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

«На правах рукопису»
УДК 004.056

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Виявлення вразливостей Android-застосунків із використанням підходів реверсної інженерії

Виконав (-ла): студент (-ка) 2 курсу, групи ФБ-71мп
(шифр групи)

Різник Вадим Олександрович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Галицька Ірина Євгенівна _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Рецензент к.т.н., доцент ФІОТ Жданова О.Г.
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

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

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 125 Кібербезпека (« Системи і технології Кібербезпеки »)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ В.Грайворонський
(підпис)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Різнику Вадиму Олександровичу

1. Тема дисертації Виявлення вразливостей Android-застосунків із використанням підходів реверсної інженерії,

науковий керівник дисертації д.т.н. Галицька Ірина Євгенівна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» листопада 2018 р. № 4171-с

2. Термін подання студентом дисертації 12.12.2018 р.

3. Об'єкт дослідження: технології та засоби тестування на проникнення мобільних застосунків ОС Android

4. Вихідні дані: застосунок, що базується на методах реверсної інженерії та аналізу графу викликів.

5. Перелік завдань, які потрібно розробити:

1. Вивчення структури операційної системи Android та механізмів захисту закладених в неї
2. Дослідження методологій та засобів тестування безпеки Android-застосунків
3. Аналіз недоліків в процесі тестування безпеки застосунків
4. Побудова інструменту аналізу безпеки Android-застосунків

6. Орієнтовний перелік ілюстративного матеріалу 15 слайдів

7. Орієнтовний перелік публікацій публікації відсутні

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Узгодження теми		
2.	Вивчення структури операційної системи Android та механізмів безпеки закладених в неї		
3.	Дослідження методологій та засобів тестування безпеки Android застосунків		
4.	Практичне застосування досліджених методологій, проведення тестування безпеки застосунків		
5.	Аналіз недоліків в процесі тестування безпеки застосунків		
6.	Побудова інструменту аналізу безпеки Android-застосунків		

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Робота об’ємом 113 сторінок, яка містить 42 ілюстрації, 13 таблиць, 7 джерел за переліком посилань та 1 додаток.

Метою даної кваліфікаційної роботи є аналіз сучасних методів та інструментів проведення тестування безпеки Android-застосунків, аналіз ефективності цих методів та інструментів. При виявлені значних недоліків створення рішення для їх усунення, або створення рішення для покращення існуючих методик.

Об’єктом дослідження є технології та засоби тестування захисту Android-застосунків.

Предметом дослідження є забезпечення належного рівня захисту Android-застосунків через проведення якісного тестування на вразливості.

Методом дослідження є опрацювання літератури та інших інформаційних джерел за даною темою, аналіз існуючих методів та засобів захисту інформації та їхніх характеристик, вивчення методів тестування на проникнення, та використання їх на практиці.

Результати роботи можуть бути використані в методології SDL, як інструмент тестування на проникнення, для забезпечення захисту застосунка.

ІНФОРМАЦІЙНА БЕЗПЕКА, ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, БЕЗПЕКА ANDROID-ЗАСТОСУНКІВ, РЕВЕРС-ІНЖИНИРИНГ, ЗАХИСТ ІНФОРМАЦІЇ

РЕФЕРАТ

Работа объемом 113 страниц, содержащая 42 иллюстрации, 13 таблицу, 7 источников по перечню ссылок и 1 приложения.

Целью данной квалификационной работы является анализ современных методов и инструментов проведения тестирования безопасности Android-приложений, анализ эффективности этих методов и инструментов. При обнаружении значительных недостатков создания решения для их устранения, или создание решения для улучшения существующих методик. Объектом исследования являются технологии и средства тестирования защиты Android-приложений.

Предметом исследования является обеспечение надлежащего уровня защиты Android-приложений через проведение качественного тестирования на уязвимости.

Методом исследования является обработка литературы и других информационных источников по данной теме, анализ существующих методов и средств защиты информации и их характеристик, изучение методов тестирования на проникновение, и использование их на практике. Результаты работы могут быть использованы в методологии SDL, как инструмент тестирования на проникновение, для обеспечения защиты приложений.

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ, ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ, БЕЗОПАСНОСТЬ ANDROID-ПРИЛОЖЕНИЙ, РЕВЕРС-ИНЖИНИРИНГ, ЗАЩИТА ИНФОРМАЦИИ

ABSTRACT

This work consists of 113 pages, containing 42 illustrations, 13 table, 7 sources and 1 attachment.

The purpose of this qualification work is to analyze current methods and tools for testing Android-applications security, analysis of the effectiveness of these methods and tools. If significant drawbacks are found, creating a solution to fix them, or creating a solution to improve the existing techniques.

The object of the research is the technologies and means of testing Android application protection.

The subject of the study is to provide an high level of protection for Android applications through quality vulnerability testing.

The research method is the processing of literature and other information sources on this topic, analysis of existing methods and means of protecting information and their characteristics, studying the methods of penetration testing, and their use in practice.

The results of the work can be used in the SDL methodology, as a penetration testing tool, to provide application protection.

INFORMATION SECURITY, INFORMATION TECHNOLOGIES, SAFETY ANDROID APPLICATIONS, REVERSE ENGINEERING, PROTECTION OF INFORMATION

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

SDL – Secure Development Lifecycle

IPC – Inter Process Communication

AIDL – Android Interface Definition Language

JVM – Java virtual machine

ART – Android Runtime

DEX – Dalvik executable

JIT – just-in-time compilation

JNI – Java Native Interface

MAC – mandatory access control

ОС – Операційна Система

БД – База Даних

ВСТУП

Одним з найважливіших компонентів захисту застосунків є їх тестування на проникнення. І невід'ємною частиною цього тестування для нативних застосунків є зворотна розробка. Зворотна розробка є клопітким і одним найскладніших інструментом інженера, він займає левову частину часу, що відводиться для тестування на проникнення. Але попри складність цього інструменту, не можна не відмітити його ефективність.

Для настільних застосунків є давно зарекомендований і надзвичайно зручний інструмент IDA Pro. Але чи є аналогічний інструмент в світі мобільних застосунків (а саме Android)? Одним з основних гравців на цьому ринку є компанія «PNF Software» з своїм інструментом «JEB Decompiler». Але при всій потужності цього інструменту, він дійсно гарний лише в декомпіляції коду – але не пропонує зручних інструментів аналізу результатів. Ця робота направлена на аналіз сучасних прийомів зворотної розробки Android застосунків та створення більш швидких і зручних методів, інструментів, технік направлених саме на ефективне тестування на проникнення.

Актуальність даної кваліфікаційної роботи випливає з необхідності тестування захищеності Android-застосунків, для підвищення рівня безпеки користувача.

Метою даної роботи є дослідження методів та інструментів тестування на захищеність Android-застосунків, з ціллю підвищення якості та ефективності цих сутностей. І в результаті досягнення високої захищеності мобільних застосунків.

Для досягнення мети було поставлено наступні завдання:

- Вивчення роботи ОС Android та механізмів безпеки в ній
- Вивчення інструментів та методик тестування захисту застосунків
- Отримання практичних навичок тестування безпеки Android-застосунків
- Аналіз недоліків сучасних методів та інструментів тестування

- Побудова власного рішення тестування безпеки на основі виявлених недоліків

Об’єктом дослідження є технології та засоби тестування безпеки Android-застосунків.

Предметом дослідження є забезпечення належного рівня тестування безпеки мобільних застосунків.

Методом дослідження є опрацювання літератури та інших інформаційних джерел за даною темою, вивчення механізмів безпеки ОС та методик тестування безпеки застосунків в теорії та застосування отриманих знань на практиці, вивчення недоліків популярних інструментів безпосередньо на практиці.

Наукова новизна одержаних результатів полягає у використанні найбільш вдалих рішень та засобів для аналізу архітектури застосунка на предмет вразливих ділянок, та розгляд графу викликів не як кінцевого продукту аналізу вихідного коду, а розробка методів автоматичного аналізу цього графу.

Практичне значення результатів роботи випливає з того, що створений інструмент дозволить автоматизувати частину ручної роботи тестувальника, оптимізувавши пошук вразливостей.

1 ОПЕРАЦІЙНА СИСТЕМА ANDROID

1.1 Архітектура ОС Android

Android базується на ядрі Linux, але значно відрізняється від більшості інших Linux-систем. Як і в інших Linux-системах, ядро Linux забезпечує такі низькорівневі речі, як управління пам'яттю, захист даних, підтримку мультипроцесності і багатопоточності. Але - за кількома винятками - в Android відсутні інші звичні компоненти GNU/Linux-систем: тут немає нічого від проекту GNU, не використовується X.Org, systemd. Всі ці компоненти замінені аналогами, більш пристосованими для використання в умовах обмеженої пам'яті, низькій швидкості процесора і мінімального споживання енергії - таким чином, Android більше схожий на вбудовану (embedded) Linux-систему, ніж на GNU/Linux.

Саме ядро Linux в Android теж трохи модифіковано: було додано кілька невеликих компонентів, в тому числі ashmem (anonymous shared memory), Binder, wakelocks (управління сплячим режимом) і low memory killer. Початково вони представляли собою патчі до ядра, але їх код був досить швидко доданий назад в upstream-ядро. Вони відсутні в ОС Linux: більшість інших дистрибутивів відключають ці компоненти при збірці.

Як libc (стандартної бібліотеки мови C) в Android використовується не GNU C library (glibc), а власна мінімалістичний реалізація під назвою bionic, оптимізована для вбудованих (embedded) систем - вона значно швидше, менше і менш вимоглива до пам'яті, ніж glibc, яка обросла безліччю шарів сумісності. В Android є оболонка командного рядка (shell) і безліч стандартних для Unix-подібних систем команд / програм. У вбудованих системах для цього зазвичай використовується пакет Busybox, який реалізує функціональність багатьох

команд в одному виконуваному файлі; в Android використовується його аналог під назвою Toybox. Як і в «звичайних» дистрибутивах Linux (і на відміну від вбудованих систем), основним способом взаємодії з системою є графічний інтерфейс, а не командний рядок. Проте, «дістатися» до командного рядка дуже просто - достатньо запустити застосунок-емулятор терміналу. За замовчуванням він зазвичай не встановлено, але його легко, наприклад, завантажити з Play Store (Terminal Emulator for Android, Material Terminal, Termux). У багатьох «просунутих» дистрибутивах Android - таких, як LineageOS (колишній CyanogenMod) - емулятор терміналу встановлено.

Другий варіант - підключитися до Android-пристрою з комп'ютера через Android Debug Bridge (adb). Це дуже схоже на підключення через SSH:

З інших знайомих компонентів в Android використовуються бібліотека FreeType (для відображення тексту), графічні API OpenGL ES, EGL і Vulkan, а також легка СУБД SQLite.

Крім того, раніше для реалізації WebView використовувався браузерні движок WebKit, але починаючи з версії 7.0 замість цього використовується встановлений застосунок Chrome (або інше; список застосунків, яким дозволено виступати в якості WebView provider, конфігурується на етапі компіляції системи). Всередині себе Chrome теж використовує заснований на WebKit движок Blink, але на відміну від системної бібліотеки, Chrome оновлюється через Play Store - таким чином, всі функції, які залежать WebView, автоматично отримують останні поліпшення і виправлення вразливостей.

Рисунок 1.1 – Android Framework

1.2 Будова Android застосунків

Як легко помітити, використання Android принципово відрізняється від використання Linux. Унікальних особливостей Android - в тому, що програми не контролюють безпосередньо процес, в якому вони запущені.

Основна одиниця в Unix-подібних системах - процес. І низькорівневі системні сервіси, і окремі команди в shell, і графічні застосунки - це процеси. У більшості випадків процес являє собою чорний ящик для решти системи - інші компоненти системи не знають і не піклуються про його стан. Процес починає виконуватися з виклику функції main () (насправді _start), і далі реалізує якусь свою логіку, взаємодіючи з рештою системи через системні виклики і найпростішу міжпроцесну взаємодію (IPC).

Оскільки Android теж Unix-подібний, все це вірно і для нього, але в той час як низькорівневі частини - на рівні Unix - оперують поняттям процесу, на більш високому рівні - рівні Android Framework - основною одиницею є застосунок. Застосунок - не чорний ящик: він складається з окремих компонентів, добре відомих решті системи.

У застосунків Android немає функції main (), немає однієї точки входу. Взагалі, Android максимально абстрагує поняття застосунок запущено як від користувача, так і від розробника. Звичайно, процес застосунка потрібно запускати і зупиняти, але Android робить це автоматично (докладніше я розповім про це в наступних статтях). Розробнику пропонується реалізувати кілька окремих компонентів, кожен з яких має свій власний життєвий цикл.

Для реалізації такої системи потрібно, щоб програми мали можливість спілкуватися один з одним і з системними сервісами - іншими словами, потрібен дуже просунутий і швидкий механізм IPC. Цей механізм - Binder.

1.3 Механізм міжпроцесної взаємодії Binder

Binder - це платформа для швидкої, зручної і об'єктно-орієнтованої взаємодії між процесами.

Розробка Binder почалася в Be Inc. (Для BeOS), потім він був портований на Linux і відкритий. Основний розробник Binder, Dianne Hackborn, була і залишається одним з основних розробників Android. За час розробки Android Binder був повністю переписаний.

Binder працює не поверх System V IPC (яке навіть не підтримується в bionic), а використовує свій невеликий модуль ядра, взаємодія з яким з userspace відбувається через системні виклики (в основному ioctl) на «віртуальному пристрої» */dev/binder*. З боку userspace низькорівнева робота з Binder, в тому

числі взаємодія з */dev/binder* і *marshalling/unmarshalling* даних, реалізована в бібліотеці libbinder.

Низькорівневі частини Binder оперують в термінах об'єктів, які можуть пересилатися між процесами. При цьому використовується підрахунок посилань (reference-counting) для автоматичного звільнення невикористовуваних загальних ресурсів і повідомлення про завершення віддаленого процесу (link-to-death) для звільнення ресурсів всередині процесу.

Високорівневі частини Binder працюють в термінах інтерфейсів, сервісів і проксі-об'єктів. Опис інтерфейсу, що надається програмою іншим програмам, записується на спеціальній мові AIDL, зовні дуже схожому на оголошення інтерфейсів в Java. З цього опису автоматично генерується справжній Java-інтерфейс, який потім може використовуватися і клієнтами, і самим сервісом. Крім того, з .aidl-файлу автоматично генеруються два спеціальних класи: Proxy (для використання з боку клієнта) і Stub (з боку сервісу), що реалізують цей інтерфейс.

Для Java-коду в процесі-клієнті проксі-об'єкт виглядає як звичайний Java-об'єкт, який реалізує наш інтерфейс, і цей код може просто викликати його методи. При цьому в згенеровану реалізацію проксі-об'єкта автоматично серіалізуються передані аргументи, спілкуються з процесом-сервісом через libbinder, десеріалізує переданий результат виклику і повертає його з Java-методу.

Stub працює навпаки: він приймає вхідні виклики через libbinder, десеріалізує аргументи, викликає абстрактну реалізацію методу, серіалізує повернене значення і передає його процесу-клієнта. Відповідно, для реалізації сервісу програмісту досить реалізувати абстрактні методи в успадкованому від Stub класі.

Така реалізація Binder на рівні Java дозволяє більшості коду використовувати проксі-об'єкт, взагалі не замислюючись про те, що його функціональність реалізована в іншому процесі. Для забезпечення повної прозорості Binder підтримує вкладені і рекурсивні міжпроцесні виклики. Більш

того, використання Binder з боку клієнта виглядає абсолютно однаково, незалежно від того, чи розташована реалізація використовуваного сервісу в тому ж або в окремому процесі.

Для того, щоб різні процеси могли «знайти» сервіси один одного, в Android є спеціальний сервіс ServiceManager, який зберігає, реєструє і видає токени всіх інших сервісів.

Binder широко використовується в Android для реалізації системних сервісів (наприклад, пакетного менеджера і буфера обміну), але деталі цього приховані від розробника застосунків високорівневими класами в Android Framework, такими як Activity, Intent і Context. Застосунки можуть також використовувати Binder для надання один одному власних сервісів - наприклад, застосунок Google Play Services взагалі не має власного графічного інтерфейсу для користувача, але надає розробникам інших програм можливість користуватися сервісами Google Play.

1.4 Мова програмування превалююча в ОС Android

Хоча використання високорівневих мов для серйозної розробки зараз вже нікого не дивує, з популярних операційних систем тільки у Android «рідна» мова - Java.

Переваги мови програмування JAVA:

По-перше, Java - найпопулярніша мова програмування. У Java величезна екосистема бібліотек і інструментів розробки (в тому числі систем збирання і IDE). Про Java написано безліч статей, книг і документації. Нарешті, існує безліч кваліфікованих Java-розробників.

Програми на Java, як і на багатьох інших високорівневих мовах, переносимі між операційними системами і архітектурами процесора. Практично це проявляється, наприклад, в тому, що застосунки для Android працюють без

перекомпіляції на пристроях будь-якої архітектури (Android підтримує ARM, ARM64, x86, x86-64 і MIPS).

На відміну від низькорівневих мов на зразок C і C++, що використовують ручне управління пам'яттю, в Java пам'ять автоматично управляється середовищем часу виконання (runtime environment). Програма на Java навіть не має прямого доступу до пам'яті, що автоматично запобігає кільком класам помилок, що часто призводять до падінь і вразливостей в програмах, написаних на низькорівневих мовах - «висячі посилання» (через які відбувається use-after-free), розіменування нульового покажчика (при спробі це зробити викидається NullPointerException), читання неініціалізованої пам'яті і вихід за межі масиву.

Використання повноцінної чистки сміття (в порівнянні з automatic reference counting) позбавляє програміста від всіх проблем і складнощів з циклічними посиланнями і дозволяє реалізовувати ще більш просунуті (advanced) залежності між об'єктами.

Це робить розробку під Android більш приємною, ніж розробку з використанням низькорівневих мов, а застосунки під Android набагато більш надійними, в тому числі і з точки зору безпеки.

1.5 Середовище виконання застосунків

На відміну від більшості інших високорівневих мов, програми на Java не поширюються в вигляді вихідного коду, а компілюються в проміжний формат (байткод), який представляє собою виконуваний бінарний код для спеціального процесора.

Хоча робляться спроби створити фізичний процесор, який виконував би Java-байткод безпосередньо, в переважній більшості випадків в якості такого процесора використовується емулятор - JVM. Зазвичай використовується реалізація від Oracle/OpenJDK під назвою HotSpot.

В Android використовується власна реалізація під назвою ART, спеціально оптимізована для роботи на мобільних пристроях. У старих версіях Android (до 5.0 Lollipop) замість ART використовувалася інша реалізація під назвою Dalvik.

І в Dalvik, і в ART використовується власний формат байткода і власний формат файлів, в яких зберігається байткод - DEX. На відміну від class-файлів в «звичайної Джаві», весь Java-код програми зазвичай компілюється в один DEX-файл classes.dex. При побудові Android-застосунка Java-код спочатку компілюється звичайним компілятором Java в class-файли, а потім конвертується в DEX-файл необхідні інструменти (можливо і зворотне перетворення).

І HotSpot, і Dalvik, і ART застосунково оптимізують виконується код. Всі три використовують JIT, тобто під час виконання компілюють байткод в шматки повністю нативного коду, який виконується безпосередньо. Крім очевидного виграшу в швидкості, це дозволяє оптимізувати код для виконання на конкретному процесорі, не відмовляючись від повної переносимості байткода.

Крім того, ART може компілювати байткод в нативний код заздалегідь, а не під час виконання (ahead-of-time compilation) - причому система автоматично планує цю компіляцію на той час, коли пристрій не використовується і підключено до зарядки (наприклад, вночі). При цьому ART враховує дані, зібрані профілювальником під час попередніх запусків цього коду (profile-guided optimization). Такий підхід дозволяє розраховувати код під специфіку роботи конкретного застосунка і навіть під особливості використання цього застосунка саме цим користувачем.

В результаті всіх цих оптимізацій продуктивність Java-коду на Android не сильно поступається продуктивності низькорівневого коду (на C/C ++), а в деяких випадках і перевершує її.

Java-байткод, на відміну від звичайного виконуваного коду, використовує об'єктну модель Java - тобто в байткод явно записуються такі речі, як класи, методи і сигнатури. Це уможливлює компіляцію інших мов в Java-байткод, що

дозволяє написаним на них програмами виконуватися на віртуальній машині Java і бути в тій чи іншій мірі сумісними (interoperable) з Java.

Існують як JVM-реалізації незалежних мов - наприклад, Jython для Python, JRuby для Ruby, Rhino для JavaScript і діалект Lisp Clojure - так і мови, початково розроблені для компіляції в Java-байткод і виконання на JVM, найвідоміші з яких - Groovy, Scala і Kotlin.

Найновіший з них, Kotlin, спеціально розроблений для ідеальної сумісності з Java і володіє набагато більш приємним синтаксисом (схожим на Swift), підтримується Google як офіційна мова розробки під Android нарівні з Java.

Незважаючи на всі переваги Java, в деяких випадках все-таки бажано використовувати низькорівневий мову - наприклад, для реалізації критичного по продуктивності компонента, такого як браузерний движок, або щоб використати існуючу нативную бібліотеку. Java дозволяє викликати нативний код через JNI, і Android надає спеціальні засоби для нативної розробки - Native Development Kit (NDK), до якого входять в тому числі заголовки, компілятор (Clang), відладчик (LLDB) і система збирання.

Хоча NDK в основному орієнтований на використання C/C++, з його допомогою можна писати під Android та іншими мовами - в тому числі Rust, Swift, Python, JavaScript і навіть Haskell. Більше того, є навіть можливість перенести iOS-застосунки (написання на Objective-C або Swift) на Android практично без змін.

1.6 Безпека Android

Засоби безпеки UNIX

Модель безпеки в класичному Unix заснована на системі UID/GID - спеціальних номерів, які ядро зберігає для кожного процесу. Процесам з однаковим UID дозволений доступ один до одного, процеси з різним UID ізольовані один від одного. Аналогічно обмежується доступ до файлів.

За змістом кожен UID (user ID) відповідає своєму користувачеві - за часів створення Unix була нормальною ситуація, коли один комп'ютер одночасно використовувався багатьма людьми. Таким чином, в Unix процеси і файли різних людей були захищені один від одного. Щоб надати спільний доступ до деяких файлів, користувачі об'єднувалися в групи, яким і відповідав GID (group ID).

При цьому всім програмам, що запускаються користувачем, дається повний доступ до всього, до чого є доступ у цього користувача. Власне, оскільки користувач не може спілкуватися з ядром безпосередньо, а взаємодіє з комп'ютером через shell і інші процеси - права користувача і є права програм, запущених від його імені.

Така модель має на увазі, що користувач повністю довіряє всім програмам, які використовує. У той час це було логічно, тому що програми найчастіше або були частиною системи, або створювалися (писалися і компілювалися) самим користувачем.

В Unix є і виключення з обмежень доступу - UID 0, який прийнято називати root. У нього є доступ до всього в системі, і ніякі обмеження на нього не поширюються. Цей обліковий запис використовувався системним адміністратором; крім того, під UID 0 запускаються багато системних сервісів.

У сучасному Linux ця модель була значно розширена і узагальнена, в тому числі з'явилися capabilities, що дозволяють «отримати частину root-прав», і реалізує мандатний управління доступом (MAC) підсистема SELinux, яка дозволяє застосунку обмежити права (в тому числі права root-процесів).

Все змінилось за кілька десятків років, що минули з створення Unix до створення Android, практика використання комп'ютерів («обчислювачів») значно змінилася.

Замість машин, розрахованих на паралельне використання багатьма користувачами (через термінали - то, що зараз емулюють емулятори терміналів), з'явилися персональні комп'ютери, розраховані на використання

однією людиною. З появою мобільних пристроїв - спочатку КПК, потім смартфонів, планшетів, розумних годинників - ця тенденція тільки посилилася.

На таких пристроях зберігаються гігабайти персональної інформації, доступ до якої повинен бути захищений і обмежений. У той же час розширився ринок сторонніх застосунків, яким у користувача немає ніяких підстав довіряти.

Таким чином, в сучасних умовах замість захисту різних користувачів один від одного необхідно захищати від застосунків інші застосунки, призначені для користувача дані і саму систему. Крім того, широке поширення набули віруси, які зазвичай використовують уразливості в системі - для захисту від них потрібно захищати частини системи один від одного за допомогою застосунків, щоб використання однієї уразливості не давало зловмисникові доступ до всієї системи.

Механізми безпеки ОС Android

Хоча частина Android-застосунків поставляється з системою - наприклад, такі стандартні застосунки, як Калькулятор, Годинники та Камера - більшість застосунків користувачі встановлюють зі сторонніх джерел. Найвідоміший з них - Google Play Store, але є й інші. Крім того, Android дозволяє вручну встановлювати довільні програми з APK-файлів (sideloading).

Як і інші Unix-подібні системи, Android використовує для обмеження доступу існуючий механізм UID/GID. При цьому - на відміну від традиційного використання, коли UID відповідають користувачам - в Android різні UID відповідають різним програмам. Оскільки процеси різних застосунків запускаються з різними UID, вже на рівні ядра застосунки захищені і ізольовані один від одного і не мають доступу до системи і даних користувача. Це утворює пісочницю (Application Sandbox) і дозволяє користувачеві встановлювати будь-які застосунки без необхідності довіряти їм.

Щоб все-таки отримати доступ до призначених для користувача даних, камері, дзвінків і т.д., застосунок повинен отримати від користувача дозвіл (permission). Деякі з дозволів існують у вигляді GID, в які застосунок додається, коли отримує цей дозвіл - наприклад, отримання дозволу ACCESS_FM_RADIO поміщає застосунок до групи media, що дозволяє йому отримати доступ до файлу /dev/fm. Решта існують тільки на більш високому рівні (у вигляді записів в файлі packages.xml) і перевіряються іншими компонентами системи при зверненні до високорівневого API через Binder.

Невелика частина системних сервісів в Android запускається під UID 0, тобто root, але більшість використовують спеціально виділені номери UID, підвищуючи при необхідності свої права за допомогою Linux capabilities. Крім того, Android використовує SELinux - використання SELinux в Android називають SEAndroid - для ще більшого обмеження того, які дії дозволено виконувати застосункам і системним сервісам.

Зазвичай Android не надає користувачеві прямий доступ до root-аккаунту, але в деяких випадках він може цей доступ отримати.

Крім того, веб-застосунки існують у вигляді сторінок, які можуть посилатися один на одного - як в рамках одного сайту, так і між сайтами. При цьому сторінка на одному сайті не зобов'язана обмежуватися посиланням тільки на головну сторінку іншого, вона може посилатися на конкретну сторінку всередині іншого сайту (deep linking). Посилаючись один на одного, окремі сайти об'єднуються в загальну мережу, веб.

Кілька копій однієї сторінки - наприклад, кілька профілів в соціальній мережі - можуть бути одночасно відкриті в декількох вкладках браузера. Інтерфейс браузера розрахований на перемикання між одночасними сесіями (вкладками), а не між окремими сайтами - в рамках однієї вкладки ви можете переміщатися по посиланнях (і вперед-назад по історії) між різними сторінками різних сайтів.

Все це протиставляється «десктопу», де кожен застосунок працює окремо і часто незалежно від інших - і в цьому плані те, як влаштовані застосунки в

Android, набагато ближче до web-застосунків, ніж до «традиційних» застосунків.

1.7 Базові компоненти застосунків Android

Основний вид компонентів застосунків під Android - це activity. Activity - це один «екран» застосунка. Activity можна порівняти з web-сторінкою і з вікном застосунка в традиційному віконному інтерфейсі.

Наприклад, в застосунку для електронної пошти (email client) можуть бути такі activity, як Inbox Activity (список вхідних листів), Email Activity (читання одного листа), Compose Activity (написання листа) і Settings Activity (настройки).

Як і сторінки одного сайту, activity однієї програми можуть запускатися як один з одного, так і незалежно один від одного (іншими застосунками). Якщо в інтернеті на іншу сторінку звертаються по URL (посиланням), то в Android activity запускаються через intent'и.

Intent - це повідомлення, яке вказує системі, що потрібно «зробити» (наприклад, відкрити даний URL, написати лист на цю адресу, зателефонувати на даний номер телефону або зробити фотографію).

Застосунок може створити такий intent і передати його системі, а система вирішує, яка activity (або інший компонент) буде його виконувати (handle). Ця activity запускається системою (в існуючому процесі застосунки або в новому, якщо він ще не запущений), їй передається цей intent, і вона його виконує.

Стандартний спосіб створювати intent'и - через відповідний клас в Android Framework. Для роботи з activity і intent'ами з командного рядка в Android є команда am - обгортка над стандартним класом Activity Manager: Intent'и можуть бути явними (explicit) і неявними (implicit). Явний intent вказує ідентифікатор конкретного компонента, який потрібно запустити - найчастіше

це використовується, щоб запустити з однієї activity іншу всередині однієї програми (при цьому intent може навіть не містити іншої корисної інформації).

Неявний intent обов'язково повинен вказувати дію, яку потрібно зробити. Кожна activity (і інші компоненти) вказують в маніфесті застосунка, які intent'и вони готові обробляти (наприклад, ACTION_VIEW для посилань з доменом https://example.com). Система вибирає відповідний компоненцФт серед встановлених і запускає його.

Якщо в системі є декілька activity, які готові обробити intent, користувачеві буде наданий вибір. Зазвичай це трапляється, коли встановлено кілька аналогічних програм, наприклад кілька браузерів або фоторедакторів. Крім того, застосунок може явно попросити систему показати діалог вибору (насправді при цьому переданий intent обертається в новий intent з ACTION_CHOOSER) - це зазвичай використовується для створення красивого діалогу Share: Крім того, activity може повернути результат в викликала її activity. Наприклад, activity в застосунку-камері, яка вміє обробляти intent «зробити фотографію» (ACTION_IMAGE_CAPTURE) повертає зроблену фотографію в ту activity, яка створила цей intent.

При цьому з застосунком, який містить вихідну activity, не потрібен дозвіл на доступ до камери.

Таким чином, правильний спосіб застосунком під Android зробити фотографію - це не вимагати дозволу на доступ до камери і використовувати Camera API, а створити потрібний intent і вони надають системному застосунку-камері зробити фото. Аналогічно, замість використання дозволу READ_EXTERNAL_STORAGE і прямого доступу до файлів користувача варто дати користувачеві можливість вибрати файл в системному файловому менеджері (тоді вихідного застосунком буде дозволений доступ саме до цього файлу).

При цьому «системний» застосунок - не обов'язково той, який було встановлено виробником (або автором збірки Android). Всі встановлені застосунки, які вміють обробляти даний intent, в цьому сенсі рівні між собою.

Користувач може вибрати будь-який з них в якості застосунку за замовчуванням для таких intent'ів, а може вибирати потрібну кожен раз. Вибрана програма стає «системною» в тому сенсі, що користувач вибрав, щоб саме воно виконувало всі завдання (тобто intent'и) такого типу, що виникають в системі.

Саме дозвіл на доступ до камери потрібно тільки тим програмам, які реалізують свій інтерфейс камери - наприклад, власне застосунки-камери, застосунки для відеодзвінків або доповненої реальності. Навпаки, звичайному месенджер доступ до камери «щоб можна було фото відправляти» не потрібен, як не потрібен і доступ до скоєння дзвінків з застосунком великого банку.

1.8 Ізольованість застосунків

Багато операційних систем діляться на власне операційну систему і програми, встановлені поверх, нічого один про одного не знають і не вміють взаємодіяти. Система компонентів і intent’ів Android дозволяє застосункам, як і раніше абсолютно нічого один про одного не знаючи, складати для користувача один інтегрований системний user experience - встановлені застосунки реалізують частини однієї великої системи, вони складають з себе систему. І це, з одного боку, відбувається прозоро для користувача, з іншого - представляє необмежені можливості для кастомізації.

1.9 Навігація по застосункам

В браузері користувач може перемикатися не між сайтами, а між вкладками, історія кожної з яких може містити багато сторінок різних сайтів. Аналогічно, в Android користувач може перемикатися між завданнями (tasks), які відображаються у вигляді карток на recents screen. Кожне завдання являє собою back stack - кілька activity, «накладених» один на одного.

Коли одна activity запускає іншу, нова activity поміщається в стек поверх старої. Коли верхня activity в стеці завершується - наприклад, коли користувач натискає системну кнопку «назад» - попередня activity в стеці знову відображається на екрані.

Кожен стек може включати в себе activity з різних застосунків, і кілька копій однієї activity можуть бути одночасно відкриті в рамках різних завдань або навіть всередині одного стека.

При запуску нової activity можуть бути вказані спеціальні прапори, такі як singleTop, singleTask, singleInstance і CLEAR_TOP, які модифікують цей механізм. Наприклад, програми-браузери зазвичай дозволяють запуск тільки однієї копії своєї основної activity, і для перемикання між відкритими сторінками реалізують власну систему вкладок. З іншого боку, Custom Tabs - приклад activity в браузері (найчастіше Chrome), яка поводиться майже «як завжди», тобто показує тільки одну сторінку, але дозволяє одночасно відкривати кілька своїх копій.

1.10 Життєвий цикл застосунків

Одне з основних обмежень вбудованих і мобільних пристроїв - невелика кількість оперативної пам'яті (RAM). Якщо сучасні флагманські пристрої вже оснащуються декількома гігабайтами оперативної пам'яті, то в першому смартфоні на Android, HTC Dream (він же T-Mobile G1), що вийшов у вересні 2008 року, її було всього 192 мегабайта.

Проблема обмеженою пам'яті застосунків ускладнюється тим, що в мобільних пристроях, на відміну від «звичайних» комп'ютерів, не використовуються swap-розділи (і swap-файли) - в тому числі і з-за низької (в порівнянні з SSD і HDD) швидкості доступу до SD-карт і вбудованої флеш-пам'яті, де вони могли б розміщуватися. Починаючи з версії 4.4 KitKat, Android використовує zRAM swap, тобто ефективно стискає маловикористовувані ділянки пам'яті. Проте, проблема обмеженою пам'яті залишається.

Якщо всі процеси представляють собою для системи чорний ящик, найкраща з можливих стратегія поведінки в разі нестачі вільної пам'яті - примусово завершувати якісь процеси, що і робить Linux Out Of Memory (OOM) Killer. Але Android знає, що відбувається в системі, йому відомо, які програми і які їх компоненти запущені, що дозволяє реалізувати набагато більш «розумну» схему звільнення пам'яті.

По-перше, коли вільна пам'ять закінчується, Android явно просить застосунки звільнити непотрібну пам'ять (наприклад, скинути кеш), викликаючи методи onTrimMemory / onLowMemory. По-друге, Android може ефективно проводити збірку сміття в фонових застосунках, щоб звільнити пам'ять, яку вони більше не використовують (на рівні Java), при цьому не сповільнюючи роботу поточної програми.

Але основний механізм звільнення пам'яті в Android - це завершення найменш використовуваних застосунків. Система автоматично вибирає застосунки, найменш важливі для користувача (наприклад, ті, з яких користувач давно пішов), дає їх компонентам шанс звільнити ресурси, викликаючи такі методи, як onDestroy, і завершує їх, повністю звільняючи використовувану ними пам'ять і ресурси.

Якщо користувач повертається в activity застосунки, завершеного системою через брак пам'яті, ця activity запускається знову. При цьому перезапуск відбувається прозоро для користувача, оскільки activity зберігає свій стан при завершенні (onSaveInstanceState) і відновлює його при наступному запуску. Реалізовані в Android Framework віджети використовують цей механізм, щоб автоматично зберегти стан інтерфейсу (UI) при перезапуску - з точністю до введеного в EditText тексту, положення курсору, позиції прокрутки (scroll) і т.д. Розробник програми може реалізувати збереження і відновлення якихось даних, специфічних для цього застосунка.

Підкреслю, що Android може перезапускати програми не повністю, а покомпонентно, залишаючи невикористовувані частини завершеними -

наприклад, з двох копій однієї activity одна може бути перезапущено, а інша залишитися завершеною.

З точки зору користувача цей механізм схожий на використання swap: в обох випадках при поверненні частині програми доводиться трохи почекати, поки вона завантажується знову - в одному випадку, з диска, в іншому - перестворюється по збереженому стану.

Саме цей механізм автоматичного скидання та відновлення стану створює у користувача відчуття, що застосунки «запущені завжди», позбавляючи його від необхідності явно запускати і закривати застосунки і зберігати введені в них дані.

1.11 Сервіси в ОС Android

Застосункам може знадобитися виконувати дії, не пов'язані безпосередньо з жодною activity, в тому числі, продовжувати робити їх у фоновому режимі, коли всі activity цього застосунка завершені. Наприклад, застосунок може завантажувати з мережі великий файл, обробляти фотографії, відтворювати музику, синхронізувати дані або просто підтримувати TCP-з'єднання з сервером для отримання повідомлень.

Таку функціональність можна реалізовувати, просто запускаючи окремий потік - це було б для системи чорним ящиком; в тому числі, процес був би завершений при завершенні всіх activity, незалежно від стану таких фонових операцій. Замість цього Android пропонує використовувати ще один вид компонентів - сервіс.

Сервіс потрібен, щоб повідомити системі, що в процесі застосунка виконуються дії, які не є частиною activity цього застосунка. Сам по собі сервіс не означає створення окремого потоку або процесу - його точки входу (entry points) запускаються в основному потоці застосунка. Зазвичай реалізація сервісу запускає потоки і управляє ними самостійно.

Сервіси багато в чому схожі на activity: вони теж запускаються за допомогою intent’ів і можуть бути завершені системою при нестачі пам'яті.

Запущені сервіси можуть бути в трьох станах:

Foreground service - сервіс, який виконує дію, стан якого важливо для користувача, наприклад, завантаження файлу або відтворення музики. Такий сервіс зобов'язаний відображати повідомлення в системній шторці повідомлень (приклади: стан завантаження, назва поточної пісні та управління відтворенням). Система вважає такий сервіс приблизно настільки ж важливим для користувача, як і поточна activity, і завершить його тільки в крайньому випадку.

Background service - сервіс, який виконує фонову дію, стан якої не цікавить користувача (найчастіше, синхронізацію). Такі сервіси можуть бути завершені при нестачі пам'яті з набагато більшою ймовірністю. У старих версіях Android велика кількість одночасно запущених фонових сервісів часто ставало причиною «гальм»; починаючи з версії 8.0 Oreo, Android серйозно обмежує використання фонових сервісів, примусово завершувати виконання через кілька хвилин після того, як користувач виходить з програми.

Bound service - сервіс, який обробляє входить Binder-підключення. Такі сервіси надають якусь функціональність для інших застосунків або системи (наприклад, WallpaperService і Google Play Services). В цьому випадку система може автоматично запускати сервіс при підключенні до нього клієнтів і зупиняти його при їх відключенні.

Рекомендований спосіб виконувати фонові дії - використання JobScheduler, системного механізму планування фонової роботи. JobScheduler дозволяє застосунку вказати критерії запуску сервісу, такі як:

Доступність мережі. Тут застосунок може вказати, чи потрібно цьому сервісу наявність мережевого підключення, і якщо так, то чи можлива робота в роумінгу або при використанні лімітного (metered) підключення.

Підключення до джерела живлення, що дозволяє сервісів виконуватися, не «саджаючи батарею».

Бездіяльність (idle), що дозволяє сервісам виконуватися, поки пристрій не використовується, що не сповільнює роботу під час активного використання.

Оновлення контенту - наприклад, поява нової фотографії. Період і крайній термін запуску - наприклад, очищення кеша може проводитися щодня, а синхронізація подій в календарі - щогодини.

JobScheduler планує виконання (реалізоване як виклик через Binder) зареєстрованих в ньому сервісів відповідно до зазначених критеріїв. Оскільки JobScheduler - загальносистемний механізм, він враховує при плануванні критерії зареєстрованих сервісів всіх встановлених застосунків. Наприклад, він може запускати сервіси по черзі, а не одночасно, щоб запобігти різкому навантаженню на пристрій під час використання, і планувати періодичне виконання декількох сервісів невеликими групами (batch), щоб запобігти постійне енерговитратне включення-виключення радіообладнання.

1.12 TCP-з'єднання

Компоненти бродкаст ресівер і контент провайдер

Крім activity і сервісів, у застосунків під Android є два інших виду компонентів, менш цікавих для обговорення - це broadcast receiver'и і content provider'и.

Broadcast receiver - компонент, що дозволяє з застосунком приймати broadcast'и, спеціальний вид повідомлень від системи або інших застосунків. Початково broadcast'и, як випливає з назви, в основному використовувалися для розсилки широкомовних повідомлень всім підписаних застосунків - наприклад, система посилає повідомлення AIRPLANE_MODE_CHANGED при включенні або відключенні режиму в літаку.

Зараз замість підписки на такі broadcast'и, як NEW_PICTURE і NEW_VIDEO, застосунки повинні використовувати JobScheduler. Broadcast'и використовуються або для більш рідкісних подій (таких як

BOOT_COMPLETED), або з явними intent'ами, тобто саме в якості повідомлення від однієї програми до іншого.

Content provider - компонент, що дозволяє з застосунком надавати іншим програмам доступ до даних, якими воно управляє. Приклад даних, доступ до яких можна отримати таким чином - список контактів користувача.

При цьому застосунок може зберігати самі дані яким завгодно чином, в тому числі на пристрої у вигляді файлів, у цій базі даних (SQLite) або запитувати їх з сервера по мережі. У цьому сенсі content provider - це уніфікований інтерфейс для доступу до даних, незалежно від форми їх зберігання.

Взаємодія з content provider'ом багато в чому схоже на доступ до віддаленої бази даних через REST API. Застосунок-клієнт запитує дані по URI (наприклад, content://com.example.Dictionary.provider/words/42) через ContentResolver. Застосунок-сервер визначає, до якого саме набору даних був зроблений запит, використовуючи UriMatcher, і виконує запитану дію (query, insert, update, delete).

Саме поверх content provider'ов реалізований Storage Access Framework, що дозволяє застосункам, що зберігають файли в хмарі (наприклад, Dropbox і Google Photos) надавати доступ до них іншим застосункам, не займаючи місце на пристрої повною копією всіх зберігаються в хмарі файлів.

1.13 Управління пакетами ОС

Архітектура Android побудована навколо застосунків. Саме застосунки грають ключову роль в устрої багатьох частин системи, саме для гармонійної взаємодії застосунків вибудувана модель activity і intent’ів, саме на ізоляції застосунків заснована модель безпеки Android. І якщо управлінням взаємодії компонентів застосунків займається activity manager, то за встановлення, оновлення та управління правами застосунків відповідає package manager (пакетний менеджер - в shell його можна викликати командою pm).

Як підказує сама назва «пакетний менеджер», на цьому рівні застосунку часто називаються пакетами. Пакети поширюються в форматі APK (Android package) - спеціальних zip-архівів. У кожного пакета є ім'я (також відоме як application ID), яке унікально ідентифікує це застосунок (але не його конкретну версію - навпаки, імена різних версій пакету повинні збігатися, інакше вони будуть вважатися окремими пакетами). Імена пакетів прийнято записувати в нотації зворотного DNS-імені - наприклад, застосунок YouTube використовує ім'я пакета com.google.android.youtube. Часто ім'я пакета збігається з простором імен, що використовується в його Java-коді, але Android цього не вимагає (до того ж, APK-файли застосунків зазвичай включають і сторонні бібліотеки, простір імен яких, природно, не має взагалі нічого спільного з іменами пакетів, які їх використовують).

Кожен APK при складанні повинен бути підписаний розробником з використанням цифрового підпису. Android перевіряє наявність цього підпису при установці застосунка, а при оновленні вже встановленої програми застосунково порівнює публічні ключі, якими підписані стара і нова версія; вони повинні збігатися, що гарантує, що нова версія була створена тим же розробником, що і стара. (Якби цієї перевірки не було, зловмисник міг би створити пакет з таким же ім'ям, як і у наявної програми, переконати користувача встановити його, «оновлюючи» застосунок, і отримати доступ до даних цього застосунка.)

Саме оновлення пакета є установкою його нової версії замість старої зі збереженням даних і отриманих від користувача дозволів. Можна і «відкочувати» (downgrade) застосунки до більш старих версій, але при цьому за замовчуванням Android стирає збережені новою версією дані, оскільки стара версія може бути не здатна працювати з форматами даних, які використовує нову версію.

Як я вже говорив, зазвичай код кожної програми виконується під власним Unix-користувачем (UID), що і забезпечує їх взаємну ізоляцію. Кілька застосунків можуть явно попросити Android використовувати для них

загальний UID, що дозволить їм отримувати прямий доступ до файлів один одного і навіть, при бажанні, запускатися в одному процесі.

Хоча зазвичай одному пакету відповідає один файл APK, Android підтримує пакети, що складаються з декількох APK (це називається роздільними APK, або split APK). Це лежить в основі таких «магічних» можливостей Android, як динамічне завантаження модулів програми (dynamic feature modules) і Instant Run в Android Studio (автоматичне оновлення коду запущеного застосунку без його повного переустановлення і, в багатьох випадках, навіть без перезапуску).

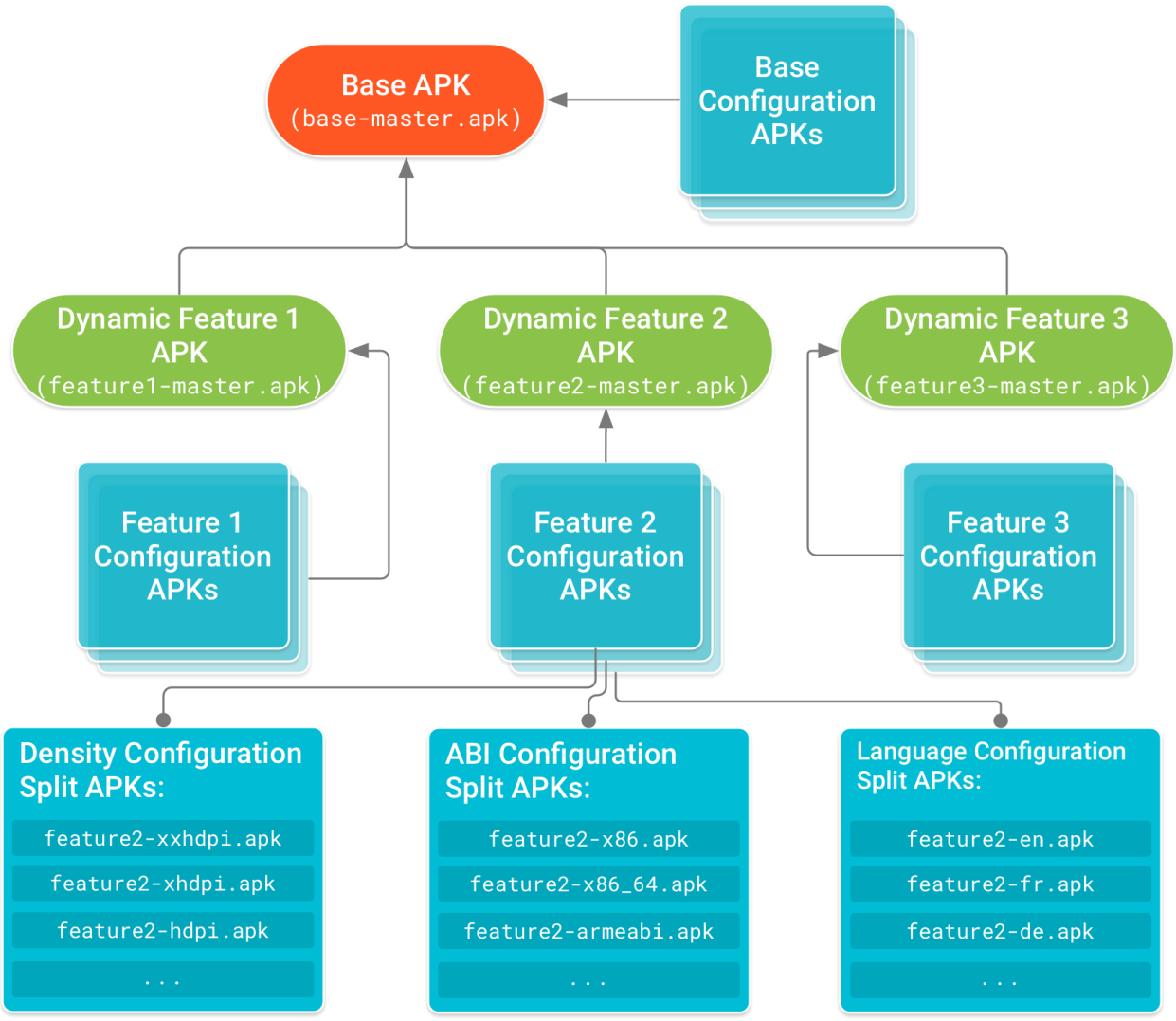

Рисунок 1.2 – Будова apk-застосунку

1.14 Файлова система

Пристрій файлової системи - один з найважливіших і найцікавіших питань в архітектурі операційної системи, і пристрій файлової системи в Android - не виняток.

Викликає зацікавлення, по-перше, те, які файлові системи використовуються, тобто в якому саме форматі вміст файлів зберігається на умовний диск (в разі Android це зазвичай flash-пам'ять і SD-карти) і як забезпечується підтримка цього формату з боку ядра системи . Ядро Linux, що використовується в Android, в тій чи іншій мірі підтримує велику кількість самих різних файлових систем - від використовуваних в Windows FAT і NTFS і використовуваних в Darwin сумнозвісної HFS + і сучасної APFS - до мережевої 9pfs. Є й багато «рідних» для Linux файлових систем - наприклад, Btrfs і сімейство ext.

Стандартом де-факто для Linux вже довгий час є ext4, використовувана за замовчуванням більшістю популярних дистрибутивів Linux. Тому немає нічого несподіваного в тому, що саме вона і використовується в Android. У деяких збірках (і деякими ентузіастами) також використовується F2FS (Flash-Friendly File System), оптимізована спеціально для flash-пам'яті (втім, з її перевагами все не так однозначно).

По-друге, інтерес представляє так званий filesystem layout - розташування системних і призначених для користувача папок і файлів в файлову систему. Filesystem layout в «звичайному Linux» заслуговує більш докладного опису; Найбільш важливі директорії:

/home зберігає домашні папки користувачів; тут же, в різних прихованих папках (.var, .cache, .config і інших), програми зберігають свої налаштування, дані та кеш, специфічні для користувача,

/boot зберігає ядро Linux і образ initramfs (спеціальної завантажувальної файлової системи),

/usr зберігає основну частину власне системи, в тому числі бібліотеки, виконувані файли, конфігураційні файли, а також ресурси - теми для інтерфейсу, значки, вміст системного мануала,

/etc зберігає загальносистемні налаштування,

/dev зберігає файли пристроїв та інші спеціальні файли (наприклад, сокет /dev/log),

/var зберігає змінювані дані - логи, системний кеш, вміст баз даних і т.п.

Android використовує схожий, але помітно відрізняючийся filesystem layout. Ось кілька найважливіших з його частин:

/data зберігає змінювані дані, ядро і образ initramfs зберігаються на окремому розділі (partition) flash-пам'яті, яка не монтується в основну файлову систему,

/system відповідає */usr* і зберігає систему,

/vendor - аналог */system*, призначений для файлів, специфічних для цієї збірки Android, а не входять в «стандартний» Android,

/dev, як і в «звичайному Linux», зберігає файли пристроїв та інші спеціальні файли.

Найцікавіші з цих директорій - */data* і */system*. Вміст */system* описує систему і містить більшість файлів з яких складається система. */system* розташовується на окремому розділі flash-пам'яті, який за замовчуванням монтується в режимі read-only; зазвичай дані на ньому змінюються тільки при оновленні системи. */data* також розташовується на окремому розділі і описує змінюваний стан конкретного пристрою, в тому числі призначені для користувача настройки, встановлені застосунки і їх дані, кеші і т.п. Очищення всіх призначених для користувача даних, так званий factory reset, при такій схемі полягає просто в очищенні вмісту розділу *data*; чиста система залишається встановлена в розділі *system*.

```
# mount | grep /system
/dev/block/mmcblk0p14 on /system type ext4 (ro,seclabel,relatime,data=ordered)
# mount | grep /data
/dev/block/mmcblk0p24 on /data type ext4 (rw,seclabel,nosuid,nodev,noatime,noauto_da_alloc,data=writeback)
```

Застосунки - а саме, їх APK, файли odex (скомпільований ahead-of-time Java-код) і ELF-бібліотеки - встановлюються в */system/app* (для застосунків, що поставляються з системою) або в */data/app* (для встановлених користувачем застосунків). Кожному предустановленному застосунком при створенні збірки Android виділяється папка з ім'ям виду */system/app/terminal*, а для встановлюваних користувачем застосунків при установці створюються папки, імена яких починаються з назви пакунка. Наприклад, застосунок YouTube зберігається в папку з назвою на кшталт /data/app/com.google.android.youtube-bkJAtUtbTuzvAioW-LEStg==/.

Оскільки встановлені застосунки зберігаються в розділі *system* (вміст якого, нагадаю, змінюється тільки при оновленні системи), їх неможливо видалити (замість цього Android надає можливість їх «відключити»). Проте, підтримується оновлення встановлених застосунків – при цьому для нової версії створюється папка в */data/app*, а поставлена з системою версія залишається в */system/app*. В цьому випадку у користувача з'являється можливість «видалити оновлення» такого застосунка, повернувшись до версії з */system/app*.

Інша особливість встановлених застосунків - вони можуть отримувати спеціальні «системні» дозволи. Наприклад, стороннім застосункам не одержати дозволи DELETE_PACKAGES, що дозволяє видаляти інші застосунки, REBOOT, що дозволяє запустити систему, і READ_FRAME_BUFFER, що дозволяє отримати прямий доступ до вмісту екрана. Ці дозволи мають рівень захисту signature, тобто застосунок, що намагається отримати доступ до них, має бути підписана тим же ключем, що і застосунок або сервіс, в якому вони реалізовані - в цьому випадку, оскільки ці дозволи реалізовані системою, ключем, яким підписана сама збірка Android.

Для зберігання змінюваних даних кожному з застосунком виділяється папка в */data/data* (наприклад, /data/data/com.google.android.youtube для YouTube). Доступ до цієї папці є тільки у самого застосунка - тобто тільки у UID, під яким запускається цей застосунок (якщо застосунок використовує кілька UID, або кілька застосунків використовують загальний UID, все може бути складніше). У цій папці застосунка зберігають настройки, кеш (в підпапках shared_prefs і cache відповідно) і будь-які інші потрібні їм дані:

```
# ls /data/data/com.google.android.youtube/
cache  code_cache  databases  files  lib  no_backup  shared_prefs
# cat /data/data/com.google.android.youtube/shared_prefs/youtube.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="user_account">username@gmail.com</string>
    <boolean name="h264_main_profile_supported7.1.2" value="true" />
    <int name="last_manual_video_quality_selection_max" value="-2" />
    <...>
</map>
```

Система знає про існування папки cache і може очищати її самостійно при нестачі місця. При видаленні застосунка вся папка цього застосунка повністю видаляється, і застосунок не залишає за собою слідів. Альтернативно, і те, і інше користувач може явно зробити в налаштуваннях:

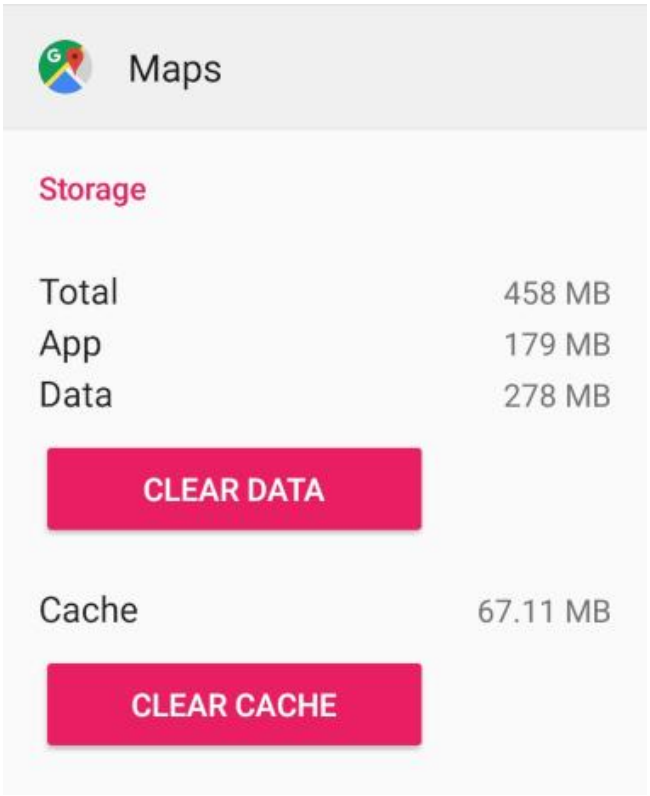

Рисунок 1.3 – Управління сховищем застосунка

Виділене кожному з застосунків сховище даних називається внутрішнім сховищем (internal storage).

Крім того, в Android є й інший тип сховища - так зване зовнішнє сховище (external storage - це назва відображає початкову задумку про те, що зовнішнє сховище повинне було розташовуватися на зовнішній SD-карті що встановлюється в телефон). По суті, зовнішнє сховище грає роль домашньої папки користувача - саме там розташовуються такі папки, як Documents, Download, Music і Pictures, саме зовнішнє сховище відкривають файлові менеджери в якості папки за замовчуванням, саме до вмісту зовнішнього сховища Android дозволяє отримати доступ з комп'ютера при підключенні по кабелю.

На відміну від внутрішнього сховища, розділеного на папки окремих застосунків, зовнішнє сховище являє собою «загальну зону»: до нього є повний доступ у будь-якої програми, яка отримала відповідний дозвіл від користувача. Цей дозвіл варто запитувати таким застосункам, як файловий менеджер; а більшості інших застосунків краще використовувати intent з дією

ACTION_GET_CONTENT, надаючи користувачу можливість самому вибрати потрібний файл в системному файловому менеджері.

Багато застосунків воліють зберігати і деякі зі своїх внутрішніх файлів, що мають великий розмір (наприклад, кеш завантажених зображень і аудіофайлів) в зовнішньому сховищі. Для цього Android виділяє застосунків в зовнішньому сховищі папки з назвами виду *android/data/com.google.android.youtube*. Самому застосунку для доступу до такої папки не потрібен дозвіл на доступ до всього зовнішнього сховища (оскільки в якості власника цієї папки встановлюється його UID), але до цієї папки може отримати доступ будь-який інший застосунок, що має такий дозвіл, тому її, дійсно, варто використовувати тільки для зберігання публічних і некритичних даних. При видаленні застосунка система видалить і його спеціальну папку в зовнішньому сховищі; але файли, створені застосунками в зовнішньому сховищі поза спеціальної папки вважаються належними користувачеві і залишаються на місці після видалення застосунку.

Початково передбачалося, що зовнішнє сховище дійсно буде розташовуватися на зовнішній SD-карті, оскільки в той час обсяг SD-карт значно перевищував обсяг вбудованої в тілі.

Традиційний підхід до безпеки комп'ютерних систем обмежується тим, щоб захищати систему від програмних атак. Вважається, що якщо у зловмисника є фізичний доступ до комп'ютера, гра вже програна: він може отримати повний доступ до будь-яких даних що зберігаються на ньому. Для цього йому досить, наприклад, запустити на цьому комп'ютері довільну контрольовану ним операційну систему, що дозволяє йому обійти будь-які обмеження прав що накладаються основною системою, або безпосередньо підключити диск з даними з іншим пристроєм. При бажанні, зловмисник може залишити комп'ютер в працездатному стані, але пропатчити встановлену на ньому систему, встановити довільні бекдори, кейлоггери і т.д.

Саме на захист від програмних атак орієнтована модель обмеження прав користувачів в Unix (і заснована на ній технологія app sandbox в Android); саме

по собі обмеження прав в Unix ніяк не захищає систему від користувача, який прокрався в серверну і отримав фізичний доступ до комп'ютера. І якщо серйозні, розраховані на багато користувачів сервера можна і потрібно охороняти від неавторизованого фізичного доступу, до персональних комп'ютерів - а тим більше мобільних пристроїв - такий підхід просто непридатний.

Намагатися покращувати ситуацію з захистом від зловмисника, який отримав фізичний доступ до пристрою, можна за двома напрямками:

По-перше, можна шифрувати дані що зберігаються на диску, тим самим не дозволяючи зловмисникові отримати доступ до самих даних, навіть якщо у нього є доступ до вмісту диска.

По-друге, можна так чи інакше обмежувати можливість завантаження на пристрої довільних операційних систем, змушуючи зловмисника проходити процедури аутентифікації і авторизації в встановленій системі.

Саме з цими двома напрямками захисту пов'язана модель безпечного завантаження в Android.

Верифікація завантажувача

Процес завантаження Android побудований так, що він, з одного боку, не дозволяє зловмисникам завантажувати на пристрої довільну ОС, з іншого боку, може дозволяти користувачам встановлювати кастомізовані збірки Android (і інші системи).

Перш за все, Android-пристрої, на відміну від «десктопних» комп'ютерів, зазвичай не дозволяють користувачеві (або зловмисникові) зробити завантаження з зовнішнього носія; замість цього відразу запускається

встановлений на пристрої bootloader (завантажувач). Bootloader - це відносно проста програма, в завдання якої (при завантаженні в звичайному режимі) входять:

ініціалізація і налаштування Trusted Execution Environment (наприклад, ARM TrustZone), знаходження розділів вбудованої пам'яті, в яких зберігаються образи ядра Linux і initramfs, перевірка їх цілісності і недоторканності (integrity) - в іншому випадку завантаження переривається з повідомленням про помилку - шляхом верифікації цифрового підпису виробника, завантаження ядра і initramfs в пам'ять і передача управління ядру.

Крім того, bootloader підтримує функціональність для поновлення і перевстановлення системи.

По-перше, це можливість завантажити замість основної системи (Android) спеціальну мінімальну систему, звану recovery. Версія recovery, що встановлюється на більшість Android-пристроїв за замовчуванням, дуже мінімалістично і підтримує тільки установку оновлень системи в автоматичному режимі, але багато ентузіастів Android встановлюють кастомную recovery.

Це робиться шляхом використання іншої «фічі» bootloader'a, спрямованої на оновлення та переустановлення системи - підтримки перезапису (flashing) вмісту і структури розділів по командам з приєднаного по кабелю комп'ютера. Для цього bootloader здатний завантажуватися в ще один спеціальний режим, який називають fastboot mode (або іноді просто bootloader mode), оскільки зазвичай для спілкування між комп'ютером і bootloader'ом в цьому режимі використовується протокол fastboot (і відповідний йому інструмент fastboot з Android SDK з боку комп'ютера).

Деякі реалізації bootloader'a використовують інші протоколи. В основному це стосується пристроїв, що випускаються компанією Samsung, де спеціальна реалізація bootloader'a (Loke) спілкується з комп'ютером за власним пропрієтарного протоколу (Odin). Для роботи з Odin з боку комп'ютера можна

використовувати або реалізацію від самих Samsung (яка теж називається Odin), або вільну реалізацію під назвою Heimdall.

Конкретні деталі залежать від реалізації bootloader'a (тобто розрізняються залежно від виробника пристрою), але в багатьох випадках установка recovery і збірок Android, підписаних ключем виробника пристрою, просто працює без застосункових складнощів:

```
$ fastboot flash recovery recovery.img
$ fastboot flash boot boot.img
$ fastboot flash system system.img
```

Таким чином можна вручну оновлювати систему; а ось встановити старішу версію не вийде: функція, відома як захист від відкату, не дозволить bootloader'у завантажити старішу версію Android, ніж була завантажена в минулий раз, навіть якщо вона підписана ключем виробника, оскільки завантаження старих версій відкриває дорогу до використання опублікованих вразливостей, які виправлені в більш нових версіях.

Крім того, багато пристроїв підтримують розблокування bootloader'a (unlocking the bootloader, також відому як OEM unlock) - відключення перевірки bootloader'ом підписи системи і recovery, що дозволяє встановлювати довільні збірки того і іншого (у частині виробників це анулює гарантію). Саме так зазвичай встановлюються такі популярні дистрибутиви Android, як LineageOS (колишній CyanogenMod), Paranoid Android, AOKP, OmniROM і інші.

Оскільки розблокування bootloader'a все-таки дозволяє завантажити на пристрої власну версію системи, з метою безпеки при розблокуванні всі призначені для користувача дані (з розділу data) примусово видаляються. Якщо систему переустановлює сам користувач, а не зловмисник, після переустановлення він може відновити свої данні з резервної копії (наприклад, з хмарного бекапа на серверах Google або з резервної копії на зовнішньому носії), якщо зловмисник - він отримає працюючу систему, але не зможе вкрасти дані власника пристрою.

Після установки бажаних збірок recovery і системи bootloader варто заблокувати назад, щоб знову захистити свої дані в разі потрапляння пристрою в руки зловмисників.

Для розблокування bootloader'a може також знадобитися дозволити її з налаштувань системи:

Популярна стороння recovery, яку встановлюють більшість - TWRP (Team Win Recovery Project). Вона містить тач-інтерфейс і безліч просунутих «фіч», в тому числі можливість встановлювати частини системи з збірок у вигляді zip-архівів, вбудовану підтримку резервних копій і навіть повноцінний емулятор терміналу з віртуальною клавіатурою:

1.15 Шифрування диска

Сучасні версії Android використовують пофайлове шифрування даних (file-based encryption). Цей механізм заснований на вбудованої в ext4 підтримки шифрування, реалізованої в ядрі Linux (fscrypt), і дозволяє системі зашифровувати різні частини файлової системи різними ключами.

За замовчуванням система шифрує більшість даних користувача, розташованих на розділі data, за допомогою ключа, який створюється на основі пароля користувача і не зберігається на диск (credential encrypted storage). Це означає, що при завантаженні система повинна попросити користувача ввести свій пароль, щоб обчислити з його допомогою ключ для розшифровки даних.

Саме тому перший раз після включення пристрою користувача зустрічає вимогу ввести повний пароль або графічний ключ, а не просто пройти аутентифікацію, приклавши палець до сканера відбитків.

Разом з credential encrypted storage в Android також використовується device encrypted storage - шифрування ключем на основі даних, які зберігаються на пристрої (в тому числі в Trusted Execution Environment). Файли, зашифровані таким чином, система може розшифрувати до того, як користувач введе пароль. Це лежить в основі функції, відомої як Direct Boot: система здатна

завантажуватися в деякий працездатний стан і без введення пароля; при цьому застосунки можуть явно попросити систему зберегти (найменш приватну) частину своїх даних в device encrypted storage, що дозволяє їм починати виконувати свої базові функції, не чекаючи повної розблокування пристрою. Наприклад, Direct Boot дозволяє будильнику спрацьовувати і до першого введення пароля, що особливо корисно, якщо пристрій непередбачено перезавантажується вночі через тимчасове відключення живлення або збою системи.

1.16 Права суперкористувача

Так званий root-доступ - це можливість виконувати код від імені «користувача root» (UID 0, також відомого як привілейований користувач). Root - це спеціально виділений Unix-користувач, якому - за декількома винятками - дозволений повний доступ до всього в системі, і на якого не поширюються жодні обмеження прав.

Як і більшість інших сучасних операційних систем, Android спроектований з розрахунком на те, що звичайному користувачеві ні для чого не потрібно використовувати root-доступ. На відміну від більш закритих операційних систем, користувачі яких називають руйнування накладених на них обмежень буквально «втечею з тюрми», в Android прямо «з коробки», без необхідності отримувати root-доступ і встановлювати спеціальні сторонні «твіки», є можливість:

встановлювати довільні програми - як з безлічі існуючих магазинів застосунків, так і довільні APK з будь-яких джерел, вибирати стандартні програми: веб-браузер, email-клієнт, камеру, файловий менеджер, лончер, застосунок для дзвінків, застосунок для СМС, застосунок для контактів і так далі, встановлювати і використовувати пакети значків (icon packs), отримувати прямий доступ до файлової системи, зберігати в ній довільні файли, підключати пристрій до комп'ютера (або навіть до іншого Android-

пристрою) і безпосередньо передавати між ними файли по кабелю, і навіть, у багатьох дистрибутивах Android, настроювати кольори і шрифти системної теми.

Отже, для звичайних завдань, з якими може зіткнутися простий користувач, root-доступ не потрібен. У той же час використання root-доступу неминуче пов'язане з багатьма проблемами з безпекою (докладніше про них нижче), тому в більшості випадків Android не дозволяє користувачеві працювати від імені root. Застосунки, в тому числі емулятори терміналів, виконуються від імені своїх обмежених Unix-користувачів; а shell, який запускається при використанні команди adb shell, працює від імені спеціально для цього призначеного Unix-користувача shell.

Проте, буває, що root доступний користувачеві:

По-перше, root-доступ зазвичай дозволений на збірках Android для емуляторів - поставляються разом з Android Studio віртуальних машин на базі QEMU, які зображують реальні пристрої і зазвичай використовуються розробниками для налагодження і тестування програм.

По-друге, root-доступ включений за замовчуванням у багатьох сторонніх дистрибутивах Android (pre-rooted ROMs).

По-третє, при розблокованому bootloader'i root-доступ на будь-який збірці Android можна включити безпосередньо, просто встановивши виконуваний файл, який реалізує команду su, через bootloader.

Ну і нарешті, по-четверте, на старих версіях системи, що містять відомі уразливості, часто можна отримати root-доступ, проексплуатувати їх (зазвичай для отримання root-доступу експлуатуються відразу декілька вразливостей в різних шарах і компонентах системи). Наприклад, якщо система не оновлювалася з першої половини 2016 року, для отримання root-доступу можна скористатися вразливістю, що одержала широку популярність Dirty Cow.

Навіщо це може бути потрібно? Звичайно, root-доступ корисний для налагодження і дослідження роботи системи. Крім того, володіючи root-доступом, можна необмежено налаштовувати систему, змінюючи її теми,

поведінку і багато інших аспектів. Можна примусово підміняти код і ресурси застосунків - наприклад, можна видалити з програми рекламу або розблокувати його платну або приховану функціональність. Можна встановлювати, змінювати і видаляти довільні файли, в тому числі в розділі system (хоча це майже напевно погана ідея).

З більш філософської точки зору, root-доступ дозволяє користувачеві повністю контролювати свій пристрій і свою систему - замість того, щоб пристрій і розробники ПЗ контролювали користувача.

З великими можливостями приходить і велика відповідальність, і зовсім не завжди цю відповідальність можна довірити користувачеві і застосунків. Іншими словами, з root-доступом пов'язана велика кількість проблем в області безпеки і стабільності системи.

Нагадаю, що користувач майже завжди взаємодіє з пристроєм не безпосередньо, а через застосунки. А це означає, що і root-доступ він буде використовувати в основному через застосунки, які, можна сподіватися, будуть сумлінно користуватися root-доступом для добрих намірів. Але якщо на Вашому пристрої наявний root, це, в принципі, означає, що застосунки можуть скористатися ним і для нехороших цілей - нашкодити системі, вкрасти цінні дані, заразити систему вірусом, встановити кейлоггер і т.д.

На відміну від традиційної моделі довіри програмами в класичному Unix, Android розрахований на те, що користувач не може довіряти стороннім застосункам - тому їх і поміщають в пісочницю. Тим більше не можна довіряти застосункам root-доступ, сподіваючись, що вони будуть використовувати його тільки на благо. Фактично, root-доступ руйнує акуратно вибудувану модель безпеки Android, знімаючи з застосунків всі обмеження і відкриваючи їм права на доступ до всього в системі.

З іншого боку, і застосунки не можуть довіряти пристрою, на якому підключений root-доступ, оскільки на такому пристрої в його роботу мають можливість непередбачуваними способами втручатися користувач і інші застосунки. Наприклад, розробники програми, що містить платну

функціональність, звичайно, не захочуть, щоб перевірку здійснення покупки можна було відключити. Багато застосунків, що працюють з особливо цінними даними - наприклад, Google Pay - явно відмовляються працювати на пристроях з root-доступом, вважаючи їх недостатньо безпечними.

Аналогічно, онлайн-ігри на зразок популярної Pokémon GO теж відмовляються працювати на root-ованних пристроях, побоюючись, що гравець зможе з легкістю порушити правила гри.

Google дозволяє виробникам встановлювати Google Play Store і Google Play Services тільки на пристрої, де root-доступ відключений, тим самим роблячи Android більш привабливою платформою для розробників, які, природно, воліють вкладати ресурси в розробку під платформи, що не дозволяють користувачам з легкістю втручатися в роботу їх застосунків. А оскільки Play Store - найбільш відомий і популярний (як серед розробників застосунків, так і серед користувачів) магазин застосунків для Android, більшість виробників вважають за краще встановлювати його на свої пристрої. (Є й винятки - наприклад, Amazon використовує власний дистрибутив Android під назвою Fire OS для своїх пристроїв - Echo, Fire TV, Fire Phone, Kindle Fire Tablet - і не встановлювати ніяких застосунків від Google). Саме тому root-доступ за замовчуванням відключений на більшості популярних Android-пристроїв.

Незважаючи на обмеження для виробників, Google дозволяє користувачам збірок, в яких дозволений root-доступ, самостійно встановлювати Google Play (зазвичай це робиться шляхом встановлення готових пакетів від проекту Open GApps); при цьому Google вимагає, щоб після установки користувач вручну зареєстрував пристрій на спеціально призначеної для цього сторінці реєстрації.

У «звичайному» Linux, як і в інших Unix-системах, для отримання root-доступу (за допомогою команд sudo і su) користувачеві потрібно ввести пароль (або свій, або пароль Unix-користувача root, в залежності від настройки системи і конкретної команди) або авторизуватись іншим способом (наприклад, за

допомогою відбитка пальця). На застосунок до власне авторизації це служить підтвердженням того, що користувач довіряє цій програмі і згоден надати їй можливість скористатися root-доступом.

Стандартна версія команди su з Android Open Source Project не запитує підтвердження користувача явно, але вона доступна тільки Unix-користувачеві shell (і самому root), що повністю відрізає застосунків можливість отримувати root-права. Багато сторонніх реалізації su дозволяють будь-якому застосунку отримувати права суперкористувача, що, як писалося вище, дуже погано в плані безпеки.

Як компроміс багатьма користувачами використовуються спеціальні програми, які керують доступом до su. При спробі програми викликати su вони запитують підтвердження у користувача, який може дозволити або заборонити застосунком отримати root-доступ. Кілька років тому була популярна одна з таких програм, SuperSU; останнім часом її витіснив новий відкритий проект під назвою Magisk.

Отримання прав суперкористувача в системі

Основна особливість Magisk - можливість проведення широкого класу модифікацій системи, в тому числі пов'язаних зі зміною файлів, розташованих в папці */system*, насправді не змінюючи сам розділ system (це називають словом systemless-ly), шляхом використання декількох просунутих «фіч» Linux . У поєднанні з Magisk Hide - можливістю не просто не давати деяким застосункам root-доступ, а повністю ховати від них сам факт наявності root-доступу і установки Magisk в системі - це дозволяє пристрою як і раніше отримувати оновлення системи від виробника і використовувати програми на кшталт того ж Google Pay, які відмовляються працювати на root-ованних системах. Magisk Hide здатний обходити навіть такі досить просунуті технології виявлення root-а, як SafetyNet від Google.

Потрібно усвідомлювати, що можливість використовувати програми на кшталт Google Pay на root-ованних пристроях незважаючи на вбудовані в них перевірки - це не вирішення пов'язаних з root-доступом проблем з безпекою. Застосунки, яким користувач довірив root-доступ, як і раніше можуть вирішити втрутитися в роботу системи і вкрасти гроші користувача через Google Pay. Проблеми з безпекою залишаються, нам лише вдається закрити на них очі.

Magisk підтримує systemless установку готових системних «твіків» у вигляді Magisk Modules, спеціальних пакетів для установки з використанням Magisk. У такому вигляді доступні, наприклад, відомий root-фреймворк Xposed і ViPER, набір просунутих драйверів і налаштувань для відтворення звуку.

1.17 Драйвери і Фрагментація

Системи на кристалі

Апаратне забезпечення традиційних настільних і серверних комп'ютерів побудовано навколо материнської плати, на яку встановлюються такі найважливіші компоненти заліза, як центральний процесор і оперативна пам'ять, і до якої потім підключаються застосункові плати - наприклад, графічна і мережева - містять інші компоненти (відповідно, графічний процесор і Wi-Fi адаптер).

На відміну від такої схеми, в більшості Android-пристроїв використовуються так звані системи на кристалі (system on a chip, SoC). Система на кристалі являє собою набір компонентів комп'ютера - центральний процесор, блок оперативної пам'яті, порти введення-виведення, графічний процесор, LTE-, Bluetooth- і Wi-Fi-модеми – повністю реалізованих і інтегрованих в рамках одного мікрочіпа. Такий підхід дозволяє не тільки зменшити фізичний розмір пристрою, щоб він помістився в кишені, і підвищити його продуктивність за рахунок більшої локальності і кращої інтеграції між компонентами, а й значно знизити його

енергоспоживання і тепловиділення, що особливо актуально для мобільних пристроїв, що живляться від вбудованої батарейки і не мають систем активного охолодження.

Але, звичайно, системи на кристалі мають і свої недоліки. Найбільш очевидний недолік полягає в тому, що таку систему не можна «проапгрейдити», докупивши, наприклад, додаткової оперативної пам'яті, як не можна і замінити погано працюючий компонент. Це, в принципі, вигідно компаніям-виробникам, оскільки спонукає людей купувати нові пристрої, коли існуючі морально застарівають або виходять з ладу, замість того, щоб їх точково оновлювати або ремонтувати.

Взаємодія з апаратними компонентами

Інший - набагато більш важливий саме в контексті Android - недолік SoC полягає в пов'язаних з ними складнощі з драйверами. Як і в традиційних системах, заснованих на окремих платах, кожен компонент системи на кристалі - від камери до LTE-модему - вимагає для роботи використання спеціальних драйверів. Але на відміну від традиційних систем, ці драйвера зазвичай розробляються виробниками систем на кристалі і специфічні для їх конкретних моделей; крім того, вихідні коди таких драйверів зазвичай не розкриваються, та й бінарні збірки в вільний доступ виробниками викладаються зовсім не завжди.

Замість цього виробник SoC (наприклад, Qualcomm) віддає готову збірку драйверів виробникам Android-пристроїв (наприклад, Sony або LG), які включають її в свою збірку Android, засновану на коді з Android Open Source Project. Так і виходить, що збірка Android, попередньо встановлена на пристрої виробником, містить всі необхідні для цього пристрою драйвера, а безпосередньо використовувати збірку для одного пристрою на іншому неможливо.

В результаті авторам збірок Android доводиться докладати окремі зусилля для підтримки кожного сімейства моделей Android-пристроїв. Самі виробники пристроїв зовсім не завжди зацікавлені в тому, щоб виділяти ресурси на портирование своїх збірок на нові версії Android. У той же час розробникам сторонніх дистрибутивів Android доводиться витягати драйвера з збірок, попередньо встановлювати виробниками, що пов'язано з застосунковими складнощами і не завжди призводить до хорошого результату. У багатьох сторонніх дистрибутивів вистачає ресурсів тільки для підтримки найбільш популярних моделей пристроїв.

Фрагментація девайсів на основі ОС Android

Це призводить до проблеми, відомої як фрагментація: екосистема Android складається з великої кількості різних збірок - як офіційних збірок від виробників пристроїв, так і версій сторонніх дистрибутивів. Багато з них до того ж засновані на старих версіях Android, оскільки для багатьох пристроїв поновлення збірок від виробника виходять повільно або не виходять зовсім.

Звичайно, повільне оновлення екосистеми означає, що не всі користувачі отримують невеликі оновлення інтерфейсу та інші видимі користувачеві поліпшення, які приносять нові версії Android. Але воно приносить і дві набагато більш серйозні проблеми.

По-перше, страждає безпека. Хоча сучасні системи використовують просунуті механізми захисту, в них постійно виявляються нові вразливості. Зазвичай ці вразливості оперативно виправляються розробниками, але це не допомагає користувачам, якщо оновлення системи, що містять виправлення, до них ніколи не доходять.

По-друге, фрагментація негативно позначається на розробників застосунків під Android - страждає так званий developer experience (DX, за аналогією з user experience/UX). У теорії, незважаючи на внутрішні відмінності в драйверах і

призначеному для користувача інтерфейсі різних версій і збірок Android, використовувані розробниками застосунків API - Android Framework, OpenGL/Vulkan і інші - повинні бути переносимі і працювати однаково. На практиці це, звичайно, не завжди так, і розробникам доводиться тестувати і забезпечувати роботу своїх застосунків на безлічі версій і збірок Android - на різних пристроях, різних версіях системи, сторонніх дистрибутивах і так далі.

Далеко не всі виробники Android-пристроїв не вважають важливим своєчасно випускати оновлення для своїх пристроїв. Наприклад, Google випускають оновлення для своїх лінійок Nexus і Pixel одночасно з тим, як виходить нова версія Android. У багатьох інших виробників портування збірок Android на нову версію займає місяці, але вони, тим не менш, намагаються випускати щомісячні оновлення безпеки.

Крім того, зовсім не всі пристрої, де використовується Android, побудовані на основі SoC. Android цілком можна встановити і на звичайний «десктопний» комп'ютер (підійдуть, наприклад, збірки від проектів Android-x86 і RemixOS). Спеціальна збірка Android вбудована в ChromeOS, що дозволяє Chromebook'ам запускати Android-застосунки поряд з Linux-застосунками і веб-застосунками. Аналогічного підходу - запуску спеціальної збірки Android в контейнері - дотримується проект Anbox, що дозволяє використовувати Android-застосунки на «звичайних» Linux-системах.

І тим не менше, погана переносимість збірок Android між різними пристроями і пов'язана з нею фрагментація екосистеми - це серйозна проблема, яка затрудняє розвиток і просування Android як платформи.

Найбільш прямий спосіб боротися з фрагментацією - це всіляко переконувати виробників пристроїв не закидати підтримку своїх збірок Android. Як показує практика, це працює, але працює недостатньо добре. Було б набагато краще, якщо можна було б розробити технічне рішення, що підвищує переносимість збірок Android і тим самим серйозно полегшує завдання авторів збірок і сторонніх дистрибутивів.

І таке рішення вже існує. У 2017 році Google анонсували Project Treble - нову, ще більш модульну (в порівнянні з вже існуючим HAL, hardware abstraction layer) архітектуру взаємодії драйверів (і решти софта, специфічного для конкретного пристрою) з іншими частинами системи. Treble дозволяє встановлювати основну систему і драйвера, специфічні для пристрою, на різні розділи файлової системи, і оновлювати - або як завгодно змінювати - систему окремо від встановлених драйверів.

Treble в корені міняє ситуацію з повільним випуском оновлень і поганою переносимістю збірок. Завдяки Treble пристрою від семи різних компаній (Sony, Nokia, OnePlus, Oppo, Xiaomi, Essential і Vivo - а не тільки від самих Google) змогли брати участь в бета-програмі Android Pie. Treble дозволив Essential випустити оновлення до Android Pie для свого Essential Phone прямо в день виходу Android Pie. Одну і ту ж складання Android - один і той же бінарний файл без перекомпіляції або будь-яких змін - тепер можна запускати на будь-якому пристрої, що підтримує Treble, незважаючи на те, що вони можуть бути засновані на абсолютно різних SoC.

Вплив Treble дійсно складно переоцінити. Java принесла можливість «write once, run everywhere» для високорівневого коду - в тому числі і можливість запускати Android-застосунки на комп'ютерах з практично будь-якою архітектурою процесора. Treble - аналогічний прорив, що дозволяє використовувати один раз написану і скомпільовану збірку Android на пристроях з абсолютно різними SoC. Тепер справа за виробниками, яким потрібно конвертувати свої драйвера в формат, сумісний з Treble. Можна сподіватися, що через кілька років проблеми з оновленнями Android-пристроїв зникнуть остаточно.

Висновки до розділу 1

Операційна система Android – прямий нащадок ОС Linux. І як нащадок ввібрала в себе багато механізмів безпеки від цієї ОС. Так як ОС розроблялася,

як система реального часу, багато механізмів, разом і з тим і механізмів безпеки, змінилося і з’явилися нові, що змінює підходи до тестування на проникнення їх застосунків. І хоча компанія Google доповнює і ускладнює ці механізми, при розробці великих проектів залишаються архітектурні прогалини, що дозволяє компрометацію цих застосунків.

Було виділено основні відмінності в ОС, що вплинули на безпеку застосунків:

1. Створення ізольованих пісочниць та розвинутої системи міжпроцесної взаємодії призвело до дроблення програм на сервіси. Це збільшило поверхню атак на застосунки.
2. Використання мови програмування JAVA та формат виконуваних файлів заснований на її байткоді що виконується емуляторами Dalvik/ART, зробило процес реверсу застосунків доволі тривіальним, вихідний код після декомпіляції майже ідентичний вихідному. Варто зазначити що новітні інструменти обфускації майже знімають це питання, та вони зазвичай задорогі.
3. Розробка механізму зберігання ключової інформації keystore сильно ускладнила криптографічну компрометацію.
4. Відсутність ролі супер користувача, знизила небезпеку підвищення привілеїв.

2 ТЕСТУВАННЯ НА ПРОНИКНЕННЯ ТА ЗВОРОТНА РОЗРОБКА ЗАСТОСУНКІВ

2.1 Архітектура Apk застосунку

Застосунок для Android упакований як файл APK (пакет Android), це ZIP-файл, що містить в собі код, ресурси, підпис, маніфест та всі інші файли, необхідні для запуску програмного забезпечення. Як і будь-якого ZIP-архіву ми можемо вивчити його вміст використавши утиліту командного рядку unzip:

```
unzip application.apk -d application
```

Вміст apk файлу:

```
/AndroidManifest.xml (файл)
```

Це бінарне представлення файлу XML маніфесту, який декларує які дозволи буде мати застосунок (деякі дозволи можуть запитуватися під час роботи застосунку, і не описуватися в цьому файлі). Також описуються Activity, Services, Resivers.

Після декомпіляції має вигляд:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.company.appname"
                                                                     platformBuildVersionCode="24"
                                                                     platformBuildVersionName="7.0">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true" android:theme="@style/AppTheme">
        <activity android:name="com.company.appname.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

На відміну від звичних настільних програм, які завжди мають точку входу(зазвичай це функція main()), Android-застосунок немає єдиної точки входу, але всі точки входу оголошені в цьому файлі.

```
/assets/* (папка)
```

Ця папка містить специфічні файли, наприклад mp3 або власні шрифти. Зазвичай для тесту на проникнення це не дуже корисна папка.

```
/res/* (папка)
```

Тут зберігаються всі ресурси, такі як XML-файли, зображення та спеціальні стилі.

```
/resources.arsc (файл)
```

В цьому файлі проіндексовані всі ресурси проекту, кожному файлу присвоєний числовий ідентифікатор.

```
/classes.dex (файл)
```

Цей файл містить Dalvik (віртуальна машина, що працює на Android) байт-код застосунку. Програма для Android (в більшості випадків) розроблена за допомогою мови програмування Java. Файли вихідних файлів Java потім складаються в байт-код(smali), який, врешті-решт, буде виконуватися Dalvik VM. Цей файл містить логіку, саме те, що нас цікавить.

Іноді також можна знайти файл classes2.dex, це пов'язано з форматом DEX, який має обмеження на кількість класів, які можна оголосити в одному файлі dex, застосунки Android ставали все більшими, і тому Google адаптував цей формат, підтримуючи додатковий файл .dex, де можна оголосити інші класи.

З нашої точки зору це не має значення, інструменти, які ми збираємося використати, зможуть його виявити і додати при декомпіляції.

```
/libs/* (папка)
```

Іноді застосунок повинен виконувати нативний код, це може бути бібліотека обробки зображень, ігровий движок чи інше. У такому випадку ці бібліотеки будуть знаходитися всередині папки libs, розділені на підпапки для архітектури (таким чином додаток буде працювати на ARM, ARM64, x86 тощо). Ці бібліотеки зазвичай розробляються на мовах програмування C/C++ і зв’язуються з основним кодом за допомогою JNI.

```
/META-INF/ (папка)
```

Кожна програма для Android повинна бути підписана з сертифікатом розробника для запуску на пристрої, навіть відлагоджувальна збірка підписується відлагоджувальним сертифікатом, у папці META-INF містяться відомості про файли всередині APK та про розробника. Всередині цієї папки ви зазвичай можете знайти:

- Файл MANIFEST.MF з SHA-1 або SHA-256 хешем усіх файлів всередині APK.
- Файл CERT.SF, що нагадує MANIFEST.MF, але підписаний з ключем RSA.
- Файл CERT.RSA, що містить відкритий ключ розробника, який використовується для підписання файлу CERT.SF та дайджестів.

Ці файли дуже важливі, для гарантування цілісності APK та право власності на код. Інколи перевіряти такий підпис може бути дуже зручним для того, щоб визначити, хто дійсно розробив даний APK. Якщо ви хочете отримати інформацію про розробника, ви можете скористатись утилітою командного рядка openssl:

```
openssl pkcs7 -in /path/to/extracted/apk/META-INF/CERT.RSA -inform DER -print
```

На виході отримуємо:

```
PKCS7:
  type: pkcs7-signedData (1.2.840.113549.1.7.2)
  d.sign:
    version: 1
    md_algs:
        algorithm: sha1 (1.3.14.3.2.26)
        parameter: NULL
    contents:
      type: pkcs7-data (1.2.840.113549.1.7.1)
      d.data: <ABSENT>
    cert:
        cert_info:
          version: 2
          serialNumber: 10394279457707717180
          signature:
            algorithm: sha1WithRSAEncryption (1.2.840.113549.1.1.5)
            parameter: NULL
          issuer: C=TW, ST=Taiwan, L=Taipei, O=ASUS, OU=PMD, CN=ASUS AMAX Key/emailAddress=admin@asus.com
          validity:
            notBefore: Jul  8 11:39:39 2013 GMT
            notAfter: Nov 23 11:39:39 2040 GMT
          subject: C=TW, ST=Taiwan, L=Taipei, O=ASUS, OU=PMD, CN=ASUS AMAX Key/emailAddress=admin@asus.com
          key:
            algor:
              algorithm: rsaEncryption (1.2.840.113549.1.1.1)
              parameter: NULL
            public_key:  (0 unused bits)
              ...
```

Таким чином ми можемо перевірити чи застосунок був дійсно підписаний розробником якому ми довіряємо, чи його модифікувала третя сторона.

2.2 Отримання Apk-застосунку

Існує два способи отримання застосунку: встановлення його на своєму пристрої та використання adb, для його отримання, або використання онлайнової служби для його завантаження.

2.3 Витяг Застосунку з Adb

Перш за все, підключимо наш смартфон до USB-порту нашого комп'ютера і отримаємо список встановлених пакетів та їх імен:

```
adb shell pm list packages
```

Це дозволить перерахувати всі пакети на смартфоні, коли область імен пакету буде знайдено, можна визначити його фізичний шлях:

```
adb shell pm path com.android.systemui
```

Отримуємо шлях до apk:

```
package:/system/priv-app/SystemUIGoogle/SystemUIGoogle.apk
```

Витягуємо його з пристрою:

```
adb pull /system/priv-app/SystemUIGoogle/SystemUIGoogle.apk
```

Ми отримали застосунок для подальшої зворотної розробки.

2.4 Використання онлайнової служби

Кілька онлайнових ресурсів доступні, якщо ви не хочете встановлювати додаток на своєму пристрої (наприклад, якщо ви досліджуєте шкідливе програмне забезпечення), ось список деяких з них:

- Apk-DL
- Evozi Downloader
- Apk Leecher

Після завантаження APK із цих служб рекомендується перевірити сертифікат розробника, як показано раніше, щоб переконатися, що ви завантажили правильний файл .apk, а не зловмисне програмне забезпечення.

2.5 Аналіз мережі

Тепер ми починаємо з деяких тестів, щоб зрозуміти, що додаток виконує під час виконання. Мій перший тест зазвичай полягає у перевірці мережевого трафіку, який генерується самою програмою, і для цього я використовую інструмент під назвою bettercap.

Переконайтеся, що ви встановили bettercap, і що ваш комп'ютер та пристрій Android перебувають у тій самій мережі Wi-Fi, тоді ви можете запустити MITM (192.168.1.5 у цьому прикладі) і переглянути його трафік у реальному часі з терміналу:

```
sudo bettercap -T 192.168.1.5 -X
```

Параметр -X активує сніффер, як тільки запускається застосунок, ви повинні побачити купу серверів HTTP та/або HTTPS, з якими ви зв'язуєтеся, тепер ми знаємо з ким спілкується застосунок. Перевіримо яку інформацію він відправляє:

```
sudo bettercap -T 192.168.1.5 --proxy --proxy-https --no-sslstrip
```

Ця команда переключає з пасивного режиму сніффу, в режим проксі-сервера. Весь трафік HTTP та HTTPS буде перехоплений за допомогою bettercap.

Якщо застосунок правильно використовує закріплення сертифікатів, ви не зможете побачити його HTTPS-трафік.

2.6 Статичний аналіз

Після мережевого аналізу ми зібрали велику кількість URL-адрес і пакетів, ми можемо використовувати цю інформацію як нашу точку входу, це буде те, що ми будемо шукати під час виконання статичного аналізу в додатку. "Статичний аналіз" означає, що ви не запускаєте застосунок, а просто вивчаєте його код. Найпопулярніші інструменти статичного аналізу:

2.7 Apktool

APKTool - інструмент, який здатний декомпілювати файл AndroidManifest у вихідному форматі XML - файлі resources.arsc, а також він конвертує файл classes.dex (та classes2.dex, якщо він присутній) у файл проміжної мови, що називається SMALI, ASM-подібна мова, що використовується для представлення операційних кодів Далвік.

Це виглядає як:

```
.super Ljava/lang/Object;
.method public static main([Ljava/lang/String;)V
    .registers 2
    sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
    const-string    v1, "Hello World!"
    invoke-virtual {v0, v1}, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
    return-void
.end method
```

Декомпіляція apk:

```
apktool d application.apk
```

Ви також можете використовувати apktool для декомпілювання APK, модифікувати його, а потім перекомпілювати, але якщо інші інструменти не зможуть декомпілювати, потрібно буде прочитати smali-код, щоб змінити програму.

2.8 jADX

Комплект jADX дозволяє просто завантажити APK та переглянути його вихідний код Java. jADX декомпілює APK до smali, а потім перетворює smali назад на Java. Зрозуміло, що читання коду Java набагато простіше, ніж читання smali.

Одним з кращих функцій JADX є пошук по рядку/символу, який дозволить вам шукати URL-адреси, рядки, методи та все, що ви хочете знайти в базі коду застосунка.

Крім того, є меню "*Find Usage*", просто виділіть деякий символ і клацніть правою кнопкою миші на ньому, ця функція дасть вам список всіх посилань на цей символ.

2.9 Dex2Jar і JD-Gui

Подібно до jADX є інструменти dex2jar та JD-GUI, після встановлення, ви будете використовувати dex2jar для перетворення APK у файл JAR:

```
/path/to/dex2jar/d2j-dex2jar.sh application.apk
```

Після того, як у вас є файл JAR, просто відкрийте його за допомогою JD-GUI, і ви побачите його код Java, як і jADX:

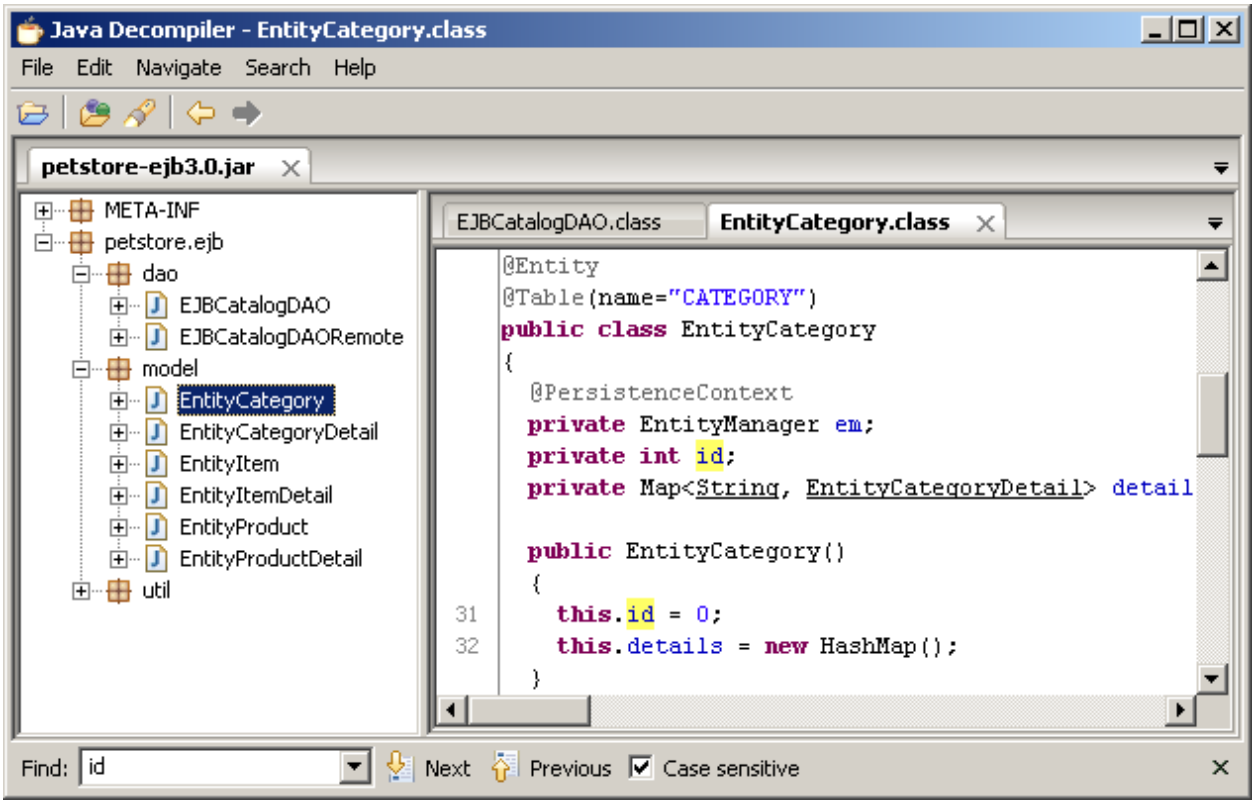

Рисунок 2.1 – Робота інструменту JD-GUI

На жаль, JD-GUI не є настільки ж функціональною, як jADX, але іноді, коли один інструмент не працює, можна спробувати інший і він може справитись.

2.10 JEB

В якості останнього засобу ви можете спробувати декомпілятор JEB. Це дуже хороше програмне забезпечення, але, на жаль, воно не є безкоштовним, є пробна версія, ось як він виглядає:

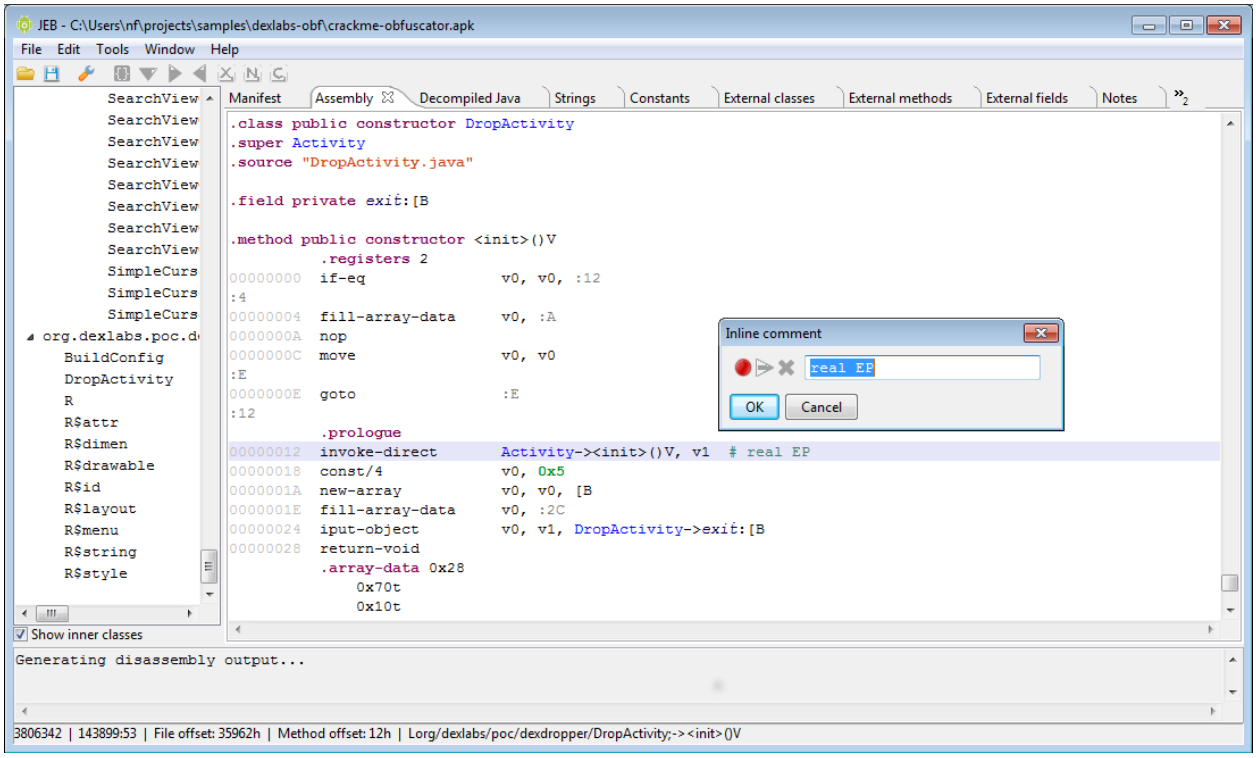

Рисунок 2.2 – Робота інструменту JEB Decompiler

JEB також оснащений диспетчером ARM (корисним, коли існують нативні бібліотеки в APK) та відладчиком (дуже корисний для динамічного аналізу), але знову ж таки це не безкоштовно, і це недешево.

Але навіть такий потужний і дорогий інструмент не надає інструментарію для зручного статичного аналізу. Тому було рішення розробити інструмент, якого не вистачає ні в одному з представлених застосунках.

Висновок до розділу 2

Як можна бачити, функціонал інструментів аналізу Android застосунків що пропонує ринок – доволі широкий, в плані Blackbox тестування. Що ж стосується Whitebox тестування, то існує 2 типи – це автоматичні аналізатори коду, які, звичайно, не зможуть знайти помилок в архітектурі застосунка, а лише неправильне використання криптографічних функцій, чи підозрілі права застосунка. Звичайно вони мають високу ймовірність false positive результатів, і це все рівно потрібно перевіряти методами інспекції коду. Другий же тип – це інструменти роботи, що направленні на інспекцію коду інженером, але їх

функціонал обмежується видобуванням коду з застосунка, вирішення імен змінних, зв’язування ресурсів з кодом. На ринку не має інструменту направленого на полегшення та прискорення роботи з кодовою базою.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗРОБЛЕНИХ АЛГОРИТМІВ

В минулому розділі ми з’ясували, що арсенал дослідника безпеки є доволі широкий і включає масу інструментів для blackbox тестування, але можливості підходу blackbox тестування є обмеженими, так як деякі умови для експлуатації застосунків є надзвичайно специфічними і їх майже неможливо знайти без ясного розуміння, як працює код.

3.1. Дослідження коду

Перегляд коду – це систематична перевірка вихідного коду програми. Призначений не лише для виявлення помилок, але є й одним з етапів розробки програмного забезпечення для покращення загальної якості програмного коду.

Перегляди коду часто сприяють знаходженню та виправленню загальних вразливостей, таких як вразливості форматних рядків, помилки некоректної послідовності виконання частин коду, витоки пам'яті та переповнення буферу, таким чином покращуючи безпеку програмного забезпечення.

Автоматизоване ПЗ для перегляду коду дозволяє зменшити завдання по перегляду великих відрізків коду завдяки автоматичній перевірці вихідного коду на відомі вразливості.

При перегляді коду рекомендують перевіряти 200—400 рядків за годину. Інспектування та перегляд більш ніж декілька сотень рядків коду за годину для критичного ПЗ (наприклад, критичного в плані безпеки вбудованого ПЗ) може бути занадто швидким для того, щоб знайти помилки.

Сучасні проекти мають надзвичайно широку кодову базу, їх розмір може досягати сотні тисяч рядків програмного коду, що потребує великих витрат на проведення аналізу коду. Але це стосується аналізу коду розробниками програмного забезпечення. Аналіз безпеки коду вимагає від інженера не тільки знання мови програмування, але і знання методологій, та розуміння роботи всіх механізмів безпеки, задіяних в коді. Звичайно, це не тільки збільшує час роботи, але і поріг входження.

Варто сказати чому в роботі поняття реверсної інженерії так тісно переплітається з поняттям дослідження коду і є по суті синонімічним. Як було сказано в розділі 1, використання Java байткоду, як основного формату виконуваних файлів системи, робить реверсну інженерію, без використання складних обфускаторів, доволі тривіальною задачею. Тому перехід від реверсної інженерії до повноцінного аналізу коду відбувається через застосування існуючого ПЗ.

Аналіз безпеки коду для Android застосунків розпочинається з аналізу Manifest.xml так як в ньому описані всі точки входу, такі як: експортні активіті, сервіси, контент провайдери, дозволені бекапи. Потім йде дослідження виявлених класів на предмет роботи з даними, їх зберігання, обробка, обмін.

```
<activity android:theme="@android:style/Theme.NoTitleBar.Fullscreen" android:name="com.appodeal.ads.LoaderActivity" android:configChanges="orientation|screenSize"/>
<activity android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen" android:name="com.appodeal.ads.VideoPlayerActivity"/>
<receiver android:name="com.appodeal.ads.AppodealPackageAddedReceiver" android:enabled="true" android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.PACKAGE_ADDED"/>
        <data android:scheme="package"/>
    </intent-filter>
</receiver>
<activity android:theme="@android:style/Theme.Translucent.NoTitleBar" android:name="com.appodeal.ads.TestActivity"/>
<meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
```

3.2. Побудова вимог

Після практичного заглиблення, в процеси тестування на проникнення та дослідження вихідних кодів програм для ОС Android, були виявленні всі складності дослідження вихідних, кодів. Для збільшення ефективності роботи потрібен був інструмент який би полегшував розуміння структури застосунку.

Для вирішення проблеми було вирішено використати існуючі інструменти аналізу. Використання цих інструментів не дало бажаного ефекту. Аналіз роботи знайдених застосунків:

JEB decompiler:

JEB - це програмне забезпечення для дизасемблювання та декомпіляції додатків для Android та власного машинного коду. Він декомпілює байт-код Dalvik з вихідним кодом Java, а також MIPS, ARM, x86 32-bit, x86 64-bit машинний код до вихідного коду C. Дизасембльований та кодовий вивід є інтерактивними і можуть бути рефакторизовані. Користувачі можуть також написати свої власні скрипти та плагіни для розширення функціональності JEB.

JEB 2.2 представив модулі відладки Android для коду Dalvik та рідного (Intel, ARM, MIPS). Користувачі можуть "без проблем відлагоджувати байт-код Dalvik та власний машинний код для всіх додатків, включаючи ті, які явно не дозволяють відладку".

В JEB 2.3 введений нативний декомпілятор коду. Перший декомпілятор, який був поставлений разом з JEB, був 32-розрядним інтерактивним декомпілятором MIPS.

Використання інструменту Call Graph Jeb decompiler:

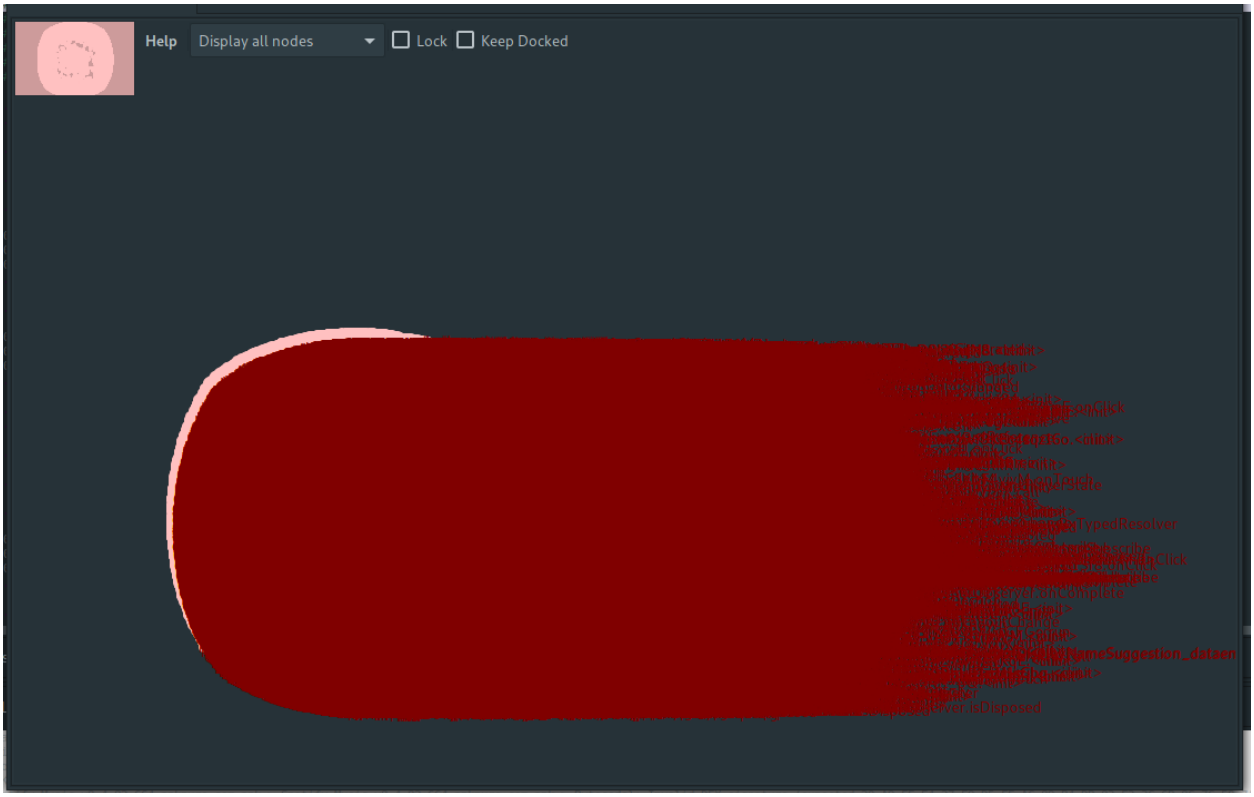

Рисунок 3.1 – Спроба інструменту JEB Decompiler відобразити граф викликів

Як можна побачити з графічного матеріалу (Рисунок 3.1) при використанні цього інструменту ми не можемо говорити про якусь зручність використання. Інструмент використовує всі ресурси ЕОМ, система функціонує нестабільно. Інструмент не дозволяє вимикати непотрібні ноди графу, їх перенасичення унеможливлює використання графу. Все ж, для розуміння роботи інструменту, проведемо аналіз вкрай малого Android застосунку:

Рисунок 3.2 – Відображення графу викликів для відносно малого проекту інструментом JEB Decompiler

Даний код має лише 3 основних класа(Додаток 1) але його аналіз вже ускладнено. Що ж проведем аналіз недоліків які ще має граф. Серед недоліків можна виділити високу розрідженість графу.

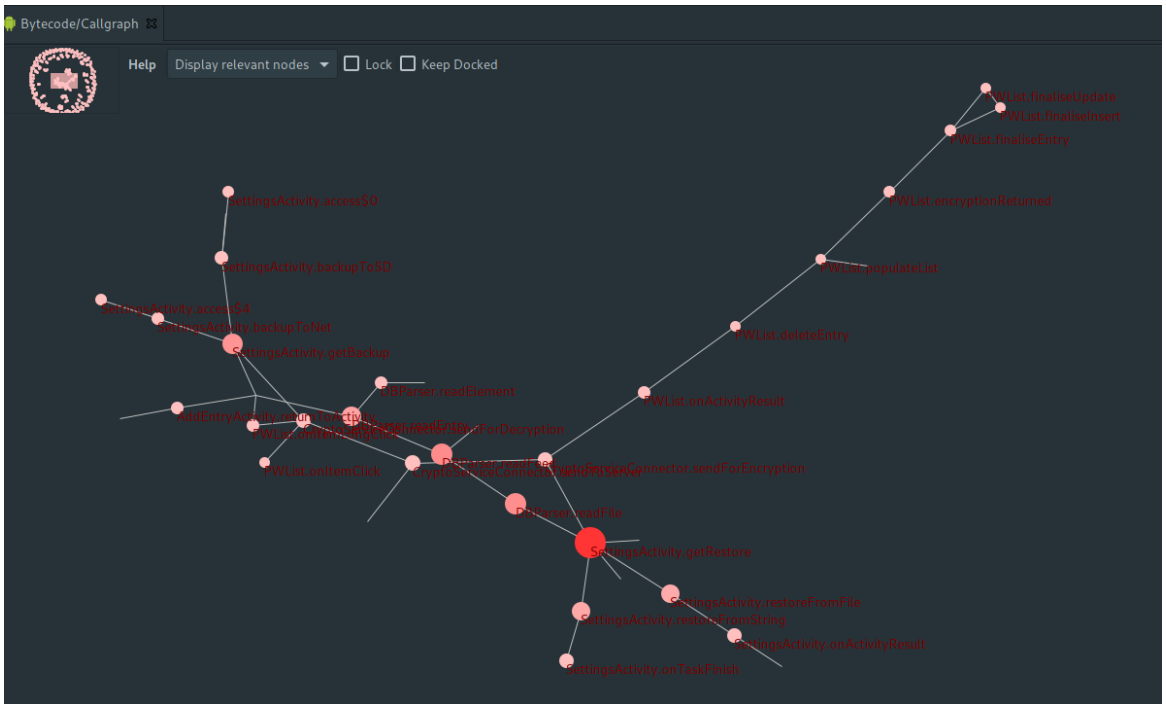

Рисунок 3.3 – Відображення зв’язної частини графу інструментом JEB Decompiler

Smalidroid:

Представник opensource рішення, що використовує систему graphviz для побудови дерева. Має консольний інтерфейс, результати зберігає у вигляді png файлів. На вхід потребує вказати клас, який буде виступати корінним. Це зменшує навантаження, але сильно обмежує функціонал інструменту. До того ж має сильно обмежену глибину викликів, і в великих проектах, програма завершує роботу не видаючи жодного рішення(Рисунок 3.4)

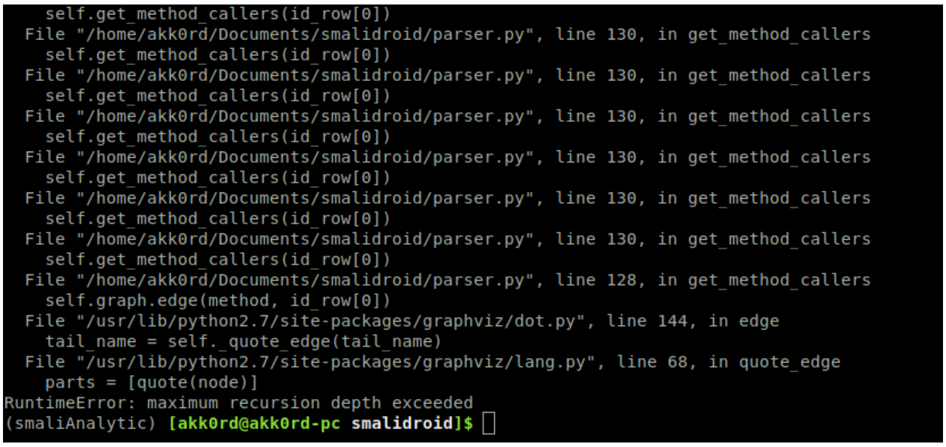

Рисунок 3.4 – Робота інструменту Smalidroid(невдале завершення)

При обробці класів що мають невелику кількість нащадків, інструмент має такий вивід(Рисунок 3.5):

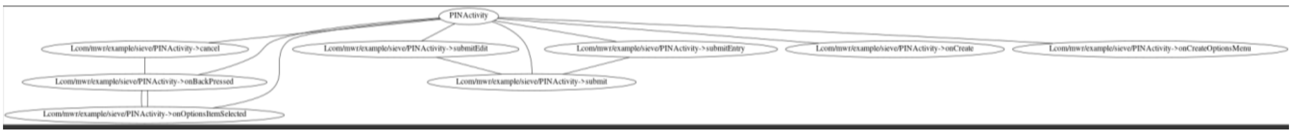

Рисунок 3.5 – Граф викликів на основі конкретної функції, побудовано інструментом Smalidroid

Вивід складно аналізувати, через його абсолютну не інтерактивність. Хоча, слід зазначити, що інструмент працює швидко. Інструмент представляє собою більшість opensource рішень, всі вони схожі за функціоналом.

Таблиця 3.1 – Порівняльна оцінка інструментів в побудові графа виклику

Інструмент	Застосунок, .apk	Час, с	Ресурси avg %		Розрідженість
			CPU	MEM	
Jeb	Complex	347	100	32	висока
	Simple	133	81	26	
Smalidroid	Complex	172	74	28	відсутня
	Simple	36	30	14	

З цих результатів можна вибудувати вимоги до нашого застосунку:

1. Інтерактивність графу
2. Споживання ресурсів не має заважати загальній роботі сучасної ЕОМ
3. Зменшення розрідженості графу, без втрати інформативності.
4. Функціонал аналізу графу, можливість виводу лише частин що аналізуються.
5. Низький час роботи.

3.3. Вибір інструментів побудови

Для ефективної роботи застосунку потрібно обрати відповідні інструменти побудови: мову програмування з можливістю швидкого прототипування, та спосіб побудови графу з можливістю високої інтерактивності.

Мову програмування було обрано за швидкістю прототипування та ознайомленістю з цією мовою(Рис).

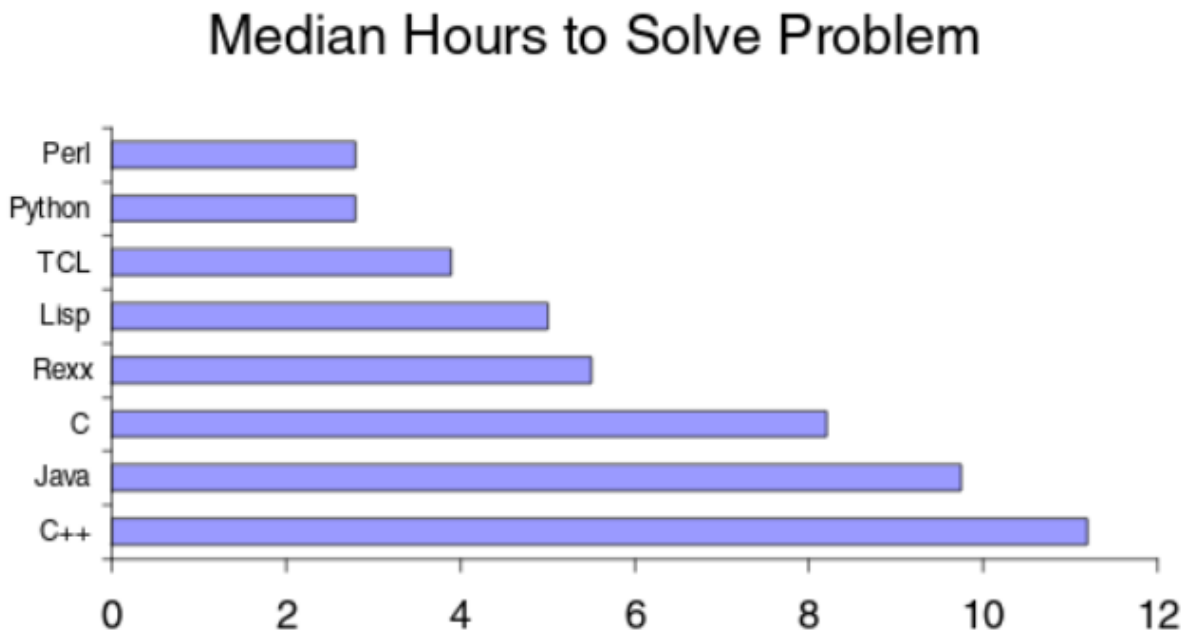

Рисунок 3.6 – Порівняння швидкості прототипування різних мов програмування

Отже, в цій роботі для створення технічної реалізації досліджуваної теми, буде використано мову програмування Python.

Для вибору засобу представлення графу було проведено аналіз можливих бібліотек та інструментів:

Graphviz - програма візуалізації графіки з відкритим кодом. Візуалізація графа є спосіб представлення структурної інформації у вигляді діаграм абстрактних графів і мереж. Це має важливі застосування в області мережевих технологій, біоінформатики, розробки програмного забезпечення, баз даних і веб-дизайну, машинного навчання, так і в візуальних інтерфейсів для інших технічних областей.

Graphviz - набір інструментів бере опис графів, з текстової мови, і робить діаграми в зручному форматі, наприклад, на зображеннях і SVG для веб-сторінок, PDF або Postscript для включення в інші документи, або відображаються в браузері інтерактивні діаграми. (Graphviz Також підтримує GXL, XML-діалект.)

Graphviz має багато корисних функцій для конкретних діаграм, такі як вибір варіантів кольору, шрифтів, табличні макети вузлів, стилі ліній, гіперпосилання, Роллан призначених для користувача форм.

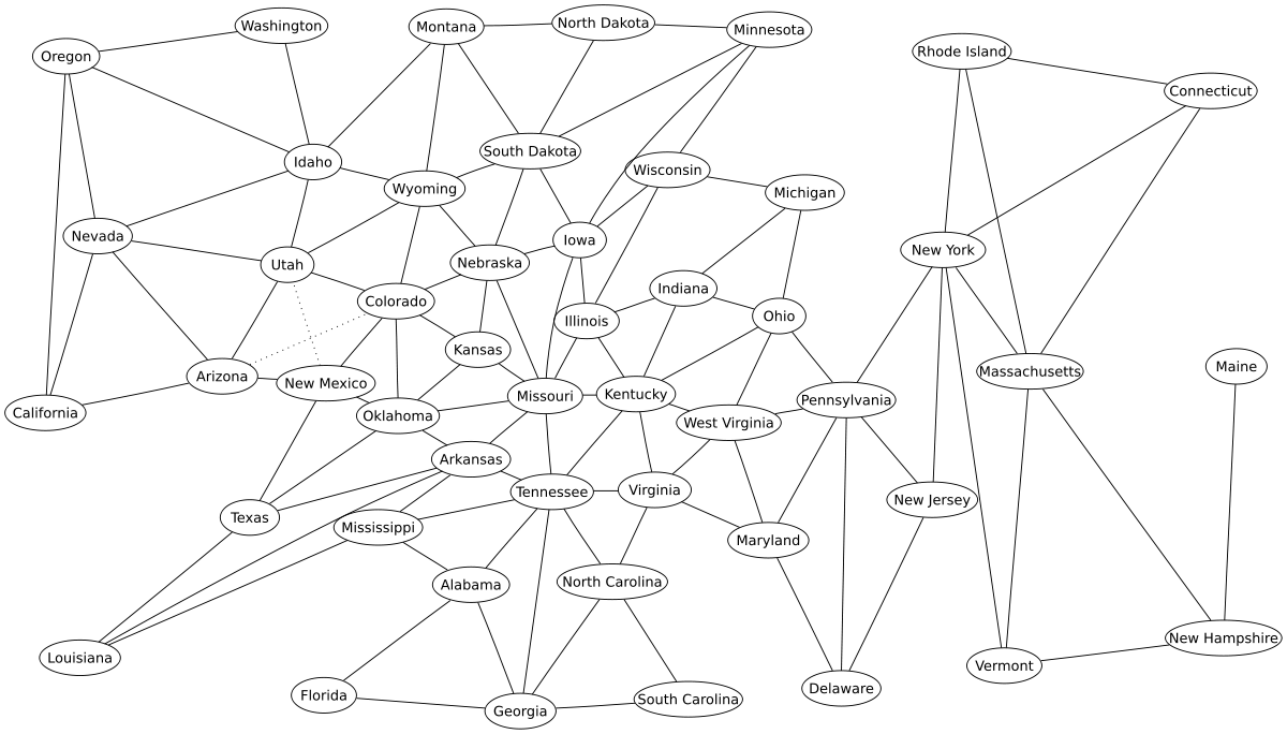

Рисунок 3.7 – Приклад роботи інструменту Graphviz

Недоліки пакету: не має засобів для інтерактивності, складно здійснювати обробку графу.

Matplotlib - це бібліотека Python 2D, яка створює показники якості публікацій у різноманітних друкованих форматах та інтерактивних середовищах на різних платформах. Matplotlib може використовуватися в скриптах Python, оболонках Python та IPython, серверах веб-додатків та чотирьох графічних наборів інструментів для користувальницького інтерфейсу.

```
In [5]: networkx.draw(G)
```

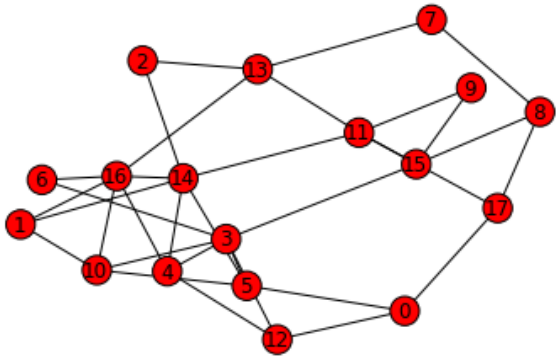

Рисунок 3.8 – Приклад побудови графу за допомогою інструментом Matplotlib

Nodebox. Бібліотека NodeBox Graph включає в себе алгоритми від NetworkX для централічності і централізованості власного вектора, реалізацію Connelly Barnes коротких шляхів Дейкстри та spring прошарок для JavaScript за допомогою Aslak Hellesoy та Dave Hoover. Метою цієї бібліотеки є візуалізація малих графіків (<200 елементів), якщо вам потрібно щось більш надійне, ми рекомендуємо використовувати NetworkX.

Графові бази даних. При підборі бібліотеки побудови графів, було знайдено рішення використовувати графові бази даних, які відкривають широкі можливості для обробки графів. Було використано дослідження з ефективності існуючих графових баз даних(Рисунок 3.9, Рисунок 3.10)

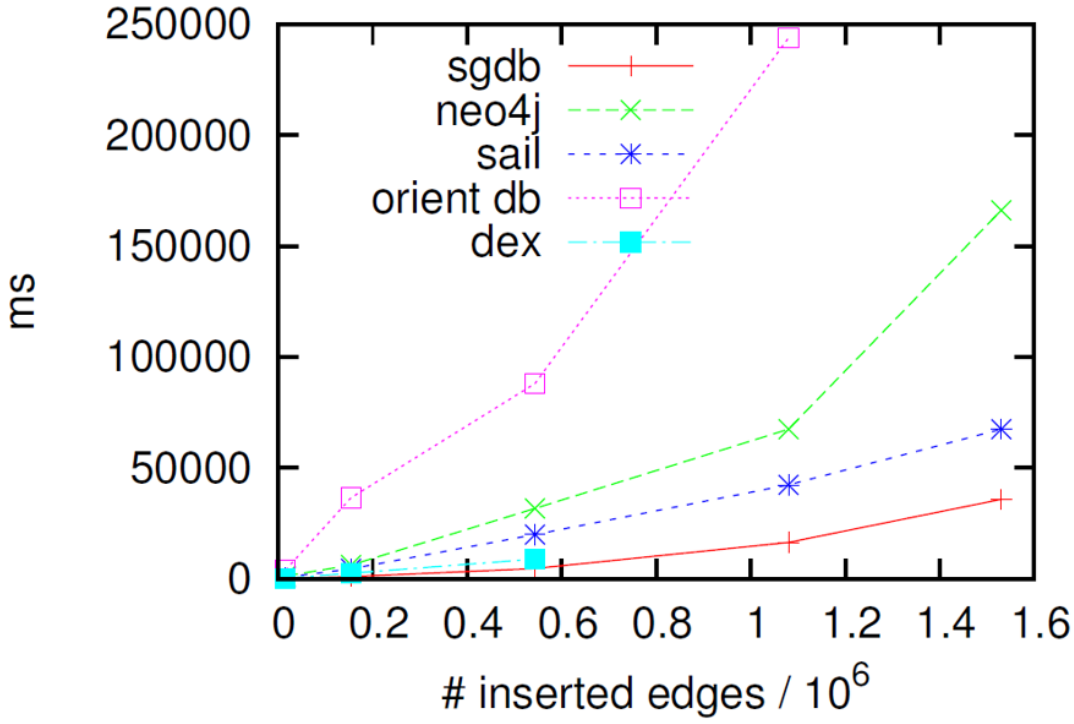

Рисунок 3.9 – Порівняння швидкості вставлення елементів в БД

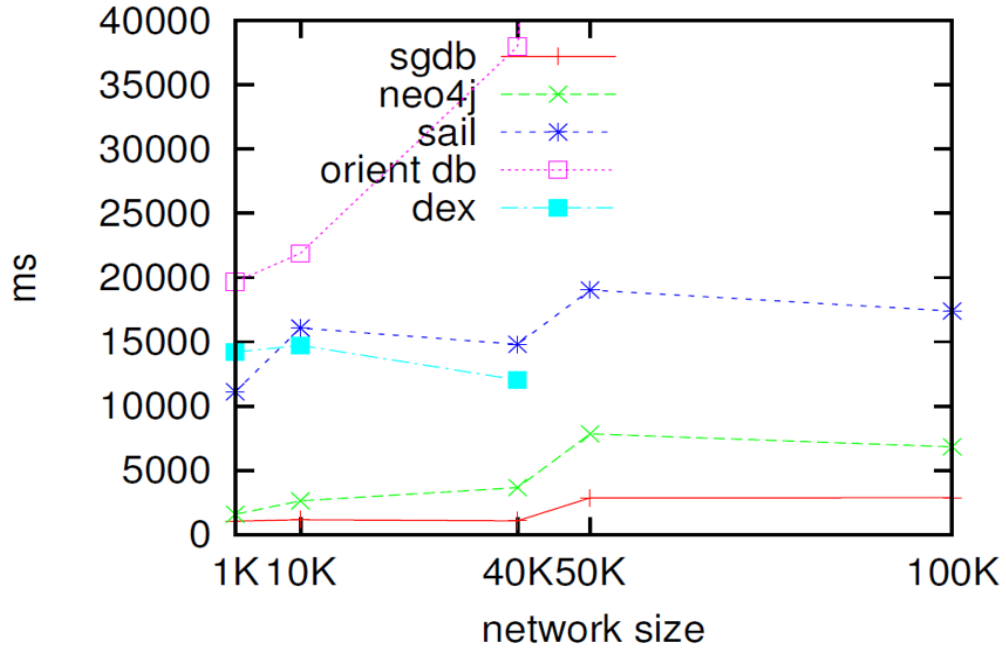

Рисунок 3.10 – Порівняння швидкості пошуку BFS(глибина 3)

Sgdb3 - це вбудована графічна база даних, яка реалізована в JAVA. Це спроба побудувати швидкий, стійкий БД для структури даних графів/мереж. Sgdb3 реалізує Blueprins інтерфейс v.1.0. Коли мова йде про обробку графа, звичайним підходом є завантаження та обробка всього графа в пам'яті, з міркувань продуктивності. Проблеми виникають, коли граф стає занадто великим і більше не вписується в пам'ять. Це пов'язано з випадковим доступом, характерним для обробки графічних даних, що призводить до великої кількості випадкових читань з диска.

Мета Sgdb3 полягає в забезпеченні високої продуктивності обробки графів, навіть для великих графіків.

Neo4j - графова система управління базами даних з відкритим вихідним кодом, реалізована на Java. Станом на 2015 рік вважається найпоширенішою графовою СУБД. Розробник - американська компанія Neo Technology, розробка ведеться з 2003 року .

Дані зберігає у власному форматі, спеціально пристосованому для подання графової інформації, такий підхід в порівнянні з моделюванням графової бази даних засобами реляційної СУБД дозволяє застосовувати додаткову оптимізацію в разі даних з більш складною структурою. Також стверджується про наявність спеціальних оптимізацій для SSD-накопичувачів, при цьому для обробки графа не потрібно його приміщення повністю в оперативну пам'ять обчислювального вузла, таким чином, можлива обробка досить великих графів.

Основні транзакційні можливості - підтримка ACID і відповідність специфікаціям JTA, JTS і XA. Інтерфейс програмування додатків для СУБД реалізований для багатьох мов програмування, включаючи Java, Python, Clojure, Ruby, PHP, також реалізовано API в стилі REST.

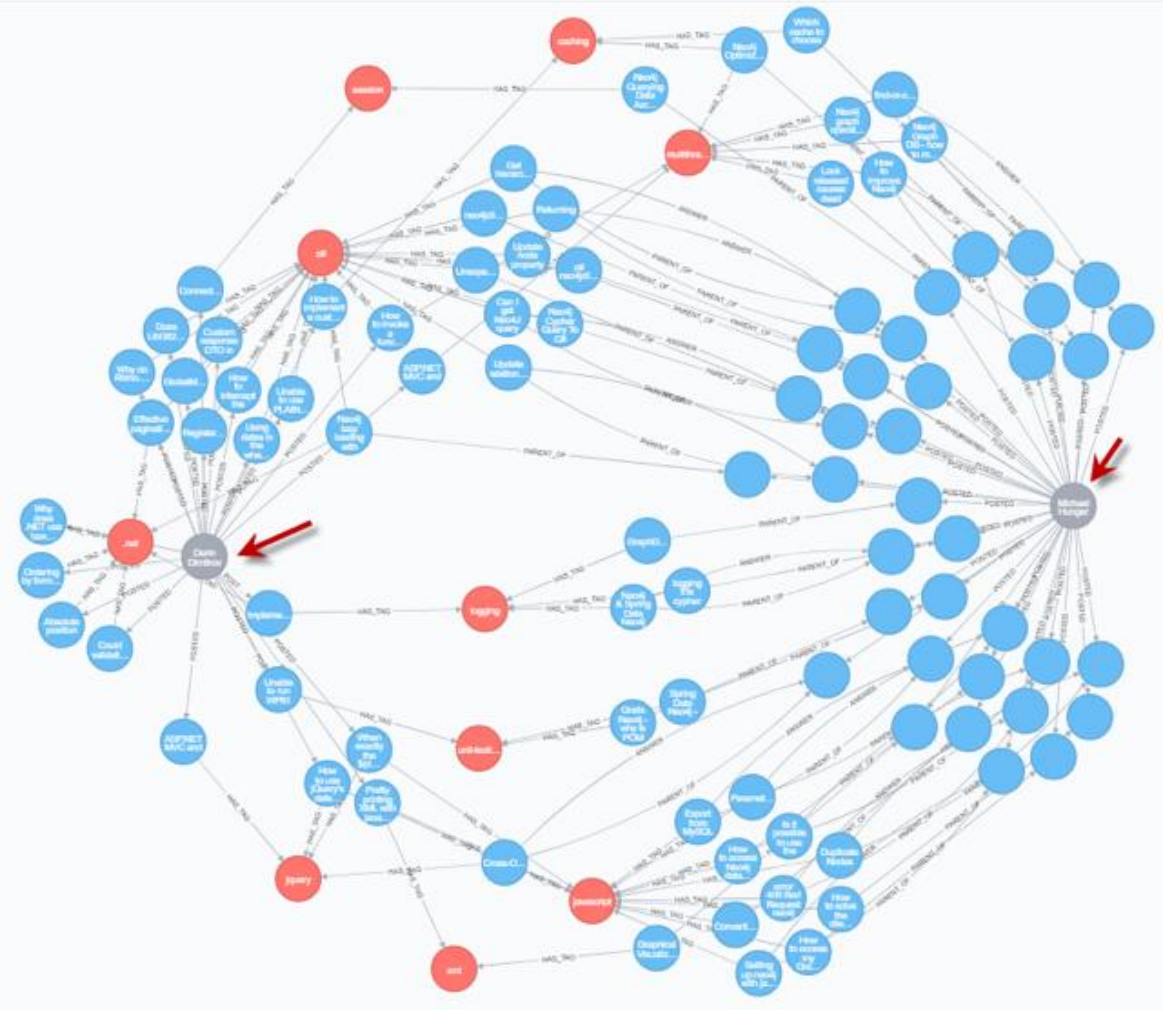

Рисунок 3.11 – Приклад роботи інструменту Neo4J

3.4. Аналіз вихідного коду

В статичному аналізі apk застосунків має місце два типа аналізу, базуючись на Java коді або на Smali.

ART - це керований runtime, який використовуються додатками та деякими системними службами на Android. ART та його попередник Dalvik були спочатку створені спеціально для проекту Android. ART, як runtime виконавець, виконує формат Executable Dalvik та специфікацію байтового коду Dex.

ART та Dalvik сумісні з runtime з використанням бай-коду Dex, тому розроблені додатки для Dalvik повинні працювати під час роботи з ART.

Вибір впав саме на Smali, адже нам не потрібно створювати декомпілятор, тим більше на ринку вже існують чудові інструменти декомпіляції. Працювати з Smali кодом важче для людини, але програмно це зробити простіше. Після прочитання документації по smali байткоду, було вирішено що сутності можна шукати в байткоді за допомогою регулярних виразів, адже вони мають чітку структуру.

```
re.compile(ur'^\.class.*\ (.+(?=\;))', re.MULTILINE)
re.compile(ur'^\.method.+?\ (.+?(?=\())\((.*?)\)(.*?$)(.*?(?=\.end\ method))', re.MULTILINE | re.DOTALL)
re.compile(ur'invoke-.*?\ {(.*?)}, (.+?(?=;))\;\-\>(.+?(?=\()\((.*?)\)(.*?)(?=$|;)', re.MULTILINE | re.DOTALL)
re.compile(ur'move-result.+?(.*?)$', re.MULTILINE | re.DOTALL)
```

Для під’єднання до сервісу бази даних використовується драйвер БД py2neo.

Після першочергового запуску застосунка було отримано такий результат:

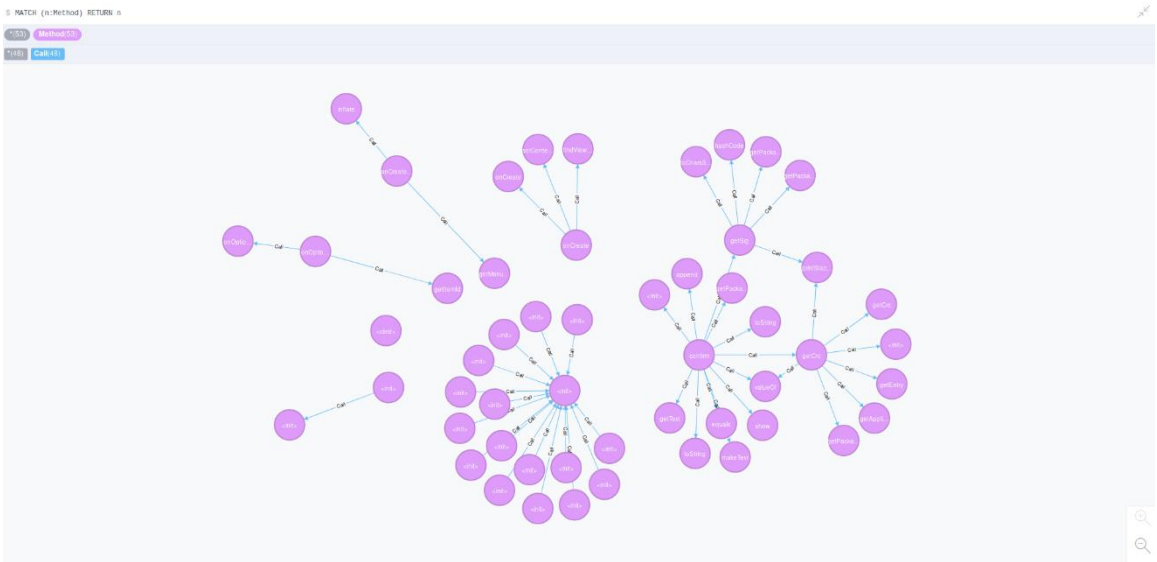

Рисунок 3.12 – Робота створеного інструменту SmaliAnalityc з низькою кількістю класів/функцій

Для повторного запуску був обраний застосунок з більшою кількістю класів, для перевірки всієї системи.

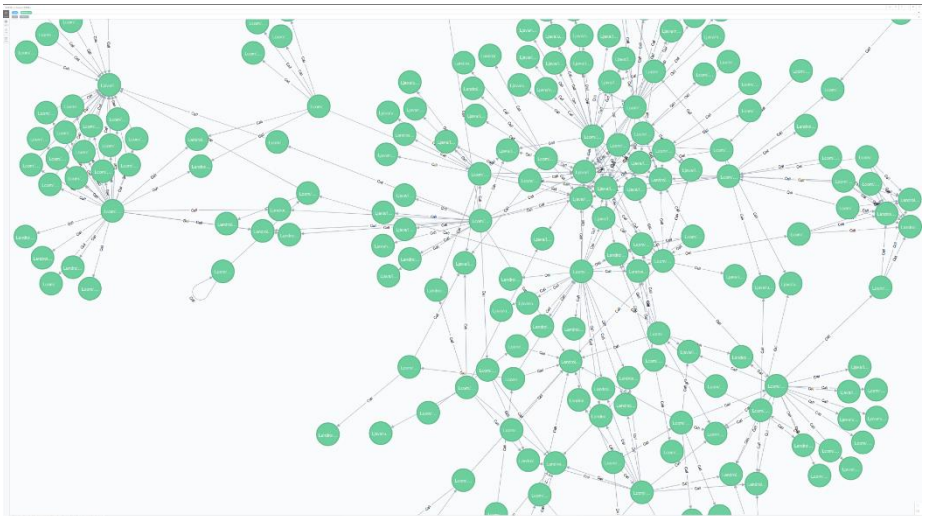

Рисунок 3.14 – Робота створеного інструменту на великому проекті

Графічний інтерфейс Neo4j не намагається втиснути всю графову структуру в екран користувача, роблячи статичним об’єктом не розмір графу, а конкретну вершину цього графу.

Часто аналітик знаходить слабкі місця в кодовій базі, за допомогою автоматичних інструментів або в результаті ручного пошуку. Не завжди зрозуміло без детального розуміння архітектури, чи правильно обробляються дані, що це за дані, і звідки вони приходять до функції. Для цього випадку можна скористатися можливостями графової БД, і знайти всі шляхи від вхідних точок, до потрібної функції:

```
MATCH (a:Method{name:Manifest.xml})-[r*..3]-(c)-[k*..3]-(b:Method{name:'Lcom/mwr/example/sieve
/FileSelectActivity->onItemClick'})
return a,b,c
```

MATCH (a:Method{name:Manifest.xml})-[r*..3]-(c)-[k*..3]-(b:Method{name:'Lcom/mwr/example/sieve/FileSelectActivity->onItemClick'})

return a,b,c

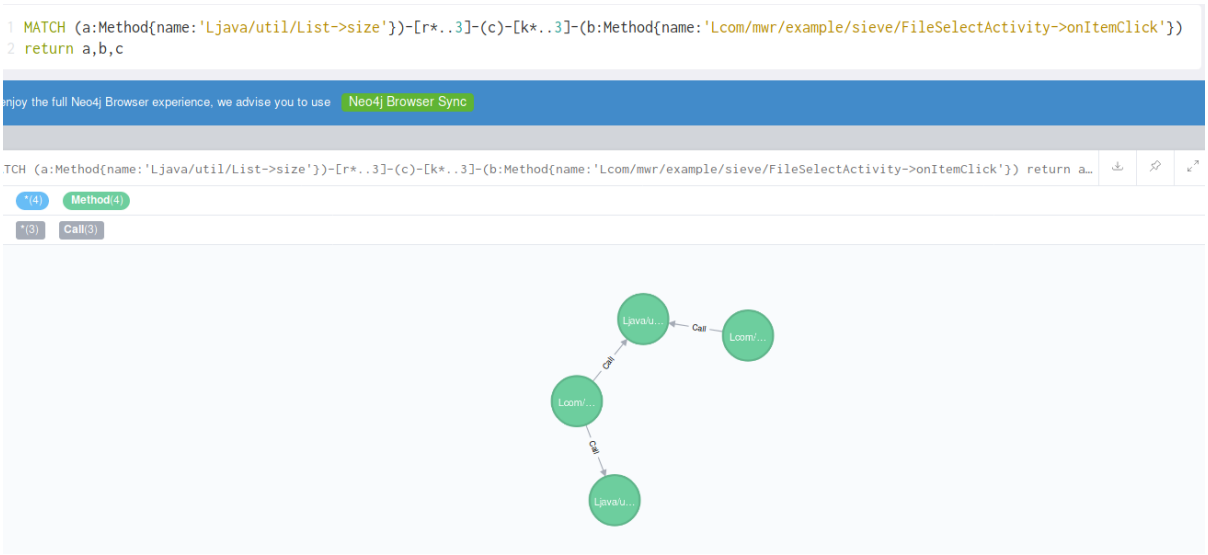

Рисунок 3.15 – Пошук шляхів між функціями в застосунку

Особливість створеної моделі полягає також в тому, що в графовому представленні розділяється клас та ім’я функції, що дозволяє робити пошук по всих стандартних функціях.

Висновки до розділу 3

В даному розділі представлений процес розробки інструменту для тестування безпеки Android-застосунків. Розробка даного інструменту була логічним продовженням теоретичних та практичних досліджень в сфері кібербезпеки мобільних застосунків. В цьому розділі можна бачити кропіткий процес підбору інструментів та аналізу конкурентів на ринку мобільної безпеки, що придставляють близький функціонал, а також неспроможність цих інструментів вирішувати задачі, які були поставлені за мету вирішення розробленим інструментом.

Інструмент має недолік, вирішення якого не має наукової новизни, але корисний в інструменті, як в продукті – це аналіз обфускованих застосунків. Цей недолік можна прибрати, використовуючи пропрієтарне програмне забезпечення.

Отже, розроблений інструмент виконує поставлену мету, і покриває одну з задач пошуку вразливостей, а саме тестування архітектурних особливостей продукту.

4 АНАЛІЗ ВИХОДУ НА РИНОК СТАРТАП ПРОЕКТУ

4.1 Опис ідеї стартап проекту

Опис ідеї стартап проекту полягає в описі основних характеристик проекту, визначенні його сильних та слабких сторін, напрямків застосування, потенційних користувачів та проведенні порівняльного аналізу з конкурентами в галузі. Ця інформація оформлена у вигляді таблиць та наведена нижче за текстом (табл. 4.1-4.2).

Таблиця 4.1 – Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Пропонується продукт, для використання в процесі тестування Android-застосунків. Дозволяє швидко встановлювати зв’язки між функціями та класами, виявляти недоліки в архітектурі застосунки. Продукт легко включається в процес SDL. Інструмент буде цікавий як розробникам мобільних застосунків, що мають свій відділ тестування безпеки, так і компаніям що займаються тестуванням на проникнення та технічним аудитом. Інструмент легко працює з smali-кодами, порог входження низький.	1. Використання інструменту компаніями що займаються аудитом безпеки.	1. Збільшення числа клієнтів за рахунок прискорення процесу тестування. 2. Підвищення якості аудитів.
	2. Використання інструменту компаніями що створюють мобільні застосунки.	1. Забезпечення безпеки своїх продуктів. 2. Оптимізація часу виконання тесту на безпеку застосунку. 3. Підвищення ефективності тестів.

Конкурентами є потужні інструменти, що зарекомендували себе на ринку. Основною відмінністю є те, що створений інструмент дозволяє автоматизувати аналіз архітектури, використовуючи лаконічний інтерфейс графу. Конкуренти ж можуть лишу будувати сам граф – представляючи його як кінцевий результат, але в умовах того, що сучасні проекти надзвичайно великі цей підхід незручний.

Довгостроковими перспективами є:

- Збільшення кількості клієнтів, що будуть використовувати запропоновані методи.
- Еволюція інструменту: зменшення розрідженості графу шляхом аналізу неявних способів взаємодії.
- Додавання деобфускаторів для розширення можливостей інструменту.
- Додавання декомпіляторів, або взаємодії з існуючими для «безшовного» використання продукту.

Таблиця 4.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	*Техніко-економічні характеристики ідеї*	*(Потенційні) товари/концепції конкурентів*		*W (слабка сторона)*	*N (нейтральна сторона)*	*S (сильна сторона)*
		Мій проект	*Конкурент*			
1	2	3	4	5	6	7
1.	Економічні	Витрати на заробіттну плату співробітників, утримання офісних приміщень та потужностей, амортизаційні витрати – 150000₴	Витрати на заробіттну плату співробітників, утримання офісних приміщень та потужностей, амортизаційні витрати – 500000₴	Недостатність витрат на маркетинг, але популяризація проекту забезпечується на даному більш дешевим шляхом, або безкоштовно (інформацію наведено нижче). При подальшому розвитку компанії показники будуть перерозподілені (що також не суттєвим недоліком, так є неминучим при розвитку)	Достатність ресурсів для підтримки проекту.	Значно менші витрати на утримання будівель та ресурсів за рахунок невеликого штату та відсутності необхідності в утриманні великих потужностей (в зв’язку з вузьким напрямком діяльності та її специфікою), а також відданням деяких видів діяльності на аутсорсинг.
2.	Технічні	Використання невеликих потужностей в зв’язку з вузьким напрямком роботи.	Використання великих потужностей в зв’язку з суміжними видами діяльності корпорації.	Для розвитку проекту виникнуть суміжні види діяльності, що потребуватимуть більших потужностей.	Достатність ресурсів для підтримки проекту.	Значно менші витрати на підтримання ресурсів в зв’язку з вузьким напрямком діяльності та її специфікою, а також відданням деяких видів діяльності на аутсорсинг.

Продовження таблиці 4.2

1	2	3	4	5	6	7
3.	Безпеки	Забезпеченя фізичної безпеки, використовуючи власні ресурси. Забезпечення конфіденційності даних, що передаються третім сторонам для обробки та зберігання.	Використання послуг сторонніх компаній для забезпечення як фізичної безпеки, так і інформаційної. Ускладнене розподіленністю офісів компанії.	Необхідність наявності відповідних процесів та відповідальність за їх належне функціонування. Забезпечення конфіденційності інформації, переданої на обробку та зберігання третім сторонам, а також відповідальність, гарантується контрактними зобов’язаннями.	Процеси забезпечуються та виконуються належним чином.	Скорочення витрат та можливість повного контролю процесів і швидкого реагування. Скорочення витрат на утримання необхідного персоналу на постійній основі.
4.	Екологічні	Мінімізація негативного впливу на навколишнє середовище забезпечується використанням технологій кондиціонування та енергозбереження.	Розподіленість систем та використання послуг провайдерів серверних рішень не дає змоги контролю забезпечення екологічності.	Необхідність наявності відповідних процесів та технологій, відповідальність за їх належне функціонування.	Процеси забезпечуються та виконуються належним чином.	Скорочення витрат в зв’язку з забезпеченням енергоефективності та відсутність шкоди довкіллю.

1	2	3	4	5	6	7
5.	Масштабовність	Зручність адаптації до сфери покриття, зручність масштабування у випадку виникнення суміжних видів дяльності.	Необхідність внесення суттєвих змін в більшість процессів.	Потрібно задіюяти науково-технічний потенціал, створювати R&D центри.	Процеси забезпечуються та виконуються належним чином.	Легкість адаптації до будь-яких потреб та умов дільності користувача, забезпечення можливості надання повної підтримки по продукту.
6.	Ергономічні	Можливість надання продукту по різним планам, створення підписки або повна купівля продукту.	Надання продукту в певній детермінованій формі.	Необхідність наявності відповідних процесів та технологій, відповідальність за їх належне функціонування.	Процеси забезпечуються та виконуються належним чином.	Легкість адаптації до будь-яких потреб та умов дільності користувача, забезпечення можливості надання послуг ширшому колу користувачів.

4.2 Технологічний аудит ідеї проекту

Таблиця 4.3 – Технологічна здійсненність ідеї проекту

№ п/п	*Ідея проекту*	*Технології її реалізації*	*Наявність технологій*	*Доступність технологій*
1	2	3	4	5

Продовження таблиці 4.3

1	2	3	4	5
1.	Інструмент аналізу архітектури застосунку через графове представлення, надається як інструмент для тестування безпеки Android-застосунків	Для впровадження даного проекту необхідно: 1. Найняти команду технічних спеціалістів для забезпечення підтримки та актуалізації моделі	Технології є доступними та стосуються виключно найму персоналу або передачі певного виду робіт на аутсорсинг.	Технологія є доступною
		1. Найняти команду технічних спеціалістів для створення основного продукту	Технології є доступними та стосуються виключно найму персоналу або передачі певного виду робіт на аутсорсинг.	Технологія є доступною
		Створення R&D центру для дослідження безпеки мобільних застосунків. Розвивати співпрацю з	Технологія є доступною, але складною, потрібно найняти висококваліфікований персонал, співпрацювати з університетами, досліджувати науковий стан сфери	Технологія є доступною

Продовження таблиці 4.3

1	2	3	4	5
		закладами вищої освіти для вирішення наукових проблем представлених в продукті, та пошуку працівників.		
		3. Практична реалізація продукту, та його тестування для забезпечення якості.	Технології є доступними та стосуються найму кваліфікованого персоналу та побудови чіткого технічного завдання. Частину роботи можна віддати на аутсорсинг.	Технологія є доступною
1. Обрана технологія реалізації ідеї проекту: • Найняти команду технічних спеціалістів для забезпечення підтримки користувачів • Найняти команду технічних спеціалістів для створення основного продукту • Створити R&D центру для дослідження безпеки мобільних застосунків, та еволюція продукту на основі цієїх інформації • Практична реалізація продукту.				

За результатами аналізу, наведеного вище, було встановлено, що даний проект є технічно реалізовним. Оскільки для реалізації необхідно лише найняти вищевказаний персонал з відповідним рівнем кваліфікації у вищевказаних питаннях, то стратегією до реалізації цього завдання обрано проведення ряду інтерв'ю з кандидатами та формування команд, аналіз доцільності передачі певних процесів на аутсорсинг та, відповідно, передача пених процесів на аутсорсинг або найм персоналу для виконання їх всередині компанії.

4.3 Аналіз ринкових можливостей запуску стартап проекту

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	*Показники стану ринку (найменування)*	*Характеристика*
1	2	3
1	Кількість головних гравців, од	20
2	Загальний обсяг продаж, грн/ум.од	150 000 ум.од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Необхідність початкового капіталу та наявність конкуренції
5	Специфічні вимоги до стандартизації та сертифікації	-
6	Середня норма рентабельності в галузі (або по ринку), %	Рентабельність проекту складає 125%, що є вищим за показник рентабельності банківського вкладу на 10-13%.

4.4 Розроблення маркетингової програми

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*
1	2	3	4	5
1.	– Потреба клієнтів в організації безпеки мобільних застосунків, сучасні тренди розвитку інформаційної безпеки, та законопроекти ціллю яких є контроль безпеки конфеденційних даних користувачів, формують попит в розробників програмного забезпечення.	– Компанії що займаються аудитом безпеки компаній, та надають послуги тестування безпеки Android-застосунків – Компанії, що розробляють власні застосунки для ОС Android, з використанням методик впровадження безпеки своїх застосунків (наприклад SDL)	– Для компаній, що займаються аудитом кібербезпеки, буде цікаво використання плагінів, а також використання нашого продукту у зв'язці з вже зарекомендованими інструментами на ринку – Для компаній, які є розробниками програмного забезпечення для ОС Android, більш цікаво саме простота, та наявність розвиненої системи підтримки.	– Підтримка продукту, вирішення проблем, врахування побажань користувачів у наступних оновленнях – Гарна документація продукту, розвинене API – Швидкість продукту, зручність користування продуктом

Таблиця 4.6 – Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	2	3	4
1.	Недостатня інформованість суспільства щодо питань інформаційної безпеки та необхідності побудови систем захисту інформації	Потенційні користувачі нехтують питаннями забезпечення захисту інформації через недостатню обізнаність в галузі інормаційної безпеки і, як наслідок, неусвідомлення криточності інформаційних активів та можливих наслідків для організації; вразливостей та загроз, застосовних до систем, в яких проводиться обробка та/або зберігання інформації і каналів її передачі.	Підвищувати інформованість суспільства стосовно питань інформаційної безпеки, брати участь у різноматних конференціях та подібних заходах, поширення інформації за допомогою соціальних мереж та публікації статей, що мають просвітницький характер; виступаючи на конференціях спрямованих на проблеми інформаційної безпеки та співпрацюючи з венлорами мобільних застосунків
2.	Недостатність стартового капіталу	Неможливість подолання бар’єру виходу на ринок через недостатність стартового капіталу.	Залучення інвесторів, які отримуватимуть певний відсоток майбутього прибутку.
3.	Наявність конкуренції	Неможливість подолання бар’єру виходу на ринок через велику конкуренцію в галузі.	Орієнтація на здобуття конкурентої переваги шляхом покращення якості продукту, розвиток функціоналу, якого немає в конкеурентів.

Таблиця 4.7 – Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	2	3	4
1.	Необхідність забезпечення належного рівня захисту мобільних застосунків та користувачів вендора від кібератак.	Інструмент створено на основі потреб галузі тестування безпеки, та надає можливість ефективного аналізу застосунків.	Забезпечення та підтримання виского рівня продукту, а також надання підримки всим користувачам. Ці процеси функціонують завдяки потужній науково-технічній базі що закладено з початку роботи проекту.
2.	Необхідність відповідності сучасним регламентним законам щодо обробки і зберігання користувацьких даних	В зв’язку з турбованістю країн станом інформаційної безпеки – вони впроваджують закони що впливають на процеси керування користувацькими даними, в результаті цих змін, компаніям потрібно підвищувати рівень кіберзахисту та безпеки застосунків, адже крім самого факту кіберзлочину, та репутаційних збитків це може потягнути за собою і низку санкційних мір з боку регулювальника.	Забезпечення та підтримання виского рівня продукту, а також надання підримки всим користувачам. Ці процеси функціонують завдяки потужній науково-технічній базі що закладено з початку роботи проекту.
3.	Підвищення рівня стурбованності питаннями інформаційної безпеки	Внаслідок глобальної інформатизації, рівень критичності інформаційних активів для організації набуває найвищого значення з точки зору впливу на бізнес, та, головне – досягнення організацією її цілей, що є основним показником ефективності.	Розповсюдження інформації щодо важливості питань інформаційної безпеки вищевказаними методами.

Таблиця 4.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1	2	3
1. Тип конкуренції - чиста	Конкуренція характерна для галузі з низьким ступенем монополізації	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів
2. За рівнем конкурентної боротьби - інтернаціональний	Конкурентна боротьба ведеться на міжнародному рівні.	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів
3. За галузевою ознакою - внутрішньогалузева	Конкуренція в межах галузі кібербезпеки мобільних застосунків.	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів
4. Конкуренція за видами товарів: - товарно-родова	Товарно-родова - конкуренція між різними видами товарів, які можуть виконувати подібні функції. Мається на увазі конкуренція з боку товарів-субститутів (замінників).	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів
5. За характером конкурентних переваг - нецінова	Конкуренція за рахунок покращення якості надаваних послуг	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів
6. За інтенсивністю - марочна	Необхідність випуску ряду товарів під однією маркою	Забезпечення більш високого рівня програмного забезпечення, використання інноваційного продукту як основної переваги серед конкурентів

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	*Прямі конкуренти в галузі*	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
1	2	3	4	5	6
Складові аналізу	Аудитові компанії в сфері інформаційної безпеки; Компанії, що створюють та розповсюджують практичні рішення для забезпечення інформаційної безпеки. Основний фокус на конкурента PNF Software	Потенційними конкурентами можуть виступати невеликі опенсорсні проекти, які почнуть створювати аналогічний функціонал при його популяризації на ринку.	В зв'язку зі специфікою роботи питання про вплив зі сторони постачальників не є застосовним.	Дана послуга є дедалі більш необхідною в умовах переходу до використання мобільних рішень клієнти не чинять впливу на умови роботи на ринку, але можуть створювати певний попит відносно до ступеня їх обізнаності в питаннях інформаційної безпеки та розуміння необхідності впровадження систем захисту.	Фактори загроз з боку зоку товарів-субститутів полягають в тому, що користувачі даної послуги можуть обирати між пропозиціями базуючись на власних знаннях та на рекламних компаніях. Пир цьому користувач може необ'єктивно оцінити послугу з огляду на недостатність вхідної інформації або її неактульність, а також через відстуність розуміння того, що саме очікується від надаваємої послуги, а також як трактувати результати проведеної роботи. Так як, наприклад, інструмент орієнтовано на системи захисту інформації в мобільних застосунках і диференціація критичності визначеного переліку засобів безпеки є дещо іншою, ніж для систем, на які орієнтовано послуги компаній-конкурентів.

Продовження таблиці 4.9

1	2	3	4	5	6
Висновки:	Конкуренція в галузі інструменто орієнтованої інформаційн ої безпеки для Android застосунків не є досить високою. Великі компанії-конкуренти займаються суміжними питаннями безпеки,тому вихід проекту на ринок не буде суттєво ускладнено питаннями конкуренції.	– Можливості виходу на ринок обгрунтовані рентабельністю проекту, попитом, створеним за разунок підвищення рівня стурбованності суспільства питаннями інформаційної безпеки;	В зв’язку зі специфіко ю роботи питання про контроль умов роботи на ринку зі сторони постачаль ників не є застосовн им.	Дана послуга є дедалі більш необхідною в умовах переходу до використання мобільних застосунків; клієнти не чинять впливу на умови роботи на ринку, але можуть створювати певний попит відносно до ступеня їх обізнаності в питаннях інформаційної безпеки та розуміння необхідності впровадження систем захисту.	В зв’язку зі специфікою роботи питання про обмеження роботи на ринку зі сторони товарів-субститів не є застосовним.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1.	Частка ринку	Враховуючи той факт, що тип родового середовища в галузі – консолідований ринок, тобто існує група компаній, які контролюють разом понад 40% ринку, а також те, що інтенсивність суперництва між діючими конкурентами при низьких темпах зростання ринку є однією з головних сил, які діють на конкуренцію в галузі, одним з найважливіших факторів конкурентоспроможності виступає частка ринку, яку займає виробник. В таких умовах чим більше частка ринку, тим більшими ринковими можливостями володіє виробник.
2.	Цінотворення інструменту	Чим вигіднішою є ціна для споживача, тим вірогідніше його вибір.
3.	Репутація виробника	Якщо компанія має бездоганну репутацію, особливо у сфері якості своєї продукції, то рівень довіри до неї зростає. Також репутація виробника важлива при виході на ринок з новими товарами, або при виході на нові сегменти, що полегшує позитивне сприйняття новинок.
4.	Рівень лояльності до бренду	Чим вище рівень лояльності, тим більше компанія має прихильних, а значить постійних споживачів.
5.	Унікальність позиціонування	В умовах монополістичної конкуренції, коли фактор диференціації ТМ є ключовим засобом ведення конкурентної боротьби, важливим є створення та підтримання унікального позиціонування, що створює певний захист від конкурентних зіткнень.
6.	Маркетинговий бюджет	Від розміру маркетингового бюджету залежить здатність здійснювати маркетингову стратегію підприємства. Маркетингові заходи мають забезпечувати інші конкурентні переваги такі, як рівень диференціації, лояльності, репутація виробника, дистрибуція та просування в торгових точках.

Таблиця 4.1 – Порівняльний аналіз сильних та слабких сторін «назва проекту»

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з даним проектом*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1.	Частка ринку	20					I		II
2.	Ціноутворення інструменту	10	II		I				
3.	Репутація виробника	12					I	II	
4.	Рівень лояльності до бренду	14					I,II		
5.	Унікальність позиціонування	15		I		II			
6.	Маркетинговий бюджет	10			I				II

Умовні позначки позицій конкурентів:

I - конкурент 1

II - конкурент 2

Отже, відповідно до проведеного аналізу можна сказати, що «Метод оцінювання ефективності контролів безпеки» має наступну позицію на ринку:

сильні сторони:

– унікальне позиціонування;

– ціноутворення інструменту;

слабкі сторони:

– лояльність до бренду

– захоплення ринку

Виділивши найвагоміші сильні та слабкі сторони розробленого інструменту у порівнянні з основними конкурентами і з аналізу внутрішніх факторів та використовуючи результати аналізу маркетингових загроз та можливостей, складемо матрицю SWOT-аналізу (табл. 4.12.).

Таблиця 4.2 – SWOT- аналіз стартап-проекту

Сильні сторони: • Унікальне позиціонування продукту, таким що він представляє рішення нетривіальних задач циклу пошуку вразливостей • Цінотворення інструменту на основі існуючих компонентів об’єднаних в інноваційний продукт, створює умови для низької ціни на етапі виходу на ринок	Слабкі сторони: • Так як бренд ще не зарекомендував себе на ринку, а позиції конкурентів сильні це створить перепони • Захоплення ринку відбувається поступово
Можливості: • Можливість зміцнення іміджу • Можливість збільшення обсягів реалізації • Можливість збільшення обсягів продаж за рахунок експансії	Загрози: • Загроза працювати без прибутку скорочення платоспроможного попиту • Загроза втрати споживачів внаслідок підвищення тиску зі сторони товарів-субститутів • Загроза підвищення цін

Таблиця 4.3 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1.	Підвищувати інформованість суспільства стосовно питань інформаційної безпеки, брати участь у різноматних конференціях та подібних заходах, поширення інформації за допомогою соціальних мереж та публікації статей, що мають просвітницький характер; кооперуватись з вендорами мобільних застосунків, інформаційних технологій та засобів захисту інформації для взаємного отримання вигоди в наслідок дяльності цієї кооперації.	Орієнтація на здобуття конкурентої переваги шляхом покращення якості продукту.	3-5 місяців

4.5 Розроблення ринкової стратегії проекту

Таблиця 4.4 – Вибір цільових груп потенційних споживачів

№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	2	3	4	5	6
1.	Розробники програмного забезпечення для ОС Android, які прагнуть підвищити безпеку своїх застосунків забезпечити повне покриття питань захисту інформації та вести свою діяльність у відповідності до міжнародних стандартів.	Готовність споживачів до сприйняття обумовлена необхідністю підвищення рівня безпеки.	Високий попит в зв’язку з легкістю використання продукту та вирішення задач, що не вирішує жоден з продуктів.	Інтенсивність конкуренції в сегменті не є високою.	Середня складність входу в сегмент в зв’язку з специфікою проекту.
2.	Компанії що займаються аудитом безпеки застосунків	Готовність споживачів обумовлена бажанням підвищити ефективність та швидкість виявлення помилок безпеки.	Високий попит в зв’язку з легкістю використання продукту та вирішення задач, що не вирішує жоден з продуктів.	Інтенсивність конкуренції в сегменті є високою.	Середня складність входу в сегмент в зв’язку з специфікою проекту.
Які цільові групи обрано: Розробники застосунків, Аудитові компанії					

Таблиця 4.5 – Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1.	Популяризація питань ІБ (публікації статей, участь у конференціях); Кооперування з більш відомими компаніями з метою отримання взаємовигідних результатів.	Стратегія диференційованого маркетингу	Орієнтованість на важливі з точки зору споживача властивості, які роблять товар відмінним від товарів конкурентів	Стратегія диференціації

Таблиця 4.6 – Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1.	Даний інструмент є першопрохідцем на ринку,і пропонує недоступні раніше можливості для аналізу застосунків	Передбачується утворення власного кола споживачів, оскільки модель орієнтована на певну визначену аудиторію; при цьому не можна виключати того, що частина споживачів перейде від конкурентів за рахунок більш профільного підходу та ряду якісних переваг.	Компанія не буде копіювати характеристики товару конкурента, а саме навпаки займати ніші, не зайняті конкурентами, що полегшить вихід на ринок.	Створення власного кола споживачів, чія діяльність базується на забезпеченні безпеки мобільних застосунків.

Таблиця 4.7 – Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)*
1.	• Швидкість аналізу вихідних кодів • Відносна простота у використанні • Інтеграція з використовуваними інструментами	Стратегія диференціації	Вигоди, які пропонуються в даному проекті, його конкурентні фактори (див. табл. 4.18)	• Унікальність • Доступна ціна • Реалізація нових методів

4.6 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.8 – Визначення ключових переваг концепції потенційного товару

№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити*
1.	Потреба в проведенні тестування безпеки мобільних застосунків ОС Android, для виявлення вразливостей, та їх усунення.	Забезпечення зручного та швидкого аналізу архітектурних вразливостей систем.	Автоматизація аналізу архітектурних особливостей систем, з використанням інструментів аналізу графів.

Надалі розробляється трирівнева маркетингова модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (табл. 19).

Таблиця 4.9 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Продукт, для використання в процесі тестування Android-застосунків. Дозволяє швидко встановлювати зв’язки між функціями та класами, виявляти недоліки в архітектурі застосунки. Продукт легко включається в процес SDL. Інструмент буде цікавий як розробникам мобільних застосунків, що мають свій відділ тестування безпеки, так і компаніям що займаються тестуванням на проникнення та технічним аудитом. Інструмент легко працює з smali-кодами, порог входження низький.</td></tr>
<tr><td rowspan="6">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Орієнтовність на пошук архітектурних вразливостей
2. Зручність аналізу
3. Швидкість роботи продукту
4. Актуальність</td><td>М

М
М</td><td></td></tr>
<tr><td colspan="3">Якість: ISO/IEC 27001, повне покриття тестами(Unit, Smoke, System)</td></tr>
<tr><td colspan="3">Пакування: Надання користувачу підтримки, та реагування на всі зауваження та пропозиції щодо продукту.</td></tr>
<tr><td colspan="3">Марка: BigVadik- SmaliAnalityc</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="3">До продажу: популяризація інформаційної безпеки, виступи / публікації; проведення декількох промоакцій для демонстрації всього обсягу надання послуги, її результативності</td></tr>
<tr><td colspan="3">Після продажу: актуалізація, розвиток суміжних видів діяльності, підвищення зацікавості аудиторії</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: в зв’язку зі специфікою надання послуг питання про захист від копіювання полягає в захисті інтелектуальної власності.</td></tr>
</table>

Таблиця 4.10 – Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1.	20 000 - 25 000	15 000	Високий або середній	5 000 – 10 000

Таблиця 4.11 – Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*
1.	Мінімальна кількість посередників	Організація широкої мережі збуту та популяризація товару	Однорівневий канал збуту (прямий)	Залучення компаній – партнерів для організації ефективного збуту

Таблиця 4.12. – Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1.	Вимоги до повноти графу мобільного застосунку	Інформаційні мережі	Забезпечення повного покриття застосунку з створенням повного відображення на графову структуру	Донесення переваг моделі до цільових клієнтів	Рекламне повідомлення має демонструвати основні визначені переваги проекту.
2.	Вимоги до зручності використання	Інформаційні мережі	Адаптація до вимог замовника	Донесення переваг моделі до цільових клієнтів	

Висновки до розділу 4

Для оцінки доцільності виходу проекту на ринок було проведено технологічний аудит ідеї проекту, аналіз ринкових можливостей для запуску, цільовох аудиторії, потенційних клієнтів. Проведено порівняльний аналіз з конкурентами в даній та суміжних галузях на основі специфіки дяльності, та переваг.

Підсумовуючи, можна зробити висновок, що вихід на ринок буде затруднений специфікою проекту та вузькою нішею, також можливо інструмент буде відчувати протидію зі сторони основних гравців ринку. Але при поступовому введені інструменту в сферу, він завоює міцні позиції та покаже високі показники прибутку, залишаючи за собою можливості для масштабування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Elenkov N. Android Security Internals: An In-Depth Guide to Android's Security Architecture / Nikolay Elenkov. – San Francisco: no starch press, 2015. – 432 с.
2. Android Hacker's Handbook / [D. Joshua, L. Zach, C. Mulliner та ін.]., 2014. – 576 с. – (передрук).
3. Enck W. Understanding Android Security [Електронний ресурс] / William Enck. – 2009. – Режим доступу до ресурсу: https://www.computer.org/csdl/mags/sp/2009/01/msp2009010050-abs.html.
4. Knorr K. Security testing for Android mHealth apps [Електронний ресурс] / K. Knorr, D. Aspinall. – 2015. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/7107459.
5. Salva S. Data vulnerability detection by security testing for Android applications [Електронний ресурс] / S. Salva, S. Zafimiharisoa. – 2013. – Режим доступу до ресурсу: https://ieeexplore.ieee.org/abstract/document/6641043.
6. Sam M. Testing android apps through symbolic execution [Електронний ресурс] / M. Sam, N. Mirzaei. – 2012. – Режим доступу до ресурсу: https://dl.acm.org/citation.cfm?id=2382798.
7. Kowalewski S. Reverse Engineering of Mobile Application Lifecycles [Електронний ресурс] / S. Kowalewski, C. Elsemann, F. Dominik. – 2011. – Режим доступу до ресурсу: https://www.computer.org/csdl/proceedings/wcre/2011/1948/00/06079853-abs.html.

ДОДАТКИ

Додаток А. Вихідний код інструменту SmaliAnalityc

smalidroid.py

```
from parser import *
from database import *
from scanner import *
import argparse
import graphviz

class SmaliDroid(object):
    def __init__(self, args):
        self.graph = graphviz.Graph(format='svg')
        self.args = args
        self.smali_db = SmaliDatabase()
        self.smali_parser =
SmaliParser(smali_dir=self.args.smali_sources,
smali_database=self.smali_db, graph=self.graph)
        self.smali_scanner = SmaliScanner(smali_database=self.smali_db)

        if self.args.db_write:
            if not self.args.smali_sources:
                print 'no smali_sources argument. exiting.'
                return

            self.smali_db.set_db_path(self.args.db_write)
            self.smali_db.create_tables()
            self.smali_parser.parse_smali_files()

        elif self.args.db_read:
            self.smali_db.set_db_path(self.args.db_read)

        if self.args.reverse_method:
            self.build_called_from_method(self.args.reverse_method)

        if self.args.scan:
            self.scan_for_vulns()

        self.smali_db.conn.close()

    def build_called_from_method(self, method):
        # TODO merge with build_reverse_flowchart
        self.smali_parser.get_method_callers(method)
        self.graph.render(filename='img/%s' %
(self.args.reverse_method))
        self.graph.view()

    def scan_for_vulns(self):
        self.smali_scanner.scan_for_all()

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    # TODO change the parameters so they would make some sense
    parser.add_argument('smali_sources', nargs='?',
```

```
                        help="The directory path of the Smali sources to
be examined")
    parser.add_argument('-w', dest='db_write',
                        help='Write an SQLite file to this destination,
can not be used with -r option')
    parser.add_argument('-r', dest='db_read',
                        help='Read an existing SQLite file from this
destination, overrides -w option')
    parser.add_argument('-s', '--scan', action='store_true',
                        help='Scans the Smali code for possible
vulnerabilities')
    parser.add_argument('--reversemethod', dest='reverse_method',
                        help='List a reverse flowchart until the method
is ran')
    args = parser.parse_args()

smali_droid = SmaliDroid(args)
```

scanner.py

```
class SmaliScanner(object):
    def __init__(self, smali_database):
        self.db = smali_database

    def scan_for_all(self):
        print 'Scanning for vulnerabilities'
        self.scan_for_directory_traversal()
        self.scan_for_ssl_mitm()

    def scan_for_directory_traversal(self):
        print 'Scanning for possible directory traversals'
        calling_to_rows = self.db.get_calling_to_with_pattern('->unzip')

        for calling_to_row in calling_to_rows:
            unzip_sum = 0
            getabspath_sum = 0
            try:
                calling_to_methods = calling_to_row[1].split(',')
                for index, calling_to_method in
enumerate(calling_to_methods):
                    if '->unzip' in calling_to_method.lower():
                        print '%s method could be has unzip call, please
check for directory traversal' % (calling_to_row[0])
                        unzip_sum = unzip_sum + 1
                    if '->getabsolutepath' in calling_to_method.lower():
                        getabspath_sum = getabspath_sum + 1
            except IndexError:
                pass

    def scan_for_ssl_mitm(self):
        print 'Scanning for ssl cerrificate check override'
        id_rows = self.db.get_id_with_pattern('->checkClientTrusted')
        for id_row in id_rows:
            try:
                print '%s method might be vuln to SSL mitm due to
overriding checkClientTrusted' % (id_row[0])
```

```
            except IndexError:
                pass

        id_rows = self.db.get_id_with_pattern('->checkServerTrusted')
        for id_row in id_rows:
            try:
                print '%s method might be vuln to SSL mitm due to
overriding checkServerTrusted' % (id_row[0])
            except IndexError:
                pass
```

database.py

```
import sqlite3

class SmaliDatabase(object):
    def __init__(self):
        pass

    def set_db_path(self, db_path):
        self.conn = sqlite3.connect(db_path)
        self.cursor = self.conn.cursor()

    def _create_method_table(self):
        # TODO remove redundant called_from creation and usage
        self.cursor.execute('''CREATE TABLE methods
                               (id, class_name, method_name, parameters,
return_value, calling_to, called_from, data)''')

    def create_tables(self):
        self._create_method_table()

    def add_method(self, id, class_name, method_name, parameters,
return_value, calling_to='', called_from='', data=''):
        self.cursor.execute('''INSERT INTO methods
                               VALUES (?, ?, ?, ?, ?, ?, ?, ?)''', (id,

class_name,

method_name,

parameters,

return_value,

calling_to,

called_from,

data))

    def update_method_called_from(self, id, value):
        """
        params: id = the called_from value to be appended
                value = the id of the row to be appended to
        updates the row with id='value' with appending the column
```

```
'called_from' with 'id' value
        """

        self.cursor.execute('''UPDATE methods SET called_from =
                                (SELECT called_from FROM methods WHERE id
IN (?))
                                || ? || "," WHERE id IN (?)''', (value,
                                                                 id,
                                                                 value))

    def get_calling_to(self):
        self.cursor.execute('SELECT id, calling_to FROM methods')
        return self.cursor.fetchall()

    def get_called_from_method(self, method):
        self.cursor.execute('''SELECT id FROM methods where calling_to
LIKE "%" || ? || "%"''', (method,))
        return self.cursor.fetchall()

    def get_calling_to_with_pattern(self, pattern):
        self.cursor.execute('''SELECT id, calling_to FROM methods WHERE
calling_to LIKE "%" || ? || "%"''', (pattern,))
        return self.cursor.fetchall()

    def get_id_with_pattern(self, pattern):
        self.cursor.execute('''SELECT id FROM methods WHERE id LIKE "%"
|| ? || "%"''', (pattern,))
        return self.cursor.fetchall()

    def get_method_data(self, id):
        self.cursor.execute('''SELECT data FROM methods WHERE id IN
(?)''', (id,))
return self.cursor.fetchall()
```

parser.py

```
import re
import os
import fnmatch
from py2neo import Graph, Path, Node, Relationship, NodeMatcher
graph = Graph("http://neo4j:password@localhost:7474")

class SmaliParser(object):
    def __init__(self, smali_dir, smali_database, graph):
        """
        constructor for SmaliParser object
        :param smali_dir: directory of the smali files to be parsed
        :param smali_database: instance of SmaliDatabase object
        :param graph: instance of graphviz object
        """
        self.db = smali_database
        self.dir = smali_dir
        self.graph = graph

        self.pattern_class_name = re.compile(ur'^\.class.*\
(.+(?=\;))', re.MULTILINE)
```

```
        self.pattern_method_data = re.compile(ur'^\.method.+?\
(.+?(?=\())\((.*?)\)(.*?$)(.*?(?=\.end\ method))', re.MULTILINE |
re.DOTALL)
        self.pattern_called_methods = re.compile(ur'invoke-.*?\
{(.*?)}, (.+?(?=;))\;\-\>(.+?(?=\())\((.*?)\)(.*?)(?=$|;)',
re.MULTILINE | re.DOTALL)
        self.pattern_move_result = re.compile(ur'move-
result.+?(.*?)$', re.MULTILINE | re.DOTALL)

    def parse_smali_files(self):
        """
        parses all smali files in the
        :return: null
        """
        print 'Parsing smali files'
        for root, dirnames, filenames in os.walk(self.dir):
            for filename in fnmatch.filter(filenames, '*.smali'):
                filepath = os.path.join(root, filename)

                with open(filepath, 'r') as smali_file:
                    content = smali_file.read()
                    self.parse_smali_file(content)

        self.db.conn.commit()

    def parse_smali_file(self, content):
        """
        parses a single smali file and inserts the data to the smali
database
        :param content: content of the smali file
        :return: null
        """
        # TODO add the regex to parse the called function and change
the FOR loop
        # TODO change code to use all the new methods
        # parse class
        class_name = self.get_class_name(content=content)
        methods = self.get_methods(content=content)
        # add methods to db
        for method in methods:
            method_name = method[0].split(' ')[-1]
            method_parameters = method[1]
            method_return_value = method[2]
            method_data = method[3]
            method_id = '%s->%s' % (class_name, method_name)
            called_methods =
self.get_called_methods(content=method_data)
            method_calling_to = ''

            # Neo4j
            matcher = NodeMatcher(graph)
            bma = matcher.match("Method", name=method_id).first()
            if bma is None:
                tx = graph.begin()
                a = Node("Method", name=method_id)
                tx.create(a)
                tx.commit()
```

```
            # Neo4j

            for called_method in called_methods:
            # Neo4j
                name = '%s->%s' % (called_method[1], called_method[2])
                sma = matcher.match("Method", name=name).first()
                if sma is None:
                    tx = graph.begin()
                    b = Node("Method", name=name)
                    tx.create(b)
                    tx.commit()
                sma = matcher.match("Method", name=name).first()
                bma = matcher.match("Method", name=method_id).first()
                tx = graph.begin()
                mab = Relationship(bma, "Call", sma)
                tx.create(mab)
                tx.commit()
            # Neo4j
                method_calling_to = '%s%s->%s,' % (method_calling_to,
called_method[1], called_method[2])

            self.db.add_method(id=method_id, class_name=class_name,
method_name=method_name, parameters=method_parameters,
calling_to=method_calling_to, return_value=method_return_value,
data=method_data)
            '''
            matcher = NodeMatcher(graph)
            name = "init"
            ma = matcher.match("Method", name=name).first()
            name = "Test"
            mb = matcher.match("Method", name=name).first()
            print(ma)
            print(mb)
            tx = graph.begin()
            mab = Relationship(ma, "Call", mb)
            tx.create(mab)
            tx.commit()
            '''

    def get_class_name(self, content):
        """
        gets the class name of a single smali file content
        :param content: smali file content
        :rtype: string
        :return: the name of the class
        """
        data = re.findall(self.pattern_class_name, content)
        return data[0]

    def get_methods(self, content):
        """
        gets all methods in a single smali file content
        :param content: smali file content
        :rtype: list of lists
        :return: [0] - method name
                 [1] - method parameters
                 [2] - method return value
```

```
                [3] - method data
        """
        data = re.findall(self.pattern_method_data, content)
        return data

    def get_called_methods(self, content):
        """
        gets all the method called inside a smali method data. works
just fine with a single smali line
        :param content: content of the smali data to be parsed
        :rtype: list of lists
        :return: [0] - called method parameters
                 [1] - called method object type
                 [2] - called method name
                 [3] - called method parameters object type
                 [4] - called method return object type
        """
        data = re.findall(self.pattern_called_methods, content)
        return data

    def get_result_object_name(self, content):
        data = re.findall(self.pattern_move_result, content)
        return data[0].split(' ')[-1]

    def get_method_callers(self, method):
        """
        recursive method for getting the callers for the method
        :param method:
        :return: null
        """
        # TODO move the db query to database.py
        print 'Creating called from column for method %s' % (method)
        #self.db.cursor.execute('''SELECT DISTINCT id, calling_to FROM
methods where calling_to LIKE "%" || ? || "%"''', (method,))
        data = self.db.get_called_from_method(method)

        if data == '':
            return
        for id_row in data:
            if id_row[0] == method:
                return
            edge_connection1 = '\t\t"%s" -- "%s"' % (method,
id_row[0])
            edge_connection2 = '\t\t"%s" -- "%s"' % (id_row[0],
method)
            if edge_connection1 in self.graph.body:
                continue
            if edge_connection2 in self.graph.body:
                continue
            self.graph.edge(method, id_row[0])
            #self.db.update_method_called_from(id=id_row[0],
value=method)
            self.get_method_callers(id_row[0])

    def analyze_method_pattern_flow(self, content, pattern):

        smali_lines = filter(None, content.splitline())
```

```
        # return a list with all line positions of smali lines that
contain the pattern
        pattern_lines_indexes = [i for i, smali_line in
enumerate(smali_lines) if pattern in smali_line]

        for pattern_line_index in pattern_lines_indexes:
            line = smali_lines[pattern_line_index]
            data = self.get_called_methods(line)
            method_name = data[2]

            if not method_name == pattern:
                return
            expected_result_object_line =
smali_lines[pattern_line_index + 1]

            if not 'move-result' in expected_result_object_line:
                print 'ERROR: move-result opcode not found in next
line. line no: %s' % (str(pattern_line_index + 1))
                return

            result_object_name =
self.get_result_object_name(expected_result_object_line)
            # return a list with all line positions of smali lines
that contain the result object name
            object_name_indexes = [i for i, smali_line in
enumerate(smali_lines) if result_object_name in smali_line]
            for object_name_index in object_name_indexes:
                # TODO check if the object name is called elsewhere
(optimally after the move-result opcode)
pass
```