resources.getString(R.string.calendar_event)
val password = string.split(" ")[1]
```

Також ми можемо розбити рядок на частини, а потім при використанні зібрати його в один. Приховаємо наш пароль, для цього створимо ще декілька рядків у файлі strings.xml:

```
<string name="calendar_event">Event 412 started</string>
<string name="sport_event">Event 510 started</string>
<string name="fitness_event">Event 731 started</string>
```

Задля ускладнення необхідно склеювати рядки в різних місцях програми: методах, класах:

```
val a = resources.getString(R.string.calendar_event).split(" ")[1]
val b = resources.getString(R.string.sport_event).split(" ")[1]
val c = resources.getString(R.string.fitness_event).split(" ")[1]
val password = a + b + c
```

Якщо ми будемо склеювати рядки, які не знаходяться у файлі strings.xml, то ми можемо зіткнутись з проблемою покращення продуктивності компілятора, адже він може зібрати такий рядок воєдино. Через це модифікатори static та final до таких змінних рядків краще не застосовувати.

Для ще більшого заплутування ми можемо робити XOR над рядками. Механізм полягає в тому, що ми беремо рядок, створюємо ще один рядок-ключ, перетворюємо їх на масив байтів та застосовуємо до них операції XOR. В результаті такої операції ми отримаємо закодований за допомогою XOR рядок. Створимо функції шифрування та дешифрування [23]:

```
private fun encode(input: String, key: String): String {
    return Base64.encodeToString(xor(input.toByteArray(), key.toByteArray()), 0)
}
private fun decode(input: String, key: String): String {
    return String(xor(Base64.decode(input, 0), key.toByteArray()))
}
private fun xor(a: ByteArray, key: ByteArray): ByteArray {
```

```
    val out = ByteArray(a.size)
    for (i in a.indices) {
        out[i] = (a[i] xor key[i % key.size])
    }
    return out
}
```

Необхідно створити рядок з ключем та з його допомогою зашифрувати рядки, які необхідно сховати і вивести їх в консоль:

```
val pass = "8535"
Log.d("TAG", "onCreate: ${encode(a, pass)}")
Log.d("TAG", "onCreate: ${encode(b, pass)}")
Log.d("TAG", "onCreate: ${encode(c, pass)}")
```

Після запуску цього коду в консолі ми побачимо результат шифрування частин рядку: DAQB, DQQD, DwYC. Тепер необхідно замінити оригінальні частини рядка на зашифровані та використовувати функцію розшифрування при доступі до них:

```
private fun isValidPassword(input: String): Boolean {
    val a = resources.getString(R.string.calendar_event).split(" ")[1]
    val b = resources.getString(R.string.sport_event).split(" ")[1]
    val c = resources.getString(R.string.fitness_event).split(" ")[1]
    val password = Utils.decode(a) + Utils.decode(b) + Utils.decode(c)
    return input == password
}
```

При використанні такого механізму рядки не будуть відкрито зберігатись у вихідному коді додатку, але з’являється ключ, який нам також необхідно приховувати.

Найбезпечнішим способом приховування рядків є їхнє розміщення в «нативному» коді додатку, у файлі, який компілюється в інструкції ARM/ARM64. Розібрати такий код важко, адже декомпілятора для нього не існує, дизасембльований код важкий для читання та розуміння і потребує навичок при його дослідженні. В Android «нативний» код пишеться на мовах C/C++ [24]. Для

прикладу я обрав мову C++. Необхідно створити файл app/src/main/jni/pass.cpp, в якому буде знаходитись наша бібліотека з функцією, яка повертає рядок:

```
#include <jni.h>
extern "C" jstring Java_knu_fit_Utils_getPass(JNIEnv *env, jobject) {
   return (*env).NewStringUTF("8535");
}
```

Після цього у функції розшифрування необхідно викликати «нативний» метод, який повертає пароль:

```
private external fun getPass(): String
fun decode(input: String): String {
   return String(xor(Base64.decode(input, 0), getPass().toByteArray()))
}
```

Під час компіляції додатку створюється бінарний файл з файлу pass.cpp. Для більшого захисту ми можемо використовувати механізми захисту рядків на мові C/C++ задля ускладнення їхнього знаходження.

3.3 Захист від модифікацій

Ще одним засобом захисту додатку є захист від його модифікації. Всередині пакету APK є набір метаданих, які зберігають контрольні суми абсолютно всіх файлів пакету, а власне метадані підписані ключем розробника [25]. Якщо змінити додаток і знову його запакувати, то його метадані зміняться, а також доведеться їх підписувати заново. А оскільки ключа розробника, яким підписувався оригінальний пакет APK, у людини, яка модифікує додаток, немає, він використовує випадково згенерований ключ. Через те, що система Android не має бази всіх цифрових підписів усіх можливих Android розробників, такий модифікований додаток може бути запущений [26].

Найпростішим механізмом, що реалізує цей засіб, є перевірка цифрового підпису додатку. Необхідно отримати хеш ключа поточного цифрового підпису пакету та порівняти його з раніше збереженим. Якщо вони співпадають, то можна

вважати, що додаток не було модифіковано [27]. Отримати цифровий підпис у вигляді рядка можна за допомогою наступної функції:

```
fun getSignature(context: Context): String {
    val packageInfo = context.packageManager.getPackageInfo(context.packageName, PackageManager.GET_SIGNATURES)
    var signature = ""
    packageInfo.signatures.forEach {
        val md = MessageDigest.getInstance("SHA")
        md.update(it.toByteArray())
        signature = Base64.encodeToString(md.digest(), Base64.DEFAULT)
        Log.d("TAG", "Signature: $signature")
    }
    return signature
}
```

Після виклику функції в консолі ми отримаємо рядок вигляду zhUALunjnt9CKkCYoTuumH3Y0pI=. Далі необхідно його зберегти та створити функцію, яка порівнює збережений підпис додатку з поточним:

```
fun isValidSignature(context: Context): Boolean {
    return getSignature().trim() == getCurrentSignature(context).trim()
}
```

Функція повертає true, якщо цифровий підпис додатку є оригінальним, та false, якщо додаток було модифіковано. Далі ми можемо аварійно завершити додаток, якщо його цифровий підпис є не оригінальним:

```
if (Utils.isValidSignature(this)) {
    System.exit(0)
}
```

При використані такого способу необхідно сховати функцію перевірки цифрового підпису та сам рядок з оригінальним підписом, адже той, хто буде зламувати додаток, може просто вирізати функцію перевірки.

Ще одним механізмом захисту є перевірка джерела інсталяції додатку. Якщо ми будемо поширювати наш додаток тільки через магазин додатків Google Play, то

ми можемо додати таку перевірку [28]. Магазин додатків Google Play має ім’я пакету com.android.vending. Функція для перевірки виглядає наступним чином:

```
fun isValidInstaller(context: Context): Boolean {
    val installer = context.packageManager.getInstallerPackageName(context.packageName)
    if (BuildConfig.DEBUG) {
        return true
    }
    return installer != null && installer == "com.android.vending"
}
```

Якщо ми запускаємо додаток в режимі дебагу, тобто тестуємо його під час розробки, то функція повертає значення true.

3.4 Захист від відладки

Крім захисту від модифікації додатку, ми можемо додати функції виявлення відладчика. Одним із засобів дослідження додатку, є його запуск в режимі відладки. Можна декомпілювати пакет APK додатку, встановити в ньому прапорець відладки, відкрити вихідні дані у інтегрованому середовищі розробки Android Studio та запустити процес відладки. Після цього додаток легко досліджується, адже в режимі відладки можна побачити всю його логіку роботи. Необхідно, щоб додаток перевіряв чи підключений відладчик та, якщо він підключений, приховував свою логіку роботи або завершував свою роботу [29]. Функція для такої перевірки на мові Kotlin виглядає наступним чином:

```
fun isDebugMode(): Boolean {
    return Debug.isDebuggerConnected()
}
```

При зламі додатку часто використовується емулятор системи Android задля полегшення процесу зламу. Тому в деяких випадках можна зробити перевірку на те, чи виконується додаток у віртуальному середовищі чи ні [30]. Стандартний емулятор інтегрованого середовища розробки Android Studio встановлює наступні змінні зі значеннями:

```
ro.hardware=goldfish
```

```
ro.kernel.qemu=l
ro.product.model=sdk
```

Наступна функція повертає true, якщо додаток запущено у середовищі емулятора та false, якщо на реальному пристрої:

```
fun isEmulatorRun(): Boolean {
    return try {
        when {
            getSystemProperty("ro.hardware").contains("goldfish") ||
                    getSystemProperty("ro.kernel.qemu").isNotEmpty() ||
                    getSystemProperty("ro.product.model").contains("sdk") -> true
            else -> false
        }
    } catch (e: Exception) {
        false
    }
}
private fun getSystemProperty(name: String): String {
    val sysProp = Class.forName("android.os.SystemProperties")
    return sysProp.getMethod("get", String::class.java).invoke(sysProp, name) as String
}
```

3.5 Функціональні можливості додатку

Для створення додатку було обрано безкоштовне інтегроване середовище Android Studio від компанії Google. Для написання основного коду було обрано мову програмування Kotlin, а для створення «нативного» коду було обрано мову C++. Для коректного створення додатку необхідно завантажити Android SDK – набір із засобів розробки, утиліт і документації, який дозволяє створювати прикладні додатки для платформи Android. Для створення та запуску «нативного» коду необхідно завантажити та інсталювати у інтегроване середовище програмування Android NDK – це необхідний набір інструментів для розробки компонентів програмного забезпечення для платформи Android, який базується на C/C++ та інших мовах програмування, він містить у собі лімітований набір загальновживаних

«нативних» бібліотек та API, написаних на C/C++ та інших мовах програмування, документацію і мінімальний набір прикладів для демонстрації базового функціонала. Завантажити всі необхідні інструменти розробки з офіційних джерел.

Для написання інтерфейсу буде використовуватись структура Layout [31] та мови розмітки XML з використання базових елементів без застосування сторонніх бібліотек.

Додаток повинен відображати поле вводу з паролем та кнопкою підтвердження, яка перевіряє введений пароль з збереженим, інформацію про правильність введення паролю та інформацію про додаток, для зручності перевірки роботи механізмів захисту.

Необхідно створити макет з елементами, які відображають інформацію про: правильність введення паролю, поточний цифровий підпис додатку, збережений цифровий підпис додатку, те, чи ввімкнений режим відладки, те, чи було модифіковано додаток, те, чи інстальовано додаток з магазину додатків Google Play, те, чи запущений додаток на фізичному девайсі чи ні.

Наступний XML код відображає поле вводу паролю, інформацію про правильність вводу паролю та кнопку перевірки введення:

```
<EditText
   android:id="@+id/etInputPass"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_marginStart="16dp"
   android:layout_marginEnd="16dp"
   android:inputType="text"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintTop_toTopOf="parent" />
<TextView
   android:id="@+id/tvResult"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textSize="24sp"
   app:layout_constraintBottom_toTopOf="@+id/etInputPass"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
<androidx.appcompat.widget.AppCompatButton
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="Check password"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/etInputPass" />
```

Для прикладу було обрано пароль 412510731. З другого підпункту третього розділу ми вже дізнались, що операція XOR над трьома частинами цього паролю з ключем 8535 є рядки DAQB, DQQD, DwYC. Їх я додав у ресурси додатку, зкомпонувавши з іншим текстом для заплутування:

```
<resources>
    <string name="app_name">Bachelor</string>
    <string name="calendar_event">Event DAQB started</string>
    <string name="sport_event">Event DQQD started</string>
    <string name="fitness_event">Event DwYC started</string>
</resources>
```

Для виконання розшифрування рядків було створено клас Encoder, в якому є публічна функція decode та приватні функції xor та getPass, які були розглянути у другому підрозділі третього розділу. Функція getPass повертає рядок з «нативної» бібліотеки pass, яку необхідно завантажити при створенні класу:

```
init {
    System.loadLibrary("pass")
}
private external fun getPass(): String
```

Для перевірки правильності введеного паролю користувачем необхідно створити функцію перевірки на головному екрані, яка буде брати зашифровані

рядки з ресурсів, розшифровувати їх, порівнювати з введеним паролем та повертати значення true або false відповідно до результату порівняння:

```
private fun isValidPassword(input: String): Boolean {
    val a = resources.getString(R.string.calendar_event).split(" ")[1]
    val b = resources.getString(R.string.sport_event).split(" ")[1]
    val c = resources.getString(R.string.fitness_event).split(" ")[1]
    val encoder = Encoder()
    val password = encoder.decode(a) + encoder.decode(b) + encoder.decode(c)
    return input == password
}
```

Далі необхідно на відповідну кнопку перевіряти введений пароль, який ввів користувач:

```
findViewById<Button>(R.id.button).setOnClickListener {
    val inputUserPassword = findViewById<EditText>(R.id.etInputPass).text.toString()
    val isValidInputUserPassword = isValidPassword(inputUserPassword)
    val tvResult = findViewById<TextView>(R.id.tvResult)
    if (isValidInputUserPassword) {
        tvResult.text = "PASS"
        tvResult.setTextColor(ContextCompat.getColor(this, R.color.green))
    } else {
        tvResult.text = "ERROR"
        tvResult.setTextColor(ContextCompat.getColor(this, R.color.red))
    }
}
```

Для виведення на екран поточної версії підпису необхідно створити клас ModificateHelper, в якому буде функція getCurrentSignature, яка повертає цій підпис:

```
fun getCurrentSignature(context: Context): String {
    val packageInfo = context.packageManager.getPackageInfo(context.packageName,
PackageManager.GET_SIGNATURES)
    var signature = ""
    packageInfo.signatures.forEach {
        val md = MessageDigest.getInstance("SHA")
        md.update(it.toByteArray())
        signature = Base64.encodeToString(md.digest(), Base64.DEFAULT)
```

```
    }
    return signature
}
```

Необхідно розглянути функції, які повертають рядок з «нативного» коду, написаного на мові C++. Це функції, які повинні за назвою співпадати з пакетом, назвою класу та функцією, в якому місці вона буде викликатись. Функції повинні повертати елемент jstring, який сумісний з мовою Java, в тому числі і з мовою Kotlin. Через те, що додаток може запуститись на різній архітектурі, то він при збірці проекту генерує код для декількох платформ: ARM64, x86, x86_64 [32]. На відміну від мови Java та Kotlin, мові C/C++ необхідно знати, в якому середовищі вона працює і на основі цього створювати рядок, який ми зазначимо, адже такий код може бути запущений на різній архітектурі:

```
extern "C" jstring Java_knu_fit_Encoder_getPass(JNIEnv *env, jobject) {
    return (*env).NewStringUTF("8535");
}
extern "C" jstring Java_knu_fit_ModificateHelper_getSignature(JNIEnv *env, jobject) {
    return (*env).NewStringUTF("zhUALunjnt9CKkCYoTuumH3Y0pI=");
}
```

Далі розглянуті функції в попередніх підрозділах необхідно додати у класи та викликати при створенні екрану і вивести необхідну інформацію. Наведений код викликає функцію порівняння підписів та виводить інформацію на екран:

```
findViewById<TextView>(R.id.tvModificate).apply {
    if (ModificateHelper().isValidSignature(this@MainActivity)) {
        text = "Application not modificated"
        setTextColor(ContextCompat.getColor(this@MainActivity, R.color.green))
    } else {
        text = "Application modificated!"
        setTextColor(ContextCompat.getColor(this@MainActivity, R.color.red))
    }
}
```

Функції для перевірки чи запущено додаток на емуляторі, режиму відладчика та джерела інсталяції було додано в клас Utils. Приклад виклику функції перевірки середовища запуску та виведення інформації на екран:

```
findViewById<TextView>(R.id.tvIsEmulator).apply {
    if (Utils().isEmulatorRun()) {
        text = "Application run on emulator!"
        setTextColor(ContextCompat.getColor(this@MainActivity, R.color.red))
    } else {
        text = "Application run on device"
        setTextColor(ContextCompat.getColor(this@MainActivity, R.color.green))
    }
}
```

Також необхідно налаштувати файл створення додатку build.gradle, в якому треба ввімкнути обфускацію коду, зазначити версію та вказати ім’я Android NDK [32]:

```
android {
…
        ndk {
            moduleName = "error"
        }
    }
    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    externalNativeBuild {
        ndkBuild {
            path 'src/main/jni/Android.mk'
        }
    }
    ndkVersion '25.0.8528842 rc4'
}
```

Після зазначення всіх конфігурацій необхідно зібрати APK-пакет через “Build APK(s)”.

Далі, після реалізації всіх зазначених механізмів, необхідно спробувати декомпілювати створений додаток. Для цього використовується хмарний безкоштовний сервіс Javadecompilers. Потрібно лише завантажити APK-пакет на сервіс та ,після опрацювання, завантажити декомпільований пакет у вигляді архіву [34].

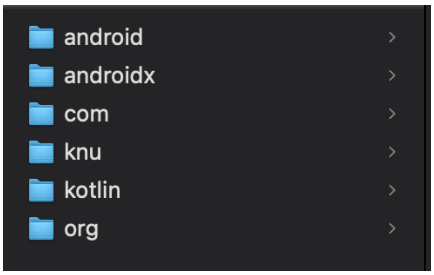

Рисунок 3.2 – Структура проекту додатку без обфускації після його декомпіляції

Для прикладу, на рисунку 3.2 зображено структуру проекту, який не було обфусковано, після його декомпіляції. Він додатково включає в себе код інтегрованих в додаток бібліотек, які необхідні для коректної роботи. Ми можемо побачити, що такий проект легко піддається дослідженню, адже має справжні назви папок.

Також, після створення APK-пакетів всі ресурси рядків додатку будуть зберігатись в одному файлі з рядками бібліотек. Навіть у додатку невеликого розміру буде зберігатись більше ста рядків, які будуть відсортовані за алфавітом. Це означає, що якщо ми надамо рядкам, в яких зберігаються зашифровані дані, імена, які не пов’язані з їхньою суттю, то віднайти логіку їхньої роботи буде набагато важче.

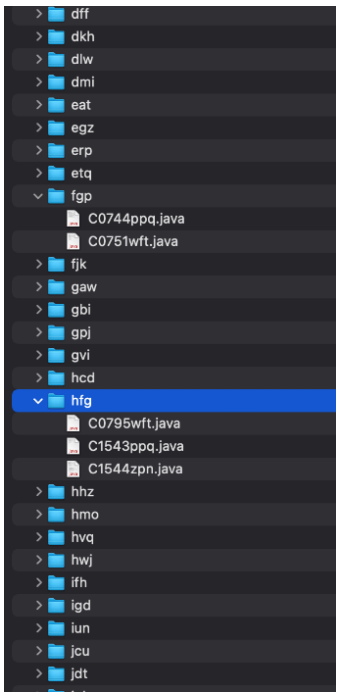

Рисунок 3.2 – Структура обфускованого проекту додатку після його декомпіляції

На рисунку 3.2 показано структурhy проекту додатку після його декомпіляції за допомогою сервісу. Наприклад, вихідний код класу Encoder став виглядати наступним чином:

```
public final class Encoder {
    public Encoder() {
        System.loadLibrary("pass");
    }
    private final native String getPass();
    public final String wft(String str) {
        C2026tbh.anq(str, "input");
        byte[] decode = Base64.decode(str, 0);
        C2026tbh.wtv(decode, "decode(input, 0)");
        byte[] bytes = getPass().getBytes(C1844wft.f6381wft);
        C2026tbh.wtv(bytes, "this as java.lang.String).getBytes(charset)");
        byte[] bArr = new byte[decode.length];
        int length = decode.length;
        for (int i = 0; i < length; i++) {
            bArr[i] = (byte) (decode[i] ^ bytes[i % bytes.length]);
        }
```

```
        return new String(bArr, C1844wft.f6381wft);
    }
}
```

Назва файлу залишилась тією самою, адже в цій функції робиться виклик «нативного» методу, це необхідно для коректної роботи додатку [35]. Також залишилась назва головного екрану після обфускації, це теж необхідно для того, щоб система Android змогла знайти та запустити наш додаток. Всі інші функції були перенесені в інші папки задля ускладнення розуміння коду. Оскільки декомпіляція відбувалась з інструкцій Java-машини у мову програмування Java, то саме цей декомпілятор, для зручності розуміння, додав назви змінним.

Отже, при впровадженні механізмів захисту в код додатку, нам необхідно змінювати назви «нативних» методів та класів, в яких вони використовуються на ті, які не мають спільної мети з їхнім реальним вмістом. І, якщо не використовується обфускація коду, то необхідно також змінювати назви функцій та класів, в які впроваджені механізми захисту. Адже, в такому випадку, якщо зловмисник знайде клас чи функцію, в якому виконується механізм захисту, він може змінити значення, яке повертається, і відімкнути такий механізм захисту, що веде за собою певні наслідки [36]. Наприклад, після аналізу декомпільованого коду було знайдено функцію порівняння, яка повертає true, якщо передані ззовні об’єкти збігаються:

```
public static boolean wft(Object obj, Object obj2) {
    return obj == null ? obj2 == null : obj.equals(obj2);
}
```

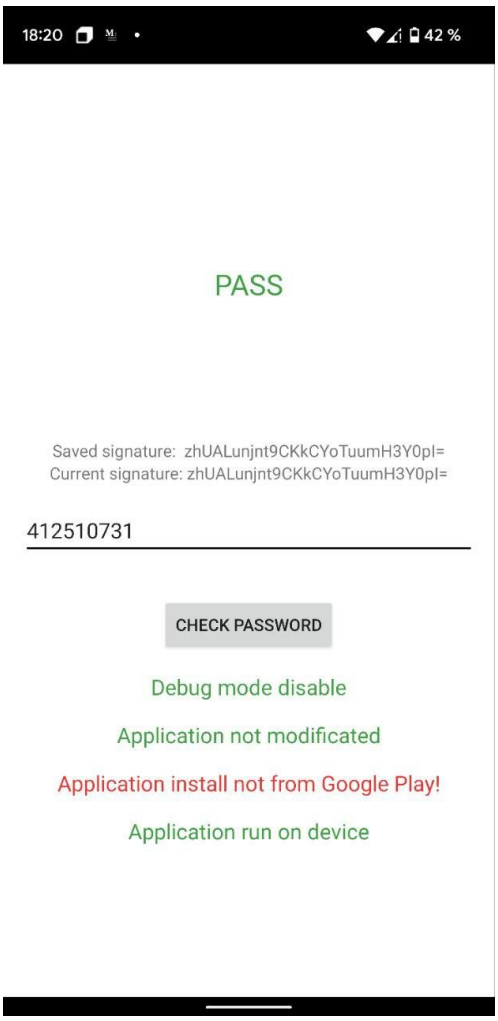

Рисунок 3.3 – Знімок екрану розробленого додатку на фізичному пристрої

На рисунку 3.3 наведено знімок екрану створеного додатку та запущеного на фізичному пристрої без модифікацій. Додаток було інстальовано через APK-пакет, який було створено з наведеного проекту. Можна побачити, що всі механізми захисту працюють коректно. Відображається помилка про джерело інсталяції, адже даний додаток не було опубліковано у магазин додатків Google Play. Додатково, для перевірки коректності роботи створеної функції розшифрування було введено вірний пароль, можна побачити повідомлення про правильність введення.

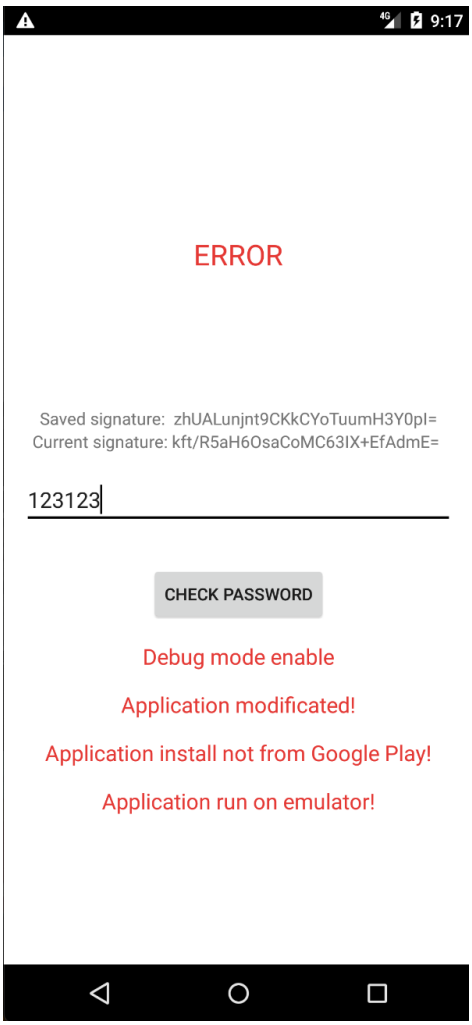

Рисунок 3.4 – Знімок екрану зміненого додатку на емуляторі

Також для перевірки мною було виконано деякі зміни в додатку та змінено цифровий підпис додатку. На рисунку 3.4 наведено знімок екрану модифікованого додатку, який було запущено на емуляторі, котрий вбудовано в інтегроване середовище розробки Android Studio та підключено відладчик. Можна побачити на рисунку, що всі впроваджені механізми захисту працюють коректно і відображаються повідомлення про модифікації, середовище роботи, ввімкнений режим відладки та джерело інсталяції додатку.

Висновки за розділом 3

У третьому розділі було розглянуто набір механізмів, що реалізують засоби захисту додатків на базі операційної системи Android.

Є декілька напрямків захисту додатків: захист вихідного коду, захист від модифікацій та захист від дослідження логіки роботи додатку. Для захисту вихідного коду додатку застосовуються різні механізми заплутування, приховування та шифрування, задля того, щоб при декомпіляції коду додатку було важко зрозуміти його логіку роботи. Наприклад, при приховуванні рядків вони не лежать відкрито в декомпільованому вигляді, а зберігаються у зашифрованому вигляді, що надає їм деякий рівень захисту. Принцип засобу захисту додатку від модифікацій полягає у перевірці його поточного цифрового підпису зі збереженим оригінальним підписом додатку.

Є декілька додаткових ознак того, що додаток було модифіковано або над ним, вірогідніше за все, проводяться деякі маніпуляції для дослідження логіки його роботи: підключений відладчик, додаток інстальовано не з надійних джерел, середовище, в якому виконується додаток (пристрій або емулятор). Необхідно поєднувати ці чинники, або перевіряти окремо, і, за необхідності, приховувати логіку роботи або взагалі аварійно завершувати додаток. Перевірку на ці ознаки необхідно додати у різні місця додатку, щоб після декомпіляції додатку код виглядав ще більш заплутаним та важчим для розуміння.

На основі наведених засобів у розділі 3 мною було розроблено додаток, за допомогою інтегрованого середовища розробки Android Studio, який виводить на екран повідомлення про наявність факторів модифікації додатку. Також створено функції XOR’у рядків для приховування паролю, який порівнюється з введеним паролем користувача, ключ для функції прихований в «нативному» коді. На екрані, для наочності, я додав відображення поточного та збереженого цифрового підпису, з яким відбувається порівняння.

Для підтвердження коректності роботи механізмів захисту було модифіковано створений додаток та запущено на емуляторі. Всі механізми захисту працюють коректно і відображаються відповідні повідомлення для наочності.

Всі створені функції та класи були детально описані. Додатково було проаналізовано вихідний декомпільований код обфускованого та не обфускованого додатку.

ВИСНОВКИ

У дипломній роботі було розглянуто засоби та механізми захисту додатків на базі операційної системи Android. Також на основі розглянутих засобів та механізмів було побудовано власний додаток, в якому було задіяно всі розглянуті механізми, за допомогою інтегрованого середовища Android Studio та мов програмування Kotlin та C++.

У першому розділі дипломної роботи було розглянуто технології, які використовуються в операційній системі Android, як додаток виконується в операційній системі, які має обмеження у середовищі виконання. Було проведено аналіз цілей захисту, яких повинен дотримуватись розробник при проектуванні та розробці мобільного додатку на базі операційної системи Android, а саме: захист даних користувача, захист додатків один від одного, захист передачі конфіденційної інформації. Також у першому розділі було розглянуто модель безпеки операційної системи Android, те, що система має власні механізми захисту: «пісочниця» додатків, система дозволів, цифровий підпис додатків.

У другому розділі дипломної роботи було досліджено найбільш поширені вразливості для додатків на базі операційної системи Android. Із найбільш поширених вразливостей було виділено та розглянуто наступні:

– Небезпечна передача даних
– Небезпечне зберігання даних
– Обхід архітектурних обмежень

Відповідно до розглянутих вразливостей було наведено приклади програмного коду, в якому вони зустрічаються.

У третьому розділі на основні досліджених основних вразливостей було розглянуто набір механізмів, що реалізують засоби захисту вихідного коду та рядків додатку, захисту від модифікацій, захисту від дослідження логіки роботи додатку.

Захист від загроз модифікації коду включає в себе захист від модифікації коду додатку та захист рядків при дослідженні додатку. Для захисту від модифікації застосовується перевірка поточного цифрового підпису додатку зі збереженим.

Захист від загроз дослідження логіки роботи додатку реалізується через обфускацію всього коду. Також код, створений на мовах C/C++ та вбудований в додаток, важко піддається дослідженню, через те, що він представляє собою машинні інструкції. Для захисту рядків використовується різне шифрування рядків, їхнє заплутування. Реалізація виявлення середовища роботи додатку або виявлення ввімкненого режиму відладки дозволяє передбачити процес дослідження додатку, в цей час приховуючи свій справжній функціонал.

Також було розроблено додаток та впроваджено в нього розглянуті механізми захисту. Для перевірки роботи впроваджених механізмів захисту було модифіковано додаток та запущено його на емуляторі.

Виходячи із поставленої мети дипломної роботи були виконані наступні завдання:

– розглянуто загальну структуру, модель захисту операційної системи Android та цілі захисту додатків;

– проведено опис найбільш поширених вразливостей, характерних для додатків на базі операційної системи Android;

– реалізовано певний набір механізмів та засобів для захисту додатків на базі операційної системи Android.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Що таке операційна система Android? [Електронний ресурс]: Theastrologypage – Режим доступу: https://uk.theastrologypage.com/android-operating-system.

2. Building Trust in Mobile Apps & Services: Data Privacy is Key [Electronic resource]: Marketingcharts – Access: https://www.marketingcharts.com/cross-media-and-traditional/word-of-mouth-78938.

3. The Core Of Android Architecture [Electronic resource]: Bartleby – Access: https://www.bartleby.com/essay/The-Core-Of-Android-Architecture-PKAZNLNLMWW.

4. Implementing ART Just-In-Time (JIT) Compiler [Electronic resource]: Android Open Source – Access: https://source.android.com/devices/tech/dalvik/jit-compiler.

5. Android Memory Overview [Part1] - Introduction to Low Memory Killer [Electronic resource]: Strayinsights – Access: https://strayinsights.blogspot.com/2020/05/android-memory-overview-part1.html.

6. Як Android “Doze” покращує час роботи від акумулятора [Електронний ресурс]: Thefastcode – Access: https://www.thefastcode.com/uk-uah/article/how-android-s-doze-improves-your-battery-life-and-how-to-tweak-it?

7. App Standby Buckets [Electronic resource]: Android Developers – Access: https://developer.android.com/topic/performance/appstandby

8. Binder Security [Electronic resource]: Medium– Access: https://baiqin-droid1001.medium.com/binder-security-f93265d2ec1b

9. Does Android's Open Source Status Make it Better in Terms of Privacy? [Electronic resource]: Android Authority – Access: https://www.androidauthority.com/does-androids-open-source-status-make-it-better-in-terms-of-privacy-62130/

10. Application Sandbox [Electronic resource]: Android Open Source – Access: https://source.android.com/security/app-sandbox

11. Interception of Android implicit intents [Electronic resource]: Oversecured – Access: https://blog.oversecured.com/Interception-of-Android-implicit-intents/

12. Permissions on Android [Electronic resource]: Android Developers – Access: https://developer.android.com/guide/topics/permissions/overview

13. Application Signing [Electronic resource]: Android Open Source – Access: https://source.android.com/security/apksigning

14. OWASP Mobile Top 10 [Electronic resource]: Owasp – Access: https://owasp.org/www-project-mobile-top-10/

15. Security with HTTPS and SSL [Electronic resource]: Android Developers – Access: https://developer.android.com/training/articles/security-ssl

16. OAuth 2 on Android [Electronic resource]: Medium – Access: https://android2ee.medium.com/oauth-2-on-android-principles-using-google-apis-afb5cc12d935

17. Access app-specific files [Electronic resource]: Android Developers – Access: https://developer.android.com/training/data-storage/app-specific

18. Path Traversal Vulnerability [Electronic resource]: Google – Access: https://support.google.com/faqs/answer/7496913

19. An overview of Android Keystore System [Electronic resource]: Codementor – Access: https://www.codementor.io/@roaim/android-keystore-system-z5gwp75o1

20. Exploiting Exported activities in Android apps [Electronic resource]: Mzfr – Access: https://blog.mzfr.me/posts/2020-11-07-exported-activities/

21. Using Technology in a Safe Way [Electronic resource]: Jssec – Access: https://www.jssec.org/dl/android_securecoding_en/4_using_technology_in_a_safe_way.html

22. Android: Proguard, D8, R8 what are they? [Electronic resource]: Medium – Access: https://nphau.medium.com/android-journey-proguard-d8-r8-what-are-they-e8f2bfe079a7

23. XOR Cipher [Electronic resource]: Geeksforgeeks – Access: https://www.geeksforgeeks.org/xor-cipher/

24. Concepts [Electronic resource]: Android Developers – Access: https://developer.android.com/ndk/guides/concepts

25. Reverse Engineering APK Files in Android Apps [Electronic resource]: Study – Access: https://study.com/academy/lesson/reverse-engineering-apk-files-in-android-apps.html

26. Inspecting APK Files [Electronic resource]: Tomas Surin – Access: https://pspdfkit.com/blog/2019/inspecting-apk-files/

27. Android Security: Adding Tampering Detection to Your App [Electronic resource]: Airpair – Access: https://www.airpair.com/android/posts/adding-tampering-detection-to-your-android-app

28. Checking if an Android app is installed via Google Play [Electronic resource]: Yellowduck – Access: https://www.yellowduck.be/posts/checking-if-android-app-installed-google-play

29. How to exploit a debuggable Android application [Electronic resource]: Securitygrind – Access: https://securitygrind.com/how-to-exploit-a-debuggable-android-application/

30. Android Emulator Detection [Electronic resource]: Medium – Access: https://ray-chong.medium.com/android-emulator-detection-4d0f994aab5e

31. Layouts [Electronic resource]: Android Developers – Access: https://developer.android.com/guide/topics/ui/declaring-layout

32. Configure the NDK for the Android Gradle plugin [Electronic resource]: Android Developers – Access: https:/developer.android.com/studio/projects/configure-agp-ndk

33. Native Development Kit (NDK) [Electronic resource]: NTU – Access: https://www3.ntu.edu.sg/home/ehchua/programming/android/Android_NDK.html

34. Decompile an APK, modify it and then recompile it [Electronic resource]: Getridbug – Access: https://getridbug.com/android/decompile-an-apk-modify-it-and-then-recompile-it/

35. Android Java Native Interface (JNI) [Electronic resource]: Sodocumentation – Access: https://sodocumentation.net/android/topic/8674/android-java-native-interface--jni-

36. Reverse Engineering Android Apps - Native Libraries [Electronic resource]: Getridbug – Access: https://www.ragingrock.com/AndroidAppRE/reversing_native_libs.html