МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

«До захисту допущена»
Зав. кафедри комп'ютерних наук
та прикладної математики
д.т.н., професор Турбал Ю.В.
«____» _____ 20_ р.

КВАЛІФІКАЦІЙНА РОБОТА

" МЕТОДИ ТА АЛГОРИТМИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ APK-ФАЙЛІВ "

Виконав:

Студент навчально-наукового інституту кібернетики, інформаційних технологій та інженерії

Група КН41
Бояр Олександр Петрович _____
підпис

Керівник: Турбал М.Ю. _____
підпис

Рівне-2024

РЕФЕРАТ

Об’єкт дослідження: сучасні технології захисту інформації.

Предмет дослідження: методи аналізу та захисту APK-файлів на основі нейронних мереж.

Мета роботи: дослідити питання безпеки Android-додатків та розробити нейронну мережу для дослідження вразливостей APK-файлів.

Актуальність. Актуальність дослідження методів та алгоритмів виявлення вразливостей APK-файлів обумовлена стрімким зростанням використання мобільних додатків на платформі Android, яка є однією з найбільш поширених операційних систем у світі. З кожним роком кількість додатків та їхніх користувачів неухильно збільшується, що водночас підвищує ризики для інформаційної безпеки. Вразливості у мобільних додатках можуть призвести до витоку конфіденційної інформації, фінансових втрат та інших значних негативних наслідків для користувачів і організацій.

Розробка ефективних методів виявлення вразливостей у APK-файлах є критично важливою для забезпечення надійного захисту даних та стабільної роботи мобільних додатків.

Ключові слова: Android, APK, нейронна мережа, база даних, безпека, вразливості.

Національний університет водного господарства та природокористування

(повне найменування вищого навчального закладу)

НН інститут *автоматики, кібернетики та обчислювальної техніки*

Кафедра *Комп'ютерних наук та прикладної математики*

Освітньо кваліфікаційний рівень *Бакалавр*

Спеціальність *112 Комп'ютерні науки*

(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____

_____

"____"_____20___року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Бояру Олександр Петрович

Тема роботи «Методи та алгоритми виявлення вразливостей APK-файлів»

Керівник роботи: доктор філософії Турбал М.Ю.

затверджені наказом вищого навчального закладу № С449 від 19.04.2024

2. **Термін здачі студентом закінченої роботи:** 22 червня 2024 року

3. **Вихідні дані до роботи:** літературні джерела з питань розробки сайтів на основі фреймворку Django.

4. **Зміст розрахунково-пояснювальної записки:** дослідження сучасних підходів до проектування WEB-додатків та розробка додатку для продажу питної води на основі фреймворку Django.

5. **Перелік графічного матеріалу:** мультимедійна презентація.

6. **Консультанти розділів роботи**

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1	*проф. Турбал М.Ю.*	*3.10.23*	*3.10.23*
Розділ 2	*проф. Турбал М.Ю.*	*18.11.23*	*18.11.23*
Розділ 3	*проф. Турбал М.Ю.*	*14.01.24*	*14.01.24*
Розділ 4	*проф. Турбал М.Ю.*	*10.03.24*	*10.03.24*

7. **Дата видачі завдання** *01 жовтня 2022 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Вивчення літератури за обраною тематикою*	*3.10.23 -14.10. 23*	*виконав*
2	*Формулювання завдання*	*15.10. 23-28.10. 23*	*виконав*
3	*Розробка алгоритму розв'язку поставленого завдання*	*28.10. 23-14.01. 23*	*виконав*
4	*Здійснення програмної реалізації*	*15.01. 24-15.02.24*	*виконав*
5	*Тестування програми*	*16.02.24-9.04.24*	*виконав*
6	*Аналіз отриманих результатів*	*10.04.24-24.04.24*	*виконав*
7	*Загальні висновки до роботи*	*24.04.24-5.05.24*	*виконав*
8	*Підготовка звіту кваліфікаційної роботи*	*13.05.24-23.05.24*	*виконав*
9	*Підготовка мультимедійної презентації*	*23.05.24-10.06.24*	*виконав*
10	*Підготовка до захисту*	*11.06.24-18.06.24*	*виконав*

Студент _____ _____

(підпис) (прізвище та ініціали)

Керівник роботи _____ _____

(підпис) (прізвище та ініціали)

Зміст

Вступ

Актуальність дослідження методів та алгоритмів виявлення вразливостей APK-файлів обумовлена стрімким зростанням використання мобільних додатків на платформі Android, яка є однією з найбільш поширених операційних систем у світі. З кожним роком кількість додатків та їхніх користувачів неухильно збільшується, що водночас підвищує ризики для інформаційної безпеки. Вразливості у мобільних додатках можуть призвести до витоку конфіденційної інформації, фінансових втрат та інших значних негативних наслідків для користувачів і організацій.

Розробка ефективних методів виявлення вразливостей у APK-файлах є критично важливою для забезпечення надійного захисту даних та стабільної роботи мобільних додатків. Важливим аспектом є також розвиток інструментів та технік для реверс-інжинірингу, які дозволяють аналізувати внутрішню структуру додатків, виявляти слабкі місця та пропонувати шляхи їх усунення. Це дослідження сприяє підвищенню загального рівня безпеки мобільних додатків, що є важливим завданням у сучасному цифровому середовищі.

Розробка і впровадження нових підходів до аналізу та захисту APK-файлів має вагоме значення для забезпечення безпеки мобільних користувачів і підтримки цілісності інформаційних систем, які все більше інтегруються у повсякденне життя.

Об’єкт дослідження: сучасні технології захисту інформації.

Предмет дослідження: методи аналізу та захисту APK-файлів на ос нові нейронних мереж.

Мета роботи: дослідити питання безпеки Android-додатків та розробити нейронну мережу для дослідження вразливостей APK-файлів.

Розділ1. Огляд відомих технологій та результатів

1.1 Аналіз дослідженості теми

Історія розвитку систем виявлення шкідливих додатків для платформи Android пройшла значний шлях з моменту появи цієї операційної системи. Сьогодні ці системи представляють собою складні та багатофункціональні інструменти, які поєднують різноманітні методи аналізу та технології машинного навчання для забезпечення максимальної безпеки користувачів.

З 2013 року кількість шкідливих додатків значно зросла, що вимагало розробки нових, більш ефективних методів їх виявлення. У цей період з'явилися перші динамічні методи аналізу, які дозволяли виявляти шкідливі додатки на основі їх поведінки під час виконання. Такі системи, як DroidMat і DREBIN, використовували машинне навчання для класифікації додатків на основі їх характеристик та поведінки.

З 2016 року розпочалася активна розробка гібридних методів, які поєднували в собі статичний та динамічний аналіз. Це дозволило значно підвищити ефективність виявлення шкідливих додатків. Водночас, розвиток методів машинного навчання, таких як метод опорних векторів (SVM), кластеризація k-NN та інші, значно підвищили точність та швидкість класифікації додатків.

На сучасному етапі системи виявлення шкідливих додатків стали ще більш складними та ефективними. Вони активно використовують штучний інтелект та великі дані для аналізу додатків та виявлення шкідливих програм. Однак, з розвитком технологій, шкідливі програми також стають більш витонченими, що вимагає постійного вдосконалення методів захисту.

Сьогодні такі системи, як Google Play Protect, надають користувачам базовий рівень захисту, але потребують постійного вдосконалення. Наприклад, системи типу AppContext та інші використовують контекстний аналіз для підвищення точності виявлення шкідливих програм.

Історичний розвиток систем виявлення шкідливих додатків для Android демонструє, як швидко еволюціонують загрози та методи їх виявлення. Починаючи з простих статичних аналізів, технології перейшли до складних гібридних методів з використанням машинного навчання та штучного інтелекту. Для забезпечення безпеки користувачів необхідно продовжувати дослідження та розробки в цій сфері, адаптуючись до нових викликів та загроз.

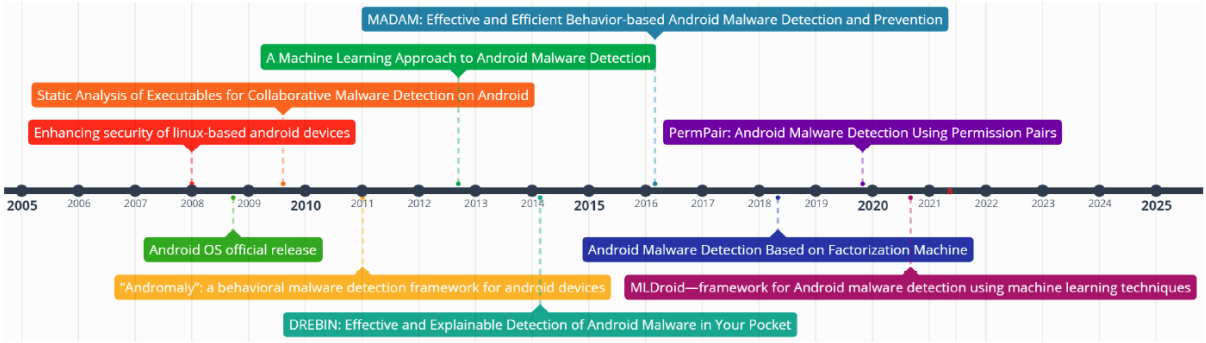


Рис. 1.1 Історичний розвиток теми виявлення шкідливих додатків для Android

Питання безпеки операційної системи Android стали об'єктом уваги дослідників ще до її офіційного випуску. У січні 2008 року була представлена перша робота з виявлення шкідливих додатків [22], розвиток якої продовжився у 2009 році [23], коли з'явилася можливість оцінити перші офіційні версії платформи Android. Ці роботи були написані Обри-Дерриком Шмідтом і групою з семи осіб з Берліна та Стамбула [22, 23].

Перші дослідження з виявлення шкідливих додатків для Android базувалися на попередніх дослідженнях, присвячених іншим мобільним системам і пристроям. Основною ідеєю було використання сервера, який здійснював аналіз безпеки додатків. Кожен додаток, встановлений на пристрій, повинен був пересилатися на сервер компанії, що постачає операційну систему, для аналізу.

У роботах [22, 23] пропонувався аналогічний підхід з вдосконаленням для системи Android. Основною ідеєю було забезпечити всі пристрої модулем аналізу додатків, який базувався на алгоритмах PART, Prism і k-найближчих сусідів (k-NN).

Цей підхід передбачав миттєву перевірку додатків і розсилку всім пристроям інформації про їх безпеку.

На сьогоднішній день існує три основні методи виявлення шкідливих додатків. Статичний метод базується на аналізі APK-файлу додатка без врахування його поведінки під час виконання. В роботах [8] і [9] розглядаються методи статичного виявлення, засновані на характеристиках безпеки додатків і графах передачі управління, а в [10] – конкретно на пошуку сигнатур. Однак ці системи практично непридатні до реальних систем через складність їх автоматизації і не найкращі результати виявлення. Динамічний метод заснований на аналізі поведінки додатка під час його виконання. Додаток запускається в безпечному середовищі (пісочниці), і на основі аналізу викликів API, системних викликів, звернень до пам'яті та мережі виявляються шкідливі дії додатка. Гібридний метод об'єднує статичний і динамічний методи, приймаючи рішення про безпеку додатка на основі обох типів даних. У сучасних системах виявлення шкідливих додатків використовується саме цей метод. Наприклад, у [11] до статичного і динамічного методів додається кластеризація, що дозволяє підвищити ефективність системи.

1.2 Принципи роботи систем виявлення і класифікації шкідливих додатків

Системи виявлення і класифікації шкідливих додатків зазвичай працюють на основі наступних принципів.

Сигнатурне виявлення - працює на основі порівняння бази раніше виявлених сигнатур шкідливих додатків з вихідним кодом аналізованого додатка. Недоліком є те, що ці системи не працюють при шифруванні вихідного коду.

Обчислення умовних ймовірностей - передбачає попереднє обчислення ймовірностей появи характеристик безпеки в вихідних кодах і маніфестах для шкідливих і безпечних додатків.

Машинне навчання - найбільш популярний і точний метод на сьогоднішній день. Включає такі алгоритми, як SVM, Random Forest, k-NN і Naïve Bayes.

Прикладом є система DroidMat, яка виконує статичний аналіз файлу маніфесту і вихідного коду додатка для виявлення характеристик безпеки і використовує k-means і k-NN для класифікації шкідливих додатків.

Сучасні системи виявлення шкідливих додатків активно використовують штучний інтелект і великі дані для аналізу додатків і виявлення шкідливих програм. Системи, такі як Google Play Protect, надають базовий рівень захисту, але потребують постійного вдосконалення. Системи, такі як AppContext і FlowDroid, застосовують контекстний аналіз і побудову графів потоків для підвищення точності виявлення шкідливих додатків. Однак ці системи все ще стикаються з викликами, пов'язаними з точністю побудови міжкомпонентних графів управління.

У роботі [23] для розширення можливостей забезпечення безпеки системи пропонується використання утиліт Linux: антивірусу, firewall, детекторів руткітів та інших інструментів, що застосовуються для аналізу, виявлення та запобігання ризикам безпеки на системному рівні. Проте, ці підходи мали кілька суттєвих недоліків. Зокрема, наступні: використання сторонніх серверів, які потребують постійної підтримки, витрати ресурсів мобільних пристроїв - включаючи заряд батареї, проблеми з передачею даних , розширення вбудованої системи та підтримка роботи сторонніх утиліт.

Згодом була опублікована робота Томаса Блізінга та інших представників німецького університету [24], де для динамічного аналізу було запропоновано використання пісочниць, в яких працюють Android-додатки. У цій роботі запропонований гібридний метод виявлення: під час статичного аналізу виявляються і логуються підозрілі та потенційно небезпечні функції і дозволи, а під час динамічного аналізу підраховується і логуються кількість системних викликів за час роботи додатка.

Недоліки запропонованої системивключають наступні: додаткове навантаження на пристрій під час гібридного аналізу, логування інформації без

негайного реагування, суб'єктивність інформації для логування -припускається, що вона може вказувати на шкідливу поведінку, але все залежить від критеріїв оцінки.

Хоча запропоновані вдосконалення в роботах [22], [23], [24] мали свої недоліки, вони заклали основу для подальших досліджень у галузі виявлення шкідливих додатків для Android. Сучасні системи виявлення активно розвиваються, використовуючи досягнення в області машинного навчання та аналізу великих даних, що дозволяє підвищити точність і ефективність виявлення шкідливих додатків.

1.3 Проблема безпеки мобільних пристроїв

В контексті інформаційної безпеки мобільні пристрої потребують індивідуального підходу до аналізу їх складових елементів. У даному випадку розглядається окремий компонент мобільних пристроїв - операційна система Android, зокрема - Android-додатки. Особливий інтерес до Android-пристроїв інформаційної безпеки зумовлений їх широким поширенням серед мобільних пристроїв. На сьогодні, платформа Android завоювала більшість ринку мобільних пристроїв і утримує позиції з вагомою перевагою - понад 70% від ринку мобільних пристроїв.

Основну частину систем Android і iOS складають додатки - від ігор до системних додатків. Однак, на відміну від iOS, яка дозволяє встановлювати додатки лише зі свого контрольованого магазину App Store, додатки на платформу Android можуть бути встановлені як з довіренного джерела - магазину Google Play, так і з будь-яких інших джерел - інших магазинів, сайтів. У такому випадку перед встановленням користувачеві буде показано попередження про можливість небезпечної установки додатків з недовірених джерел і буде запропоновано дозволити встановлення з таких джерел за допомогою відповідного налаштування. Після встановлення цього налаштування будь-який додаток, взятий з будь-якого джерела, може бути встановлено без додаткових підтверджень.

Така "відкритість" системи Android виявляється не лише в установці додатків, але й у інших аспектах її роботи, наприклад, у наданні дозволів додаткам. Іншою особливістю платформи є її програмна відкритість - будь-який розробник може вносити зміни в систему. Крім того, систему Android доробляють виробники пристроїв, які також можуть вносити помилки разом зі змінами платформи.

Магазин додатків Google Play містить величезну кількість програм. Згідно з [4], кількість програм продовжує активно зростати і досягає майже чотирьох мільйонів додатків.

Ефективна фільтрація та перевірка додатків на безпеку вимагає високоточної системи, але назвати Google Play Protect такою дуже складно через регулярне виявлення шкідливих додатків в магазині Google Play. Наприклад, найбільш значними є виявлення 9 шкідливих додатків у кінці січня 2021 року, які ставили користувачів під ризик вразливості віддаленого доступу та шкідливого ПЗ для банківських операцій, обходячи двофакторну автентифікацію банківських додатків. Присутність в Google Play такого серйозного шкідливого ПЗ свідчить про недостатню практичність та надійність систем виявлення шкідливих додатків компанії Google.

У зв'язку з цим потрібне постійне вдосконалення старих та розробка нових систем забезпечення безпеки. Одним з класів таких систем є системи виявлення шкідливих додатків, які розглядаються в цій кваліфікаційній роботі. Крім їх розгляду, у наступних розділах досліджується ефективність та застосовність систем, що базуються на методі факторизації матриць.

1.4 Виявлення та класифікація шкідливих додатків

Темі виявлення та класифікації шкідливих додатків приділяється безліч робіт, статей та звітів. Системи виявлення шкідливих додатків широко поширені і входять до складу антивірусних засобів, а також використовуються у великих сервісах, що взаємодіють з додатками сторонніх розробників, наприклад, Google Play. Тому такі

системи мають велике значення як для дослідників, так і для компаній, що їх використовують.

Зазвичай виділяють наступні методи виявлення шкідливих додатків: статичний, динамічний, гібридний.

Надалі будуть докладно розглянуті ці методи та відповідні роботи для системи Android. Статичний метод виявлення базується на аналізі APK-файлу додатка і не враховує його поведінку під час роботи. Шляхом розбору вихідних кодів та маніфесту додатка виділяються значущі характеристики безпеки, або здійснюється пошук шкідливого коду, на основі чого робиться висновок про безпеку додатка. Наприклад, у роботах [8] та [9] розглядаються методи статичного виявлення, які базуються на характеристиках безпеки додатків та графах передачі управління, а в [10] - конкретно на пошуку сигнатур. Однак такі системи практично непридатні для реальних систем через складність їх автоматизації та не найкращі результати виявлення.

Динамічний метод ґрунтується на аналізі поведінки додатка. Додаток запускається в пісочниці для того, щоб запобігти негативним наслідкам, і на основі аналізу викликів додатком сторонніх API, системних викликів, доступу до пам'яті та мережі виявляє зловмисні дії додатка. Гібридний метод поєднує статичні та динамічні методи і робить висновок про безпеку додатка як на основі статичних, так і на основі динамічних даних. У більшості сучасних систем виявлення шкідливих додатків саме цей метод використовується. Прикладом такого методу є [11], де до статичних та динамічних методів додається ще й кластеризація. Система показала достатньо високу ефективність.

В основному, системи виявлення та класифікації працюють за такими принципами:

Сигнатурне виявлення. Цей метод ґрунтується на порівнянні бази раніше виявлених сигнатур шкідливих додатків і вихідних кодів аналізованого додатка. У цьому враховуються можливості зміни сигнатур, кодування коду, але ці системи не

працюватимуть при шифруванні вихідного коду додатка. Через це на сьогодні такі системи практично не використовуються і не реалізуються.

Обчислення умовних ймовірностей. Ці системи передбачально обчислюють умовні ймовірності появи характеристик безпеки вихідних кодів та маніфесту для шкідливих і безпечних додатків. Під час аналізу додатка враховуються раніше обчислені ймовірності для кожної характеристики безпеки, і обчислюється ймовірність того, що аналізований додаток є шкідливим. Динамічні системи, які працюють за цим принципом, обчислюють умовні ймовірності критичних дій додатка і враховують їх під час аналізу.

Машинне навчання. Цей підхід використовує алгоритми машинного навчання для виявлення та класифікації шкідливих додатків. До таких алгоритмів належать: SVM (метод опорних векторів); Random Forest (випадковий ліс); kNN (метод найближчих сусідів); Naïve Bayes (наївний Баєсівський класифікатор).

Машинне навчання дозволяє системам автоматично виявляти закономірності у наборі даних та використовувати їх для класифікації нових додатків як безпечних або шкідливих. Цей підхід дозволяє покращити ефективність систем виявлення шкідливих додатків та зменшити кількість помилкових спрацювань.

Системи на основі умовних ймовірностей. Ці системи передбачають попереднє обчислення ймовірностей появи різних характеристик безпеки в вихідних кодах та маніфестах для шкідливих і безпечних додатків. Під час аналізу додатка враховуються раніше обчислені ймовірності для кожної характеристики безпеки, і розраховується ймовірність того, що аналізований додаток є шкідливим. Динамічні системи, що працюють за цим принципом, обчислюють умовні ймовірності критичних дій додатка та враховують їх під час аналізу.

Використання таких методів дозволяє покращити ефективність систем виявлення та класифікації шкідливих додатків. Кожен з наведених підходів має свої переваги та недоліки, тому важливо вибрати оптимальний метод з урахуванням конкретних вимог та обмежень задачі.

Будь-яка система виявлення та класифікації шкідливих додатків є складною інженерною задачею, яка вимагає постійного вдосконалення та адаптації до нових загроз та технологій.

Наступні кроки у дослідженні включають розгляд більш складних моделей машинного навчання, вдосконалення алгоритмів класифікації та збільшення обсягу даних для тренування. Також важливо продовжувати аналізувати нові методи та технології для покращення систем виявлення шкідливих додатків.

У сучасний час системи, які працюють на основі машинного навчання, є найбільш популярними завдяки їх високій точності виявлення та можливості класифікації шкідливих додатків. Основні виклики у цьому випадку включають: навчання систем; вибір оптимальних параметрів.

Складність навчання для різних систем представлена в Таблиці 1, де n - кількість навчальних прикладів, d - розмірність вхідного вектора.

Табл.1 Складність навчання систем виявлення та класифікації

Класифікатор	Складність
Random Forest	$O(ndlogn)$
SVM	$O(max(n,d),min((n,d)2))$
kNN	$O(ndk)$
Naïve Bayes	$O(nd)$

Складність навчання систем виявлення та класифікації може бути значною і залежить від декількох чинників, таких як обсяг та якість навчальних даних, складність моделей машинного навчання, обчислювальні витрати та інші. Наведемо основні аспекти, які впливають на складність навчання таких систем.

Обсяг та якість даних: Навчальні дані є ключовим елементом успішного навчання систем виявлення та класифікації. Недостатність даних або низька якість може призвести до низької точності моделі.

Складність моделей:Використання складних моделей машинного навчання, таких як глибокі нейронні мережі, може вимагати більшої кількості обчислювальних ресурсів і тривалого часу для навчання.

Параметризація та налаштування: Вибір оптимальних параметрів моделі, таких як гіперпараметри алгоритмів машинного навчання, може бути складним завданням, яке вимагає експертного знання та експериментів.

Обчислювальні витрати:Навчання складних моделей може вимагати значних обчислювальних ресурсів, зокрема потужних обчислювальних пристроїв або великої кількості часу на обчислення.

Відбір ознак: Важливим етапом в навчанні є відбір важливих ознак або підготовка даних для подальшого аналізу, що може вимагати додаткових знань та часу.

У цілому, складність навчання систем виявлення та класифікації є багатоаспектною і залежить від конкретного завдання, доступних ресурсів та експертної підготовки.

1.5 Огляд фреймворків для аналізу APK

Існує велика кількість реалізованих систем на основі машинного навчання, які працюють з різними методами виявлення шкідливих додатків. Деякі з таких систем описані в роботах [13] і [14].

Крім того, DroidMat [15] проводить статичний аналіз файлів маніфеста та вихідного коду додатків для платформи Android з метою витягування характеристик безпеки, таких як дозволи, використання апаратних ресурсів та виклики API. Потім він використовує метод k-means та класифікацію k найближчих сусідів (k-NN) для виявлення шкідливих програм.

DREBIN [16] також витягує аналогічні характеристики з файлів маніфеста та вихідного коду додатків, а потім використовує метод опорних векторів (SVM) для класифікації шкідливих програм на основі one-hot кодування відповідних характеристик.

У багатьох дослідженнях розглядається та реалізується пошук шаблонів шкідливої поведінки за допомогою графів потоків управління або графів викликів. Наприклад, AppContext [17] класифікує додатки, використовуючи машинне навчання на основі контекстів, які витягаються у разі виявлення підозрілої поведінки. Створюється граф викликів з вихідного коду додатка, і контекст витягається за допомогою аналізу потоків інформації всередині додатка. Потім AppContext отримує специфічні характеристики безпеки з витягнутого контексту, які потім використовуються в алгоритмах машинного навчання. У роботі [17] було проаналізовано 633 безпечні додатки з магазину Google Play та 202 зразки шкідливих програм. AppContext успішно ідентифікував 192 шкідливих додатки з точністю 87,7%.

Гаскон та співавтори у [18] також використовували графи викликів для виявлення шкідливих програм. Після витягнення графів викликів з додатків, в їхній реалізації використовується графічне ядро, і за лінійний час з графів викликів витягуються характеристики безпеки. Отримані характеристики подаються на вхід до SVM, яка в свою чергу класифікує додаток як шкідливий або безпечний. Був проведений експеримент з 136 тисячами безпечних і 12 тисячами шкідливих додатків. У рамках експерименту система правильно виявила 89% шкідливих програм з 1% ложних спрацювань. Однак недоліком цього методу є те, що він сильно залежить від точності витягнення графа викликів.

Незважаючи на відносно успішні результати зазначених досліджень, в наступній частині обговоримо, що навіть такі системи, як FlowDroid [19] та IC3 [20], все ще не можуть повністю вирішити проблему побудови міжкомпонентних потокових графів управління (ICFG), особливо потоків, створених намірами та фільтрами намірів.

Перевагою цих систем є те, що вони можуть класифікувати шкідливі додатки як за наперед визначеними класифікуючими властивостями, так і за тими, які вони самі виділять під час навчання.

Системи виявлення шкідливих додатків, засновані на машинному навчанні, мають можливість працювати як на основі результатів статичного, так і динамічного аналізу, а отже, і гібридного, що робить їх потужним, розширюваним і найефективнішим інструментом на сьогоднішній день. У зв'язку з цим такі системи активно розвиваються та вдосконалюються, а дослідники намагаються досягти їхньої найбільшої точності виявлення, застосовуючи різні методи машинного навчання.

При розгляді питання безпеки мобільних пристроїв неможливо не згадати про типи шкідливих додатків. Google виділяє 14 категорій, що стосуються Android пристроїв [21]. Серед них є як стандартні типи шкідливих додатків: використовують чорні ходи; спричиняють відмову в обслуговуванні; завантажувачі; проводять фішинг; підвищують привілеї; вимагачі; розсилають спам; шпигунські програми; троянські програми.

Так і типи, які стосуються конкретно платформи або мобільних пристроїв взагалі: програми-шахрайки з виставленням рахунків - це тип шахрайства з відправленням платних SMS-повідомлень, здійсненням дорогих дзвінків та підпискою на платні телефонні послуги; комерційне шпигунське ПЗ - призначені для слідкування за пристроєм, передачі особистої інформації з пристрою без повідомлення та згоди користувача. програми, які виконують рутування, тобто отримання пристроєм прав суперкористувача root без попередження; додатки, які містять нестандартну для Android пристроїв шкідливу функціональність, а також загрози для інших платформ.

Розділ 2. Безпека Android-додатків

2.1 Характеристики безпеки в Android-додатках

2.1.1 Послідовність аналізу

Характеристики безпеки Android-додатків є такими компонентами програм, які впливають на безпеку пристроїв Android і можуть бути використані зловмисниками для порушення інформаційної безпеки пристрою. Це може включати крадіжку конфіденційної інформації, отримання віддаленого доступу до пристрою, а також завдання шкоди як самому пристрою, так і його програмним компонентам.

Обробка Android-додатка для виділення характеристик безпеки включає кілька ключових кроків.

Розпаковка і декомпіляція вихідних кодів додатка. Першим кроком є отримання вихідного коду додатка. Це включає розпаковку APK файлу та декомпіляцію коду, щоб отримати доступ до вихідних текстів і ресурсів додатка.

Виділення характеристик безпеки з файлу маніфеста та декомпільованих кодів. Файл маніфеста (AndroidManifest.xml) містить важливі метадані про додаток, включаючи дозволи, які він запитує, та інші конфігураційні дані. Виділення характеристик безпеки з цього файлу дозволяє визначити, які дії може виконувати додаток.

Аналіз декомпільованого коду дозволяє виявити потенційно небезпечні виклики API, використання небезпечних функцій, та інші аспекти, які можуть свідчити про шкідливу активність.

Кодування характеристик безпеки у вхідний вектор для навчання і тестування нейромережевої моделі. Виділені характеристики безпеки кодуються у вектор, який може бути використаний як вхід для моделей машинного навчання. Ці моделі тренуються для розпізнавання шкідливих додатків на основі аналізу виділених характеристик.

Ці кроки є важливими для створення системи, яка ефективно виявляє шкідливі додатки, забезпечуючи таким чином безпеку користувачів та їхніх пристроїв.

2.1.2 Класифікація характеристик безпеки

Характеристики безпеки Android-додатків – це такі компоненти програм, які впливають на безпеку пристроїв Android і можуть бути використані зловмисниками для порушення інформаційної безпеки пристрою. Це може включати крадіжку конфіденційної інформації, отримання віддаленого доступу до пристрою, а також завдання шкоди як самому пристрою, так і його програмним компонентам.

Характеристики безпеки додатків можна розділити на кілька категорій залежно від їхніх функцій. Можна виділити наступні характеристики.

Список компонентів додатка - компоненти декларуються у файлі маніфеста додатка і визначають різні інтерфейси для взаємодії з кінцевим користувачем та ОС Android. Імена цих компонентів використовуються для ідентифікації відомих шкідливих додатків за їхніми компонентами.

Використовувані апаратні засоби - якщо додаток потребує доступу до таких апаратних засобів пристрою, як GPS, камера або сенсори, ці характеристики повинні бути оголошені у файлі маніфеста додатка. Надання прав доступу до певного набору апаратних засобів може вказувати на можливість витоку чутливих даних. Наприклад, задекларований доступ до GPS і мережі може свідчити про передачу зловмисникам місцеположення пристрою.

Дозволи додатка - Android використовує механізм дозволів для контролю та збереження конфіденційності користувача. Дозволи використовуються для надання прав додатку до чутливих даних, компонентів апаратних засобів і доступу до сторонніх бібліотек і інтерфейсів взаємодії додатків. Шкідливі додатки використовують особливі набори дозволів за тим же принципом, що й апаратні засоби. Крім того, існує набір як звичайних дозволів, так і небезпечних. Їхня відмінність полягає в тому, що небезпечні дозволи запитуються у користувача

безпосередньо під час роботи додатка, при цьому попереджаючи його про небезпеку надання таких дозволів. Більшість шкідливих програм базуються саме на основі великої кількості запитуваних небезпечних дозволів, тому вони враховуються в більшості систем виявлення і класифікації шкідливих додатків.

Фільтри намірів (intent filter) - Intent – намір додатка зв'язатися з іншим додатком або компонентом для отримання або передачі якої-небудь інформації. Це основний спосіб “спілкування” різних додатків, як системних, так і прикладних. Фільтр намірів використовується для того, щоб відсівати непотрібні для додатка наміри від інших додатків і компонентів та приймати наміри, що проходять по критеріям запитуваних дій, категорій намірів і передаваних даних. Як правило, шкідливі додатки здійснюють фільтрацію намірів для отримання тільки чутливої і дійсно необхідної інформації.

Крім стандартних категорій характеристик безпеки, які можна отримати з маніфеста додатка, існують особливі характеристики, що допомагають більш точно визначити додаток як шкідливий і містяться безпосередньо у коді додатка.

Підписки на широкомовні повідомлення (broadcast receiver) - широкомовні повідомлення, аналогічно намірам, містять інформацію, що розсилається додаткам і компонентам. Однак, на відміну від намірів, вони спрямовані не на конкретний додаток, а на всі додатки, підписані на відповідні повідомлення. Якщо додаток, якому спрямоване широкомовне повідомлення, не запущений, то воно запускається, обробляє повідомлення і в залежності від результату обробки або завершується, або продовжує роботу, запускаючи відповідну активність або сервіс. Прикладом широкомовного повідомлення може бути повідомлення від контролера батареї про низький заряд батареї пристрою. Відповідно, шкідливий додаток, підписуючись на широкомовні повідомлення, може контролювати стан пристрою, додатків та їх компонентів.

Використовувані постачальники контенту (content provider) - постачальники контенту є загальними базами даних, наданими додатками для інших додатків.

Прикладами постачальників контенту можуть бути браузери, контакти, медіа-додатки, налаштування. Відповідно, постачальники можуть зберігати чутливу інформацію, яка є метою для зловмисників, тому шкідливі додатки нерідко запитують дані у постачальників.

Вбудовані API - система Android надає список стандартних API для роботи з чутливими даними, доступ до яких надається за допомогою дозволів. Якщо виклик цих API здійснюється без запиту дозволу, це може свідчити про те, що додаток є root експлойтом. Крім того, виклик функцій API, що працюють з чутливими даними, також може вказувати на шкідливість додатка.

Сторонні API - при використанні сторонніх API деякі функції можуть виконуватися без надання відповідних дозволів з різних причин. Знаючи про існування таких небезпечних API і виявляючи їх у додатках, можна припустити, що такі додатки є шкідливими.

Дозволи, запитувані в коді додатка - виділяючи не тільки запитувані дозволи з файлу маніфеста, але й конкретно використовувані при виклику API та їх відповідних функцій, можна доповнити список дозволів, оскільки надання прав може здійснюватися не тільки з файлу маніфеста, але й з коду додатка.

Граф потоку управління - виконання кожного додатка відбувається послідовно – функція за функцією. Таким чином, можна створити граф, що відображає послідовність викликів функцій додатка. При його візуалізації стає зрозуміло, як працює додаток, який його алгоритм. Однак в Android-додатках може бути не одна точка входу, а багато – додаток може запускатися шляхом його відкриття користувачем, може виконувати завдання у вигляді сервісу, може відкриватися і здійснювати дії при отриманні широкомовних повідомлень тощо. Кількість точок входу залежить від кількості оголошених інтерфейсів взаємодії в додатку. З точки зору виявлення шкідливих додатків стає важливо сформувати послідовність викликів API функцій, оскільки одні й ті ж API можуть однаково

часто з’являтися як у шкідливих, так і в добросовісних додатках, однак порядок їх виклику може вказати на можливі шкідливі дії додатка.

Динамічні характеристики - витягуються під час динамічного аналізу. Динамічні характеристики дозволяють виявити аномалії, відхилення та підозрілу або шкідливу поведінку під час роботи додатків. Такими характеристиками можуть бути ресурси, що споживаються додатком, частота і адреси звернень до мережі Інтернет, а також передавані при цьому дані, відправка і прийом SMS, виконання криптографічних операцій, підвантаження DEX модулів і багато інших операцій, які можуть вказувати на шкідливість додатка.

2.1.3 Ефективні методи пошуку

Інсталяційний APK файл програми є стиснутим архівом, що містить вихідний код програми, файли ресурсів, метадані, додаткові бібліотеки та файл маніфесту програми. Вихідний код закодований у файл з розширенням DEX (Dalvik Executable), який інтерпретується та виконується віртуальною машиною Dalvik, яка згодом замінена на ART (Android Runtime), що виконує аналогічні функції. Dalvik та ART є середовищем виконання додатків, реалізованими в ОС Android. Файл маніфесту містить основну інформацію про програму, а також різні оголошення та специфікації, необхідні для роботи програми. Файлами ресурсів є зображення, HTML файли, рядки та інші елементи, що використовуються в додатку.

Оскільки DEX файли скомпільовані у виконуваний двійковий код, вони не призначені для читання або інтерпретації, і вихідний код не може бути вилучений безпосередньо з них. Тому, DEX файли потрібно дизасемблювати та декомпілювати у вихідні формати, які можна зчитувати та інтерпретувати. Такі формати представляються Smali кодом (при дизассемблірованні) і Java кодом (при декомпіляції). Smali код - байт-код машини Dalvik, який є такою ж низькорівневою мовою, як і асемблер, тому процес перекладу DEX файлу в Smali код називають дизасемблюванням.

Для отримання дизассембльованого та декомпільованого коду існує безліч утиліт. Один із сценаріїв декодування для ручного аналізу програми може виглядати наступним чином.

РозархівовуванняAPK файлу та декомпіляція вихідних кодів програмою Apk Manager → Конвертація DEX файлу у формат JAR Java коду → Відображення отриманого Java коду в утиліті jd-gui.

Крім того, можна скористатися універсальною утилітою ApkTool, функціонал якої досить широкий і зручний для отримання вихідних кодів та їх подальшої обробки.

В описаних методах потрібно додатково використовувати сторонній парсер або реалізовувати його самостійно, крім того, можуть виникнути проблеми з передачею результатів аналізу програму для їх обробки. Однак є пакети, написані мовою програмування Python, для аналізу та декомпіляції APK файлів. Їхня перевага полягає в тому, що результати аналізу можуть бути передані у вигляді структур даних безпосередньо у потрібну програму, модуль або функцію, які вже працюватимуть з результатами аналізу.

Найбільш примітний пакет в даному випадку - Androguard, який надає широкий функціонал і примітний тим, що в ньому вбудований парсер, що дозволяє отримати потрібні компоненти з APK файлу програми, включаючи перелічені раніше характеристики безпеки, для подальшого використання в оцінці безпеки програми.

2.1.4 Деякі аспекти бінарного кодування

Використання бінарних векторів для представлення характеристик безпеки додатків є ефективним підходом до кодування. У цьому представленні кожен додаток представлений у вигляді вектора довжиною $|S|$, де $|S|$ - кількість характеристик безпеки, які використовує система виявлення шкідливих додатків.

Приклад кодування додатків A і B у бінарні вектори подано на наступному рис. На цьому рисунку також показано приклад формування набору характеристик *S* з характеристик додатків A і B.

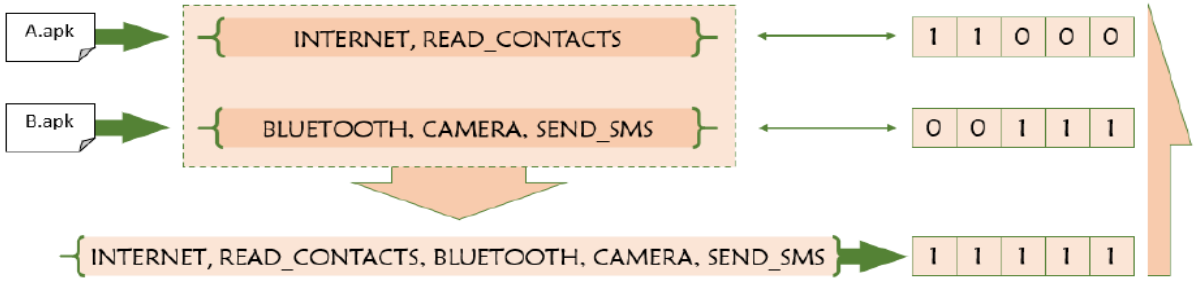


Рис.2.1 Кодування характеристик безпеки

При цьому кожний елемент вектора буде дорівнювати 1, якщо відповідна характеристика безпеки присутня в додатку, і 0 - якщо відсутня. Таким чином, отримавши бінарний вектор для кожного додатка, можна ефективно представити характеристики безпеки додатків для подальшого використання в системі виявлення шкідливих додатків.

Одиничне кодування (One-hot encoding) - це метод кодування, де кожному елементу набору елементів присвоюється бінарний вектор, розмір якого дорівнює кількості елементів у наборі. Кожен елемент вектора однозначно відповідає одному елементу набору. Таким чином, кожен об'єкт, що містить певний набір цих елементів, може бути представлений у вигляді бінарного вектора, де 1 стоїть на позиціях елементів, що належать даному об'єкту, і 0 - на інших позиціях.

2.2 Особливості організації системи Android

2.2.1 Компоненти Android

Android-додаток складається з чотирьох основних компонентів. Для вивчення можливих вразливостей додатків, необхідно розглянути складові детальніше.

Activity - компонент програми, орієнтований на користувача, або візуальний інтерфейс для взаємодії з користувачем, який базується на базовому класі Activity і

керується ActivityManager. Цей менеджер також відповідає за обробку повідомлень Intent, які надсилаються всередині або між додатками.

Broadcast Receiver - Компонент, який приймає та фільтрує широкомовні повідомлення. Для цього додаток має зареєструвати приймач у файлі Manifest, щоб фільтрувати певні типи широкомовних повідомлень за допомогою фільтра Intent, що дозволяє йому явно отримувати такі повідомлення. Крім того, метод registerReceiver може бути динамічно зареєстрований під час виконання програми.

Service - Компонент, що зазвичай використовується для обробки фонових процесів, які потребують багато часу. Користувач не взаємодіє безпосередньо з процесом додатка, що відповідає за Service. Як і інші компоненти Android-додатків, Service може також отримувати та надсилати повідомлення Intent через механізм міжпроцесорного зв'язку (IPC). Щоб використовувати Service, його потрібно зареєструвати у файлі Manifest, див. рисунок 2.2.

Content Provider - Компонент для обміну даними між додатками. Наприклад, ContactsProvider централізовано керує інформацією про контакти, до якої можуть звертатися інші додатки (після отримання дозволу). Додатки можуть створювати власний Content Provider та надавати свої дані іншим додаткам.

```
<receiver android:name="com.qihoo360.mobilesafe.pcdaemon.receiver.DaemonBroadcastReceiver" android:process=":PcDaemon">
  <intent-filter>
    <action android:name="com.qihoo360.mobilesafe.NotifyDaemonStart" />
  </intent-filter>
  <intent-filter>
    <action android:name="com.qihoo360.mobilesafe.NotifyDaemonStop" />
  </intent-filter>
</receiver>
```

Рис. 2.2 Service

```
<service android:exported="false" android:name="com.qihoo360.mobilesafe.privacyspace.PrivacySpaceGuardService" android:process=":GuardService">
  <intent-filter>
    <action android:name="com.qihoo360.mobilesafe.action.ACTION_BIND_APP_LOCK_SERVICE" />
  </intent-filter>
</service>
```

Рис.2.3 Використання Service

2.2.2 Структура APK File

APK (Android application Package) файли зазвичай містять такі файли та директорії:

1. meta-inf директорія - Директорія meta-inf включає наступні файли:
   - manifest.mf : Файл маніфесту.
   - cert.rsa : Файл підпису додатка.
   - cert.sf : Список ресурсів та їх відповідні SHA-1 підписи.
2. lib директорія - Директорія lib містить файли бібліотек, пов'язані з платформою, які можуть включати наступні файли:
   - armeabi : Всі файли, пов'язані з процесорами ARM.
   - armeabi-v7a : Файли, пов'язані з ARMv7 і вище.
   - arm64-v8a : Файли, пов'язані з усіма процесорами ARMv8.
   - x86 : Всі файли, пов'язані з процесорами x86.
   - x86_64 : Всі файли, пов'язані з процесорами x86_64.
   - mips : Файли, пов'язані з процесорами MIPS.
3. res - Директорія res зберігає інші ресурси, які не компілюються в resources.arsc.
4. assets - Директорія assets містить файли ресурсів, до яких можна отримати доступ через AssetManager.
5. AndroidManifest.xml - AndroidManifest.xml - це маніфест файлу компонентів Android, який містить назву додатка, версію, дозволи та іншу інформацію. Цей файл зберігається в APK-файлі у форматі бінарного XML і може бути перетворений у формат текстового XML за допомогою інструментів, таких як apktool, AXMLPrinter2 тощо.
6. classes.dex - classes.dex - це виконуваний файл для середовища виконання Android.
7. resources.arsc - resources.arsc містить скомпільовану частину ресурсів.

2.2.3 DEX File Format

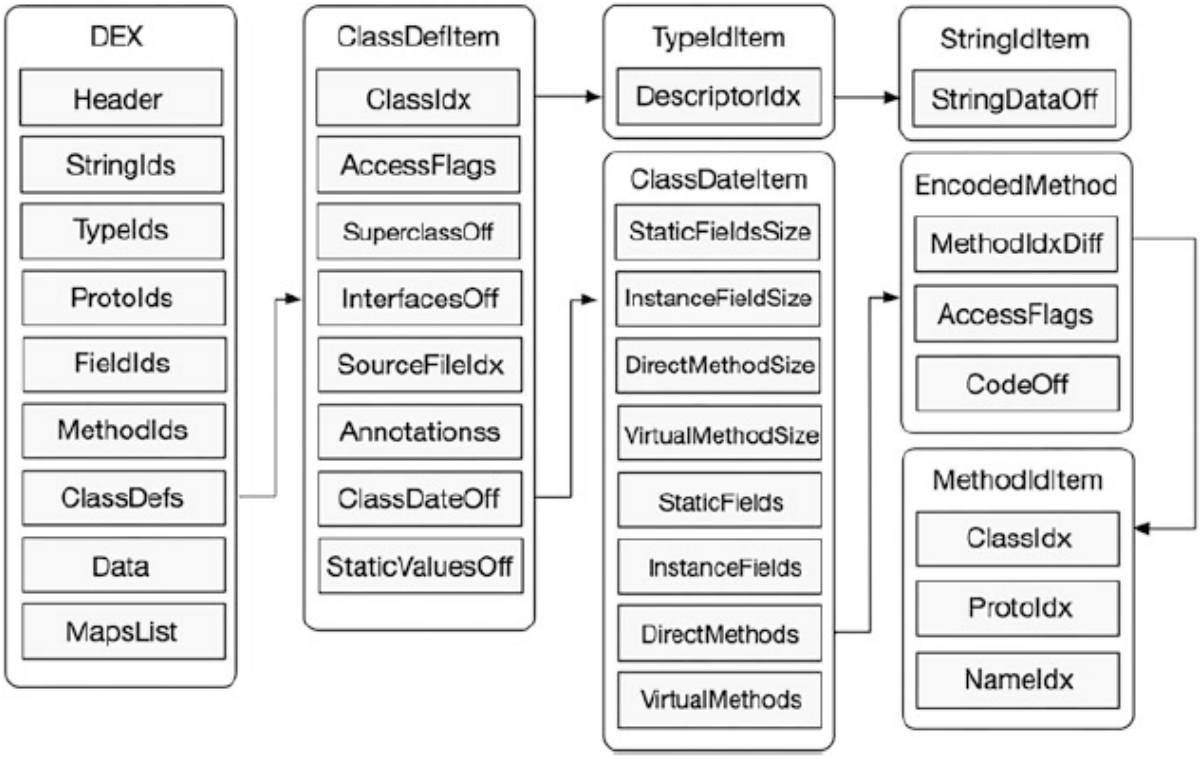


Рис.2.4 Структура DEX файлу

DEX (Dalvik Executable File) є виконуваним файлом Dalvik для Android, і DEX файл містить весь код рівня Java, який виконується. Коли DEX файл стискається та оптимізується, це не тільки зменшує розмір програми, але й підвищує ефективність пошуку класів та методів. Структура DEX файлу зображена на рисунку 4.3.

Заголовок DEX файлу містить дані, такі як розмір файлу, контрольні суми, зсуви та розміри кожної таблиці даних. Типи таблиць даних включають:

- string table : Кожен запис таблиці вказує на зсув рядкових даних. Рядкові дані складаються з двох частин: починається з довжини рядка, закодованої yleb128, далі йдуть самі дані рядка, які закінчуються символом '\0'.
- type table : Зберігає індекси кожного типу в таблиці рядків.
- proto table : Кожен запис містить три елементи: скорочення прототипу функції, індекс типу повернення та зсув параметрів. Перший елемент зсуву параметрів є типу uint і вказує кількість параметрів.

- field table : Кожен запис таблиці описує змінну трьома елементами: клас, до якого належить змінна, тип змінної та ім'я змінної.
- method table : Кожен запис таблиці описує функцію трьома елементами: клас, до якого належить функція, прототип функції та ім'я функції.
- class table : Кожен запис таблиці описує клас вісьмома елементами: ім'я класу, прапор доступу, зсув батьківського класу, зсув інтерфейсу, індекс файлу джерела, коментар класу, зсув даних класу та зсув статичної змінної.
- maps table : Зберігає розміри та початкові зсуви кожної з вищезазначених таблиць, що дозволяє системі швидко знайти кожну таблицю.

2.2.4 Android API

Codename	Version	API level/NDK release
Android11	11	API level 30
Android10	10	API level 29
Pie	9	API level 28
Oreo	8.1.0	API level 27
Oreo	8.0.0	API level 26
Nougat	7.1	API level 25
Nougat	7.0	API level 24
Marshmallow	6.0	API level 23
Lollipop	5.1	API level 22
Lollipop	5.0	API level 21
KitKat	4.4 - 4.4.4	API level 19

Рис. 2.5 Android API list

Ми будемо розглядати рівень API для Android – 28, і відповідна версія – Pie. Кожна основна версія API включає значні зміни. У файлі AndroidManifest.xml можна побачити мінімальну підтримувану версію API для додатка та версію API, яка використовується для компіляції. Офіційний список API Android наведено на рисунку 4.4.

2.2.5 Android Sample Code

Мовою програмування для Android є Java, проте з травня 2017 року, після конференції Google I/O, офіційною мовою програмування для Android була оголошена Kotlin (мова програмування, заснована на JVM), яка компенсує відсутність сучасних можливостей у Java та спрощує код, дозволяючи розробникам писати якомога менше коду. Ми будемо розглядати оригінальний код на Java як приклад для демонстрації базової структури коду Android-додатка.

```
public class MainActivity extends ActionBarActivity {
/   Called when the activity is first created.  /
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.i("CTF", "find_vulns_in_my_diplom!");
}}
```

Точкою входу для Android-додатків є функція onCreate. Файл AndroidManifest.xml містить точку входу додатка, дозволи та прийнятні параметри.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.ctf.test">
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.
READ_EXTERNAL_STORAGE"/>
<application
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
```

android:label="@string/app_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<activity android:name=".MainActivity">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

2.3 Реверс-інжиніринг APK файлів

2.3.1 JEB

Розглянемо основні інструменти та модулі, що використовуються для реверс-інжинірингу APK-файлів. Хороші інструменти можуть значно прискорити процес реверс-інжинірингу. Для платформи Android існує багато інструментів реверс-інжинірингу, таких як Apktool, JEB, IDA, AndroidKiller, Dex2Jar, JD-GUI, smali, baksmali, jadx.

Існує багато декомпіляторів для платформи Android, і JEB є найпотужнішим з них. JEB еволюціонував від раннього декомпілятора APK файлів для Android до інструмента, що підтримує не тільки декомпіляцію APK файлів, але й MIPS, ARM, ARM64, x86, x86-64, WebAssembly, EVM тощо. Його інтерфейс користувача та відкриті інтерфейси є зручними у використанні та значно зменшують складність реверс-інжинірингу, див. рисунок 4.5.

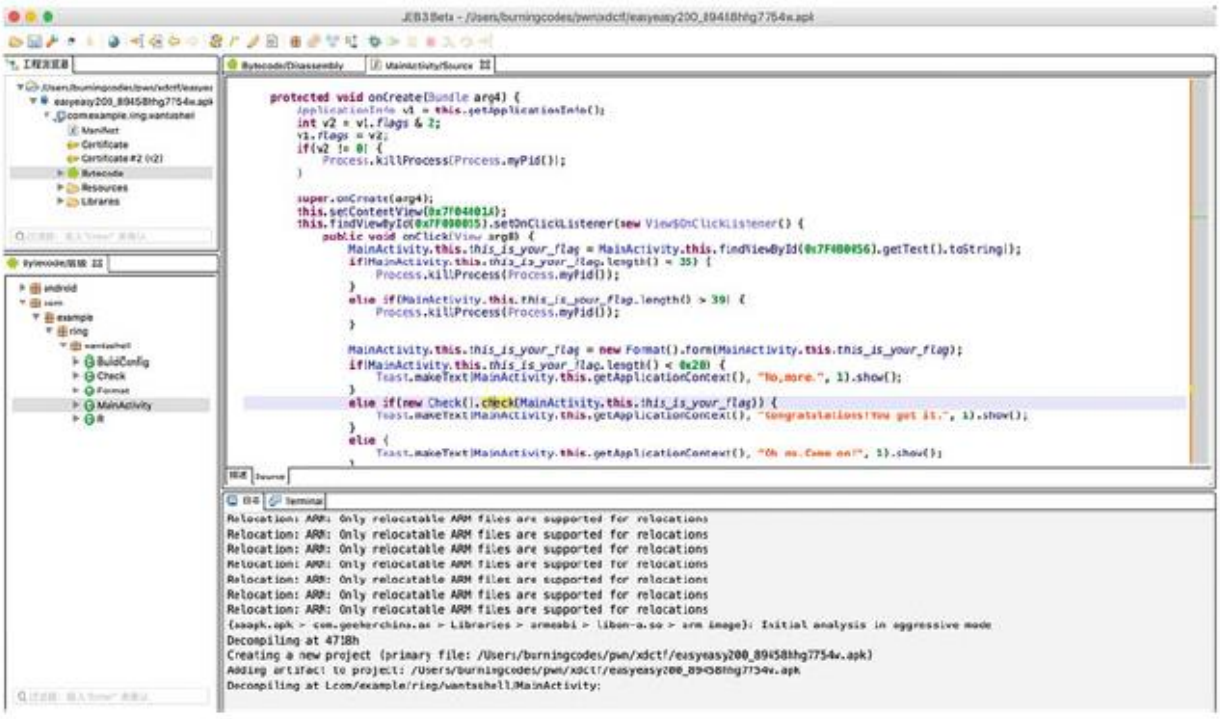


Рис. 2.7 JEB decompile APK

JEB 2.0 підтримує динамічне налагодження, що робить його легким у використанні та освоєнні, і дозволяє налагоджувати будь-який APK файл з увімкненим режимом налагодження. При спробі підключення, якщо процес позначений як D, це означає, що процес можна налагоджувати. В іншому випадку це означає, що прапорець налагодження вимкнений, і процес не можна налагоджувати, див. рисунок 4.6.

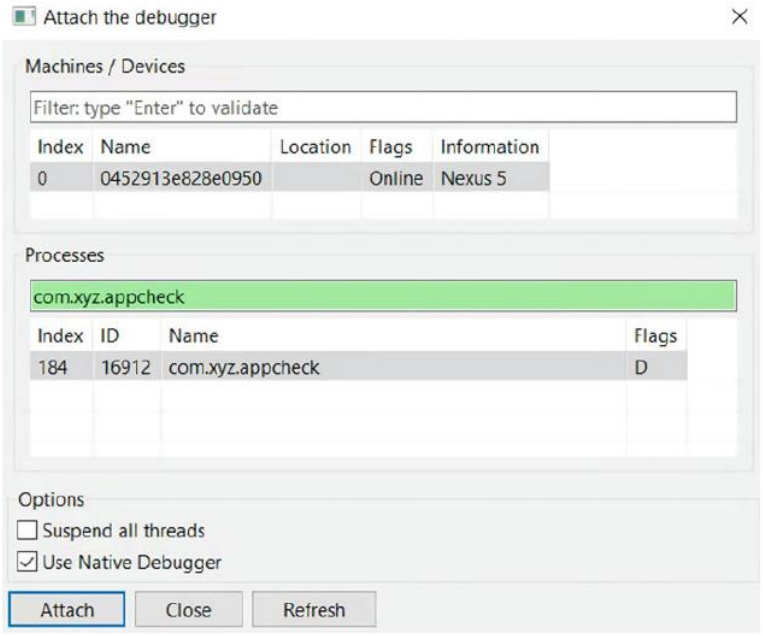


Рис. 2.8 JEB debug APK

Під час налагодження на системі OSX можна встановлювати точки зупинки на рівні smali за допомогою комбінації Command+B. У правому вікні VM/Locals можна переглядати значення кожного регістра в поточному місці. Подвійне натискання дозволяє змінювати значення будь-якого регістра, див. рисунок 2.7.

```
Class pms=SharedObject.masterClassLoader.loadClass("com.android.
server.pm.PackageManagerService");
XposedBridge.hookAllMethods(pms,"getPackageInfo",new XC_MethodHook() {
protected void afterHookedMethod(MethodHookParam param) throws
Throwable {
int x = 32768;
Object v2 = param.getResult();
if(v2 != null) {
ApplicationInfo applicationInfo = ((PackageInfo)v2).applicationInfo;
int flag = applicationInfo.flags;
if((flag&x) == 0) {
flag |= x;}
if((flag&2) == 0) {
```

flag |= 2;}

applicationInfo.flags = flag;

param.setResult(v2);}}});

Коли маємо справу з додатком з вимкненим режимом налагодження або з нерутованим Android-пристроєм, може виникнути неможливість налагодження додатка. У таких випадках можна спробувати примусово увімкнути режим налагодження шляхом підключення до системного інтерфейсу. Наступний код використовується для динамічної зміни стану налагодження нерутованого пристрою за допомогою Xposed Hook. Примусова зміна прапорця налагодження додатка у функції getPackageInfo служби PackageManagerService на стан налагодження дозволить увімкнути режим налагодження, що дасть змогу завершити динамічне налагодження на будь-якому рутованому пристрої.

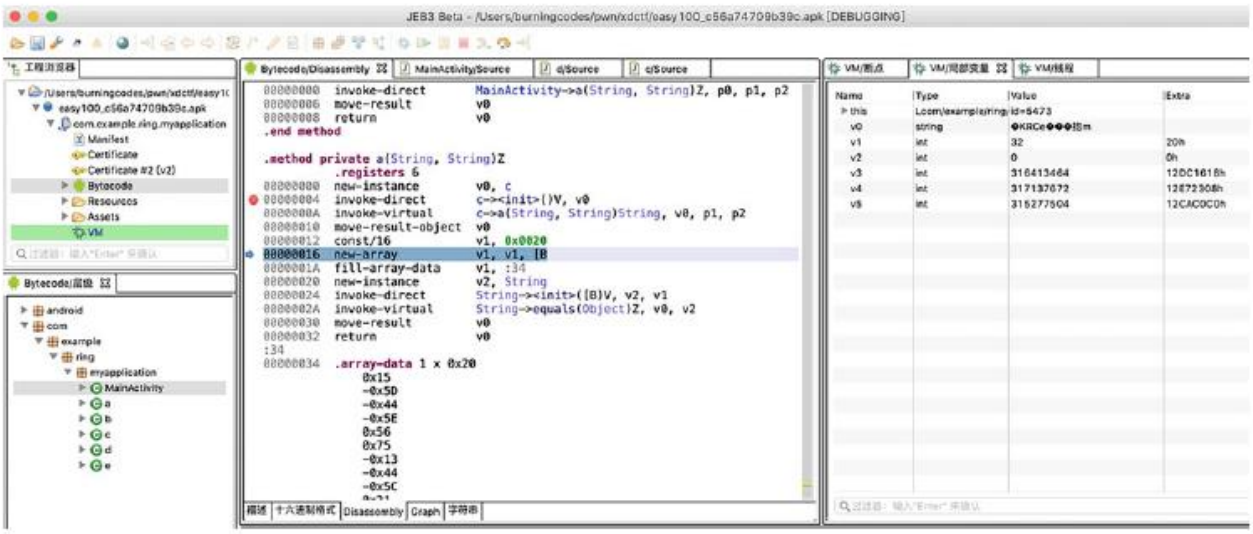


Рис.2.9 Режим налагодження шляхом підключення до системного інтерфейсу.

2.3.2 IDA

При реверс-інжинірингу нативних бібліотек краще використовувати IDA, ніж інші інструменти, такі як JEB, і його динамічне налагодження може значно прискорити швидкість реверс-інжинірингу на рівні нативного коду Android. Ми в

основному розглядали використання IDA для аналізу нативних бібліотек Android (файлів so).

Для відлагодження нативного рівня Android потрібен власний інструмент IDA - android_server: для 32-розрядних Android-телефонів використовуйте 32-розрядний android_server та 32-розрядний IDA; для 64-розрядних Android-телефонів - використовуйте 64-розрядний android_server та 64-розрядний IDA.

Сервер відлагодження IDA слухатиме порт 23946 за замовчуванням, вам потрібно використовувати команду adb forward для пересилання команд порту Android на локальну машину.

adb forward tcp:23946 tcp:23946

Деякі нативні функції (JNI_OnLoad, init_array) будуть автоматично виконуватися за замовчуванням при завантаженні so. Ці функції не можна безпосередньо відлагоджувати за допомогою вищезазначеного методу. Потрібно перервати виконання перед завантаженням динамічної бібліотеки. Оскільки всі динамічні бібліотеки завантажуються лінкером, необхідно знайти позицію, де лінкер завантажує бібліотеку, а потім встановити точку зупинки перед тим, як лінкер ініціалізує so.

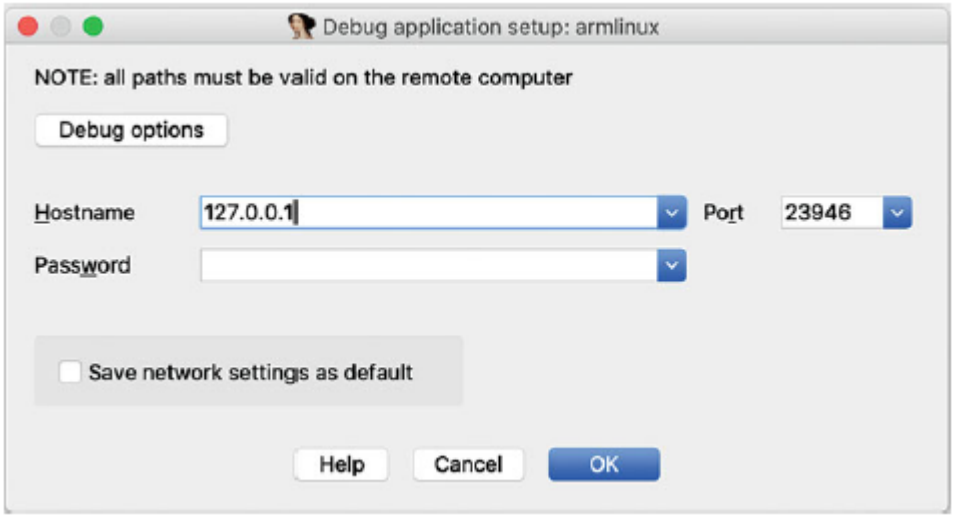


Рис. 2.10 IDA debugging interface

2.3.3 Xposed Hook

Xposed - це фреймворк для перехоплення Android для пристроїв з рут-правами, який дозволяє змінювати стан виконання додатка без зміни вихідного коду. Він працює шляхом заміни стандартного zygote телефону на власний zygote Xposed, який завантажує XposedBridge.jar під час запуску. Кроки для перехоплення Xposed наступні.

Спершу додаються мета-дані, пов'язані з Xposed, до тегу додатка в файлі AndroidManifest.xml, де xposedmodule вказує на те, що це модуль Xposed, xposeddescription описує призначення модуля і може посилатися на рядок у string.xml, а xposedminversion - це мінімальна версія Xposed Framework, яка потрібна.

```
<meta-data
android:name="xposedmodule"
android:value="true" />
<meta-data
android:name="xposeddescription"
android:value="xposed description goes here" />
<meta-data
android:name="xposedminversion"
android:value="54" />
```

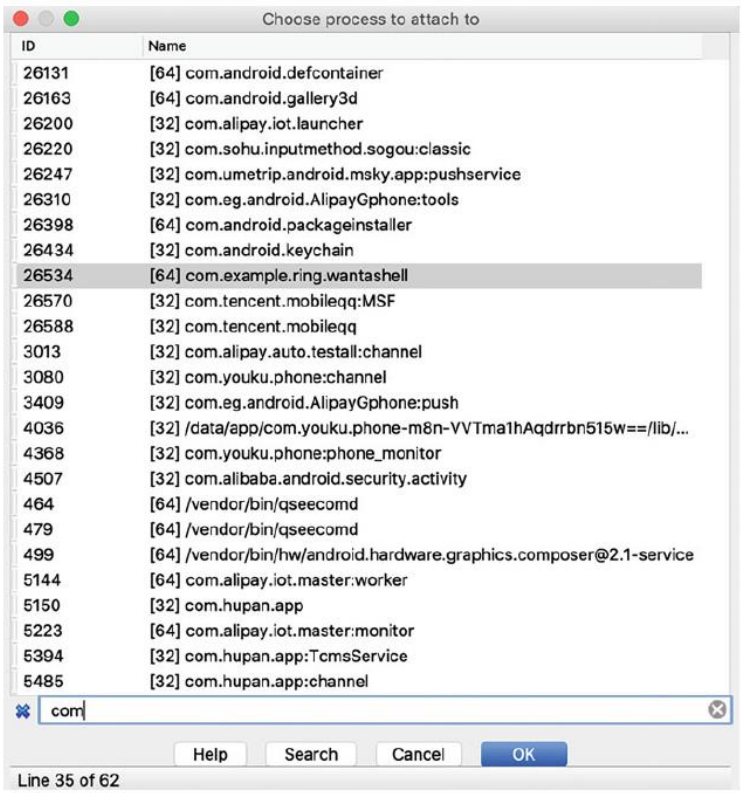


Рис. 2.11 Choose the application to be debugged

Наступне - імпорт пакета XposedBridgeApi jar. У файлі app/build.gradle у Android Studio, додається наступний код:

dependencies {provided files('lib/XposedBridgeApi-54.jar')}

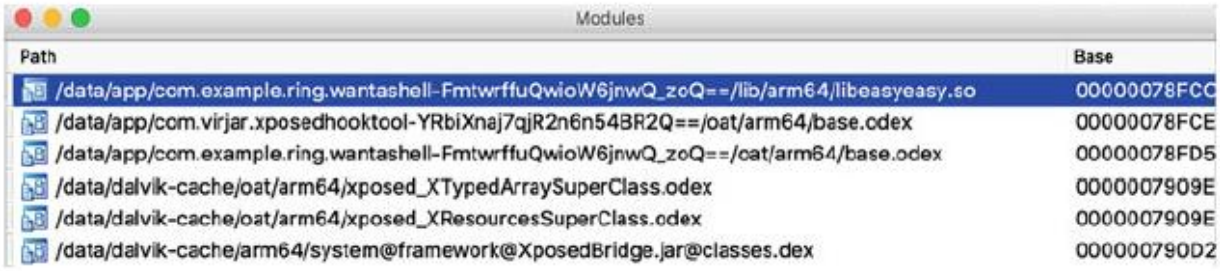


Рис. 2.12 Вибір .so file

Після синхронізації імпорт завершено.

Важливий крок - оголошення входу Xposed.

package com.test.ctf

import de.robv.android.xposed.IXposedHookLoadPackage;

import de.robv.android.xposed.XposedBridge;

import de.robv.android.xposed.callbacks.XC_LoadPackage.LoadPackageParam;

```
import android.util.Log;
public class CTFDemo implements IXposedHookLoadPackage {
public void handleLoadPackage(final LoadPackageParam lpparam) throws
Throwable {
XposedBridge.log("Loaded app: " + lpparam.packageName);
Log.d("YOUR_TAG", "Loaded app: " + lpparam.packageName )}}
```

В каталозі, який створюється, assets – зазвичай створюється файл xposed_init, з якого можна заповнити ім'я класу входу модуля Xposed, наприклад, com. test.ctf.CTFDemo.

Важливий крок - активація модуля Xposed. Активується модуль у додатку Xposed.

2.3.4 Frida Hook

Frida - це крос-платформений фреймворк для перехоплення, який підтримує iOS і Android. Для додатків Android Frida може перехоплювати не тільки функції рівня Java, але й нативні функції, що може значно покращити швидкість реверс-аналізу. Розглянемо деякі техніки використання Frida.

Hook Android Native Functions. Щоб перехоплювати нативні функції Android за допомогою Frida, спочатку потрібно встановити Frida на пристрій або емулятор Android. Після цього можна використовувати скрипти Frida для перехоплення бажаних функцій. Ось приклад простого скрипта для перехоплення нативної функції:

```
// Шукаємо нативну функцію за допомогою її сигнатури
var target_function = Module.findExportByName("libexample.so", "example_function");
// Якщо функцію знайдено, то перехоплюємо її
if (target_function !== null) {
   console.log("Знайдено цільову функцію: " + target_function);
   // Перехоплення функції
```

```
Interceptor.attach(target_function, {
    onEnter: function (args) {
        console.log("Виклик функції example_function");
        // Додаткові дії при вході в функцію   },
    onLeave: function (retval) {
        // Додаткові дії при виході з функції       }    });} else {
console.log("Цільову функцію не знайдено");}
```

Цей скрипт перехоплює нативну функцію з назвою `example_function` з бібліотеки `libexample.so`. При виклику цієї функції буде виведено повідомлення у консоль. Щоб використати цей скрипт, його потрібно запустити з допомогою Frida CLI або Frida-server на пристрої або емуляторі Android.

```
Interceptor.attach(Module.findExportByName("libc.so" , "open"), {
onEnter: function(args) {
send("open("+Memory.readCString(args[0])+","+args[1]+")");},
onLeave:function(retval){}});
```

Hook Android Java Functions. Для перехоплення Java функцій Android за допомогою Frida, можна використовувати Frida з модулем frida-java-bridge. Цей модуль дозволяє перехоплювати методи Java класів безпосередньо з рівня JavaScript. Ось приклад простого скрипта для перехоплення Java методу:

```
    // Визначаємо цільовий клас та метод
    var target_class = "com.example.MainActivity";
    var target_method = "onCreate";
    // Отримуємо посилання на клас
    var class_handle = Java.use(target_class);
    // Перехоплюємо метод
    class_handle[target_method].overload().implementation = function() {
        // Викликаємо оригінальний метод
        this[target_method].apply(this, arguments);
```

// Виводимо повідомлення при виклику методу

console.log("Метод " + target_method + " класу " + target_class + " був викликаний.");};

Цей скрипт перехоплює метод `onCreate` класу `com.example.MainActivity`. При виклику цього методу буде виведено повідомлення у консоль. Щоб використати цей скрипт, ви повинні вказати правильний шлях до цільового класу та методу. Потім ви можете запустити цей скрипт з допомогою Frida CLI або Frida-server на вашому пристрої або емуляторі Android.

Java.perform(function () {
var logtool = Java.use("com.tencent.mm.sdk.platformtools.y");
logtool.i.overload('java.lang.String', 'java.lang.String',
'[Ljava.lang.Object;'].
implementation = function(a, b, c){
console.log("hook log-->"+a+b);};});

Get class member variables from _ _fields_ _:

console.log(Activity.$classWrapper.__fields__.map(function(field) {
return Java.cast(field, Field)}));

Get Android jni env on the Native layer. Для отримання середовища JNI (Java Native Interface) на нативному рівні використовуючи Frida, можна використовувати var env = Java.vm.getEnv();

Наступний код (лістинг) дозволить отримати об'єкт середовища JNI, який можна використовувати для виклику методів Java на нативному рівні за допомогою Frida. Потім можна використовувати це середовище для виклику методів Java, передаючи йому відповідні аргументи.

var env = Java.vm.getEnv();
var arr = env.getByteArrayElements(args[2],0);
var len = env.getArrayLength(args[2]);

Get fields of the Java layer class. Для отримання поля класу на рівні Java за допомогою Frida, можна використати наступний код:

```
Java.perform(function() {
    // Отримання посилання на клас
    var targetClass = Java.use("com.example.TargetClass");
    // Отримання списку полів класу
    var fields = targetClass.class.getDeclaredFields();
    // Виведення інформації про кожне поле класу
    fields.forEach(function(field) {
        console.log("Field name: " + field.getName());
        console.log("Field type: " + field.getType());    });});
```

У цьому коді `com.example.TargetClass` - це ваш цільовий клас. Метод `Java.use()` використовується для отримання посилання на клас. Потім метод `getDeclaredFields()` використовується для отримання списку усіх полів цього класу. Кожне поле має ім'я та тип, які виводяться в консоль.

Важливо відзначити, що для виконання коду на рівні Java з використанням Frida, потрібно викликати його всередині функції `Java.perform()`.

```
var build = Java.use("android.os.Build");
console.log(tag + build.PRODUCT.value);
```

Отримання native address:

```
var fctToHookPtr = Module.findBaseAddress("libnative-lib.so").add(0x5A8);
var fungetInt = new NativeFunction(fctToHookPtr.or(1), 'int', ['int']);
console.log("invoke 99 > " + fungetInt(99) );
```

Отримання app context:

```
var currentApplication = Dalvik.use("android.app.ActivityThread").
currentApplication();
var context = currentApplication.getApplicationContext();
```

Frida зазвичай потребує прав адміністратора для повноцінного функціонування, але також існує можливість введення коду у додатки без необхідності отримання прав адміністратора. Цей процес включає декомпіляцію додатка, ввведення спільних об'єктів (so) зв'язаних з Frida, забезпечення того, що додаток завантажує ці файли під час ініціалізації, і збереження файлу конфігурації libgadget.config.so в каталозі lib, щоб вказати шлях до динамічно введеного коду JavaScript (JS).

Після перепакування додатка з цими змінами, можна використовувати Frida для перехоплення і взаємодії з додатком без необхідності отримання прав адміністратора. Цей метод особливо корисний для тестування на безпеку, налагодження та аналізу додатків у середовищах без прав адміністратора.

Розділ 3. Аналіз APK

3.1 APK Anti-debugging

3.1.1 Виявлення на основі характеристик

З метою захисту додатка розробники часто ускладнюють зворотнє проектування основних рутин додатка різними способами. Техніки налагодження є важливими засобами для зрозуміння логіки цих основних рутин для зворотних інженерів. Як результат, відповідні техніки протидії налагодженню є "бронею" для розробників. Ці анти-техніки в основному походять з платформи Windows і можуть бути розділені на наступні категорії.

Для виявлення характеристик налагоджувача виконується зокрема, перевірка працюючих портів, таких як порт 23946, який за замовчуванням використовується для налагодження IDA.

Виявлення процесів налагоджувачів та їх стану є важливою частиною антиналагоджувальних технік у безпеці додатків для Android. Розуміючи ці методи виявлення, розробники можуть посилити захист своїх додатків від спроб зворотного інженерінгу. Розглянемо кожен з наведених методів виявлення та їх обґрунтування.

3.1.2 Виявлення процесів

Це включає перевірку процесів, які зазвичай асоціюються з налагоджувачами, таких як `android_server` або `gdbserver`. Якщо будь-який з цих процесів виявлено, що він запущений і активний, це може свідчити про наявність відладки. Наприклад:

ps | grep android_server

3.1.3 Перевірка Tracepid

Шляхом перевірки поля `Tracepid` у файлах `/proc/pid/status` або `/proc/pid/task/pid/status`, ми можемо визначити, чи процес відстежується іншим налагоджувачем. Значення `Tracepid` рівне 0 свідчить про те, що процес не відстежується.

cat /proc/pid/status | grep Tracepid

3.1.4 Аналіз /proc/pid/stat

Це включає перевірку другого поля у файлах `/proc/pid/stat` або `/proc/pid/task/pid/stat`. Якщо друге поле - це 't', це свідчить про те, що процес відстежується.

cat /proc/pid/stat | awk '{print $2}'

3.1.5 Перевірка /proc/pid/wchan

Аналіз файлу `/proc/pid/wchan` або `/proc/pid/task/pid/wchan` може показати, чи процес перебуває у стані очікування через налагоджувача, що вказує на `ptrace_stop`.

cat /proc/pid/wchan

3.1.6 Виявлення стану самого процесу

Це включає різні перевірки для визначення, чи процес відстежується або маніпулюється. Наприклад, перевірка, чи батьківський процес - це `zygote`, використання функції `android.os.Debug.isDebuggerConnected`, або пошук програмних точок зупинки в коді.

Щоб уникнути цих методів виявлення, атакуючі можуть налаштувати Android ROM для приховування функцій налагодження. Наприклад, модифікація функції `ptrace` так, щоб вона завжди повертала значення, що не вказує на налагодження, дозволить уникнути виявлення `ptrace`. Також, модифікація вихідного коду для обхіду функції `isDebuggerConnected` може уникнути цього методу виявлення.

3.2 APK Unpacking

3.2.1 Injecting Process and Dumping Memory

Розглянемо наступну проблему: необхідність розпакування APK, захищеного певним типом захисника, за допомогою Frida Hook на системах Android 8.1.

```
Interceptor.attach(Module.findExportByName("libart.so",
"_ZN3art15DexFileVerifier6
VerifyEPKNS_7DexFileEPKhjPKcbPNSt3__112basic_stringIcNS8_
11char_traitsIcEENS8
_9allocatorIcEEEE"), {
onEnter: function(args) {
console.log("verify ")
var begin = args[1]
var dex_size = args[2]
var file = new File("/data/data/com.xxx.xxx/"+dex_size+".dex","wb")
console.log("dex size:"+dex_size.toInt32())
file.write(Memory.readByteArray(begin,dex_size.toInt32()))
file.flush()
file.close()},
onLeave:function(retval){ }});
```

Ідея полягає в тому, що в режимі Dalvik/ART, якщо файл DEX зберігається в пам'яті послідовно, то повинна бути можливість знайти конкретну точку, де файл DEX зберігається цілим. Використовуючи Hook, можна отримати повний

оригінальний файл DEX. Якщо немає анти-Hook коду або анти-Hook не дуже ефективний, це дуже простий і ефективний спосіб розпакувати цільовий додаток.

3.2.2 Modifying the Source

Ідея розпакування шляхом зміни вихідного коду Android схожа на використання Hook, а саме - знайти конкретну точку, де файл DEX зберігається цілим в пам'яті.

```
art/dex2oat/dex2oat.cc Android8.x
make dex2oat
// compilation and verification.
verification_results_->AddDexFile(dex_file);
std::string dex_name = dex_file->GetLocation();
LOG(INFO)<<"supersix dex file name:"<<dex_name;
if(dex_name.find("jiagu") != std::string::npos) {
int len = dex_file->Size();
char filename[256] = {0};
sprintf(filename,"%s_%d.dex",dex_name.c_str(),len);
int fd=open(filename,O_WRONLY|O_CREAT|O_TRUNC,S_IRWXU);
if(fd>0) {
if(write(fd, (char )dex_file->Begin(), len) <= 0) {
LOG(INFO)<<"supersix write fail "<<filename;}
LOG(INFO)<<"wirte successful"<<filename;
close(fd);}
else
LOG(INFO)<<"supersix write fail2 "<<filename;}
```

Наприклад, можливо модифікувати вихідний код dex2oat, щоб позбавитися від оболонки певного постачальника.

Ще один спосіб модифікації вихідного коду для розпакування полягає в зміні наступних файлів в Android 8.1 наступним чином.

```
runtime/base/file_magic.cc
art/sruntime/dex_file.cc
// art/runtime/base/file_magic.cc
#include <fstream>
#include <memory>
#include <sstream>
#include <unistd.h>
#include <sys/mman.h>
File OpenAndReadMagic(const char  filename, uint32_t  magic, std::
string  error_msg) {
CHECK(magic != nullptr);
File fd(filename, O_RDONLY, /  check_usage  / false);
if (fd.Fd() == -1) {
 error_msg = StringPrintf("Unable to open '%s' : %s", filename,
strerror(errno));
return File();}// add
struct stat st;
// let's limit processing file list
if (strstr(filename, "/data/data") != NULL) {
char  fn_out = new char[PATH_MAX];
strcpy(fn_out, filename);
strcat(fn_out, "__unpacked_dex");
int fd_out = open(fn_out, O_WRONLY | O_CREAT | O_EXCL, S_IRUSR |
S_IWUSR | S_IRGRP | S_IROTH);
if (!fstat(fd.Fd(), &st)) {
char  addr = (char )mmap(NULL, st.st_size, PROT_READ,
MAP_PRIVATE, fd.Fd(), 0);
int ret = write(fd_out, addr, st.st_size);
```

ret = 0; // no use

munmap(addr, st.st_size);}

close(fd_out);

delete []fn_out;}

3.2.3 Class Overloading and DEX Reconstruction

Для деяких оболонок файл DEX ніколи не залишається цілим в пам'яті. Тому необхідно використовувати FUPK3, інструмент для відновлення DEX, що базується на модифікації вихідного коду Android. Перш ніж використовувати FUPK3, потрібно внести зміни до вихідного коду Android перед його компіляцією.

Перевантаження класів - це процес створення кількох методів з однаковим іменем у межах одного класу, але з різними параметрами. Це означає, що в одному класі може бути кілька методів з однаковою назвою, але з різними типами аргументів або кількістю аргументів. В Java, як і в багатьох інших об'єктно-орієнтованих мов програмування, перевантаження методів дозволяє створювати більш зрозумілий та ефективний код шляхом використання одного і того ж методу для різних ситуацій. Наприклад, можна мати один метод `calculateArea()`, який приймає аргументи для обчислення площі квадрата, та інший метод `calculateArea()`, який приймає аргументи для обчислення площі кола.

DEX -відновлення - це процес відтворення або розпакування файлів DEX, які були змінені або захищені. Файли DEX є байт-кодом, який використовується в операційній системі Android для виконання програм. Іноді ці файли можуть бути зашифровані або обфусковані, щоб ускладнити їх аналіз та редагування. Для розшифрування або відновлення таких файлів може бути використано різні методи, включаючи модифікацію вихідного коду, використання інструментів розбору байт-коду, які дозволяють переглядати та редагувати файли DEX, або використання спеціалізованих програм для розпакування та відновлення файлів DEX.

Припустимо, що ми маємо захищений файл DEX, який був обфускований для ускладнення аналізу. Щоб розшифрувати цей файл, ми можемо використати

інструменти для аналізу байт-коду, такі як jadx або apktool, для декомпіляції файлу DEX у зрозумілий для нас вихідний код Java.

Наприклад, розглянемо наступний шматок коду у захищеному файлі DEX:

```
public class A {
   public void method() {
      // код методу    }
   public void method(int num) {
      // код методу з аргументом    }}
```

Цей код демонструє перевантаження методів в класі `A`. У першому методі `method()` не передаються аргументи, тоді як у другому методі `method(int num)` передається один аргумент типу `int`.

Після декомпіляції захищеного файлу DEX за допомогою інструменту, наприклад, jadx, ми можемо отримати наступний розшифрований код Java:

```
public class A {
   public void method() {
      // код методу    }
   public void method(int num) {
      // код методу з аргументом    }}
```

Таким чином, ми можемо побачити, що перевантаження методів було успішно відновлено після декомпіляції файлу DEX.

3.3 Практичне застосування реверс-інженірингу на прикладі завдань CTF

3.3.1 OLLVM Obfuscated Native App Reverse (NJCTF 2017)

Розглянемо деякі приклади. На NJCTF 2017 є нативний додаток, написаний виключно на нативному коді, чий вміст AndroidManifest.xml показано на рис. 4.15.

Можна побачити, що додаток має лише один основний клас активності: android.app.NativeActivity. Використовуючи JEB, ми бачимо, що реалізація на рівні

Java відсутня, як показано на рис. 4.16. Цей додаток містить бібліотеку (so), яка використовує OLLVM для замутнення своєї основної логіки, як показано на рис. 4.17.

Поглиблюючись у основну логіку бібліотеки, ми бачимо, що програма отримує координати x, y, z поточного пристрою з акселерометра. Потім вона проводить обчислення та виводить прапорець лише тоді, коли x, y, z відповідають певним умовам. Оскільки бібліотека сильно замутнена, дуже складно зрозуміти задовільну умову, тому може знадобитися розглянути новий вихід. Проаналізуємо функцію під назвою flg:

char __fastcall flg(int a1, char a2)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="1" android:versionName="1.0" package="com.geekerchina.an" platformBuildVersionCode="23" p
  <uses-sdk android:minSdkVersion="15" android:targetSdkVersion="23" />
  <application android:hasCode="false" android:icon="@mipmap/ic_launcher" android:label="@string/app_name">
    <activity android:configChanges="0xa0" android:label="@string/app_name" android:name="android.app.NativeActivity">
      <meta-data android:name="android.app.lib_name" android:value="an-a" />
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Рис. 3.1 AndroidManifest.xml content

Рис. 3.2 JEB

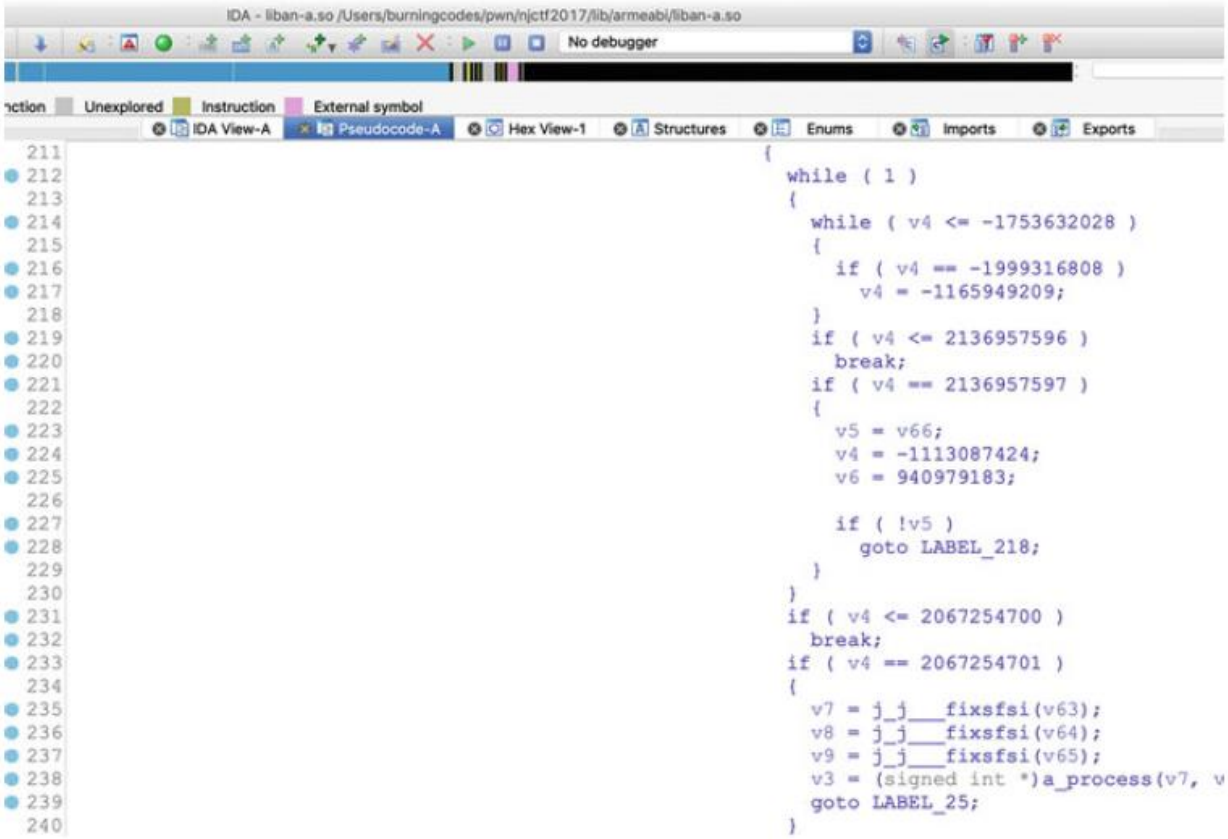


Рис. 3.3 liban-a.so

Ця функція приймає значення типу int і здійснює генерацію рядка (опис завдання передбачає, що рядок складається лише з друкованих символів). Для пошуку рядка простим рішенням є прямий виклик функції flg і запуск повного перебору для пошуку всіх комбінацій друкованих символів прапорця.

Код виглядає наступним чином:

```
int j_j___modsi3(int a, int b) {
return a%b;}
int j_j___divsi3(int a, int b) {
return a/b;}
char flg(int a1, char  out) {
char  v2; // r6@1
int v3; // ST0C_4@1
int v4; // r4@1
int v5; // r0@1
int v6; // ST08_4@1
int v7; // r5@1
int v8; // r0@1
int v9; // r0@1
```

```
char v10; // ST10_1@1
int v11; // r0@1
int v12; // r5@1
int v13; // r0@1
int v14; // ST18_4@1
int v15; // r0@1
int v16; // r0@1
char v17; // r0@1
char v18; // ST04_1@1
int v19; // r0@1
char v20; // r0@1
int v21; // r1@1
int v22; // r5@1
int v23; // r0@1
char v24; // r0@1
v2 = out;
v3 = a1;
v4 = a1;
v5 = j_j___modsi3(a1, 10);
v6 = v5;
v7 = 20   v5;
 v2 = 20   v5;
v8 = j_j___divsi3(v4, 100);
v9 = j_j___modsi3(v8, 10);
v10 = v9;
v11 = 19   v9 + v7;
v2[1] = v11;
v2[2] = v11 - 4;
v12 = v4;
v13 = j_j___divsi3(v4, 10);
v14 = j_j___modsi3(v13, 10);
v15 = j_j___divsi3(v4, 1000000);
v2[3] = j_j___modsi3(v15, 10) + 11   v14;
v16 = j_j___divsi3(v4, 1000);
v17 = j_j___modsi3(v16, 10);
// LOBYTE(v4) = v17;
v4 = v17;
v18 = v17;
v19 = j_j___divsi3(v12, 10000);
v20 = j_j___modsi3(v19, 10);
v2[4] = 20   v4 + 60 - v20 - 60;
v21 = -v6 - v14;
```

```
v22 = -v21;
v2[5] = -(char)v21   v4;
v2[6] = v14   v4   v20;
v23 = j_j___divsi3(v3, 100000);
v24 = j_j___modsi3(v23, 10);
v2[7] = 20   v24 - v10;
v2[8] = 10   v18 | 1;
v2[9] = v22   v24 - 1;
v2[10] = v6   v14   v10   v10 - 4;
v2[11] = (v10 + v14)   v24 - 5;
v2[12] = 0;
return v2;}
int main() {
char out[256], flag = 0;
for(unsigned int I = 0; I <= 4294967295-1; ++i) {
flag = 0;
memset(out, 0, 256);
flg(i, out);
if(strlen(out) >= 10) {
for(int j=0; j<12; ++j) {
if((out[j] >= 'a' && out[j] <= 'z') || (out[j] >= 'A' && out[j] <= 'Z') ||
(out[j] >= '0' && out[j] <= '9')|| out[j] == '_' )
continue;
else {
flag = 1;
break;
}}if(flag == 0)
printf("%s\n", out);}}
return 0;}
```

3.3.2 Anti-debugging and Anti-VM (XDCTF 2016)

На XDCTF 2016 є завдання з реверс-інженерії для Android, яке включає деякі базові техніки протидії відлагодженню та анти-віртуальним машинам. Завжди краще аналізувати програму через відлагодження, тому ми краще спочатку обходимо її анти-механізми.

Щоб позбутися від анти-відлагоджувального коду на рівні Java (як показано на рис. 3.4), ми повинні видалити анти-відлагоджувальний код smali, повторно упакувати та підписати додаток.

Рис. 3.4 The anti-debugging codes on the Java layer

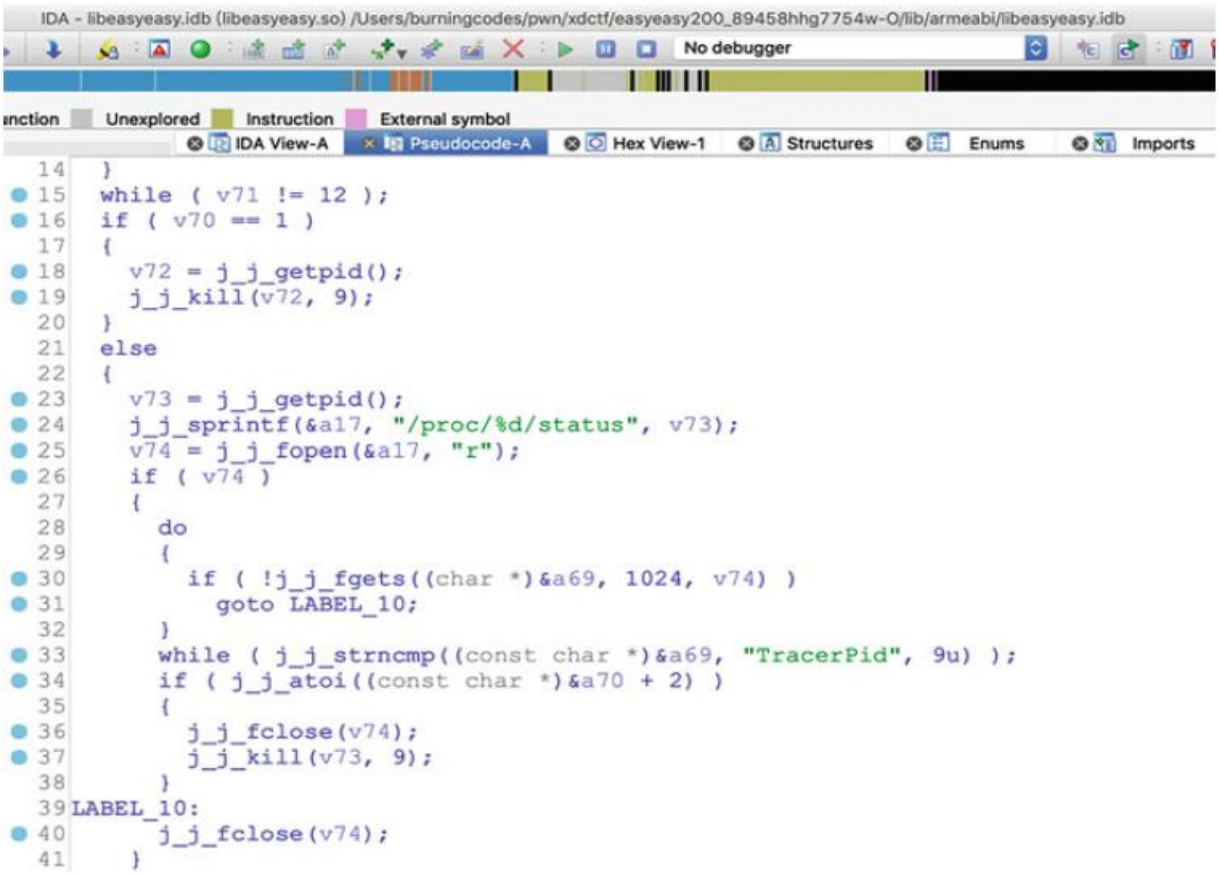


Рис. 3.5 The key function that verifies the flag is on the native layer

Ключова функція, яка перевіряє прапорець, знаходиться на нативному рівні. Як ми бачимо на рис. 3.5, тут також застосовані деякі механізми протидії відлагодженню. Програма перевіряє поточний TracerPid, щоб виявити, чи йде трасування. Відлагоджувальники, такі як IDA Pro, використовують ptrace для відлагодження програми. Якщо ми хочемо обійти цей анти-механізм, ми можемо просто змінити цю функцію, видаливши анти-коди. Або ж, ми можемо змінити вихідний код Android, встановивши TracerPid на постійне значення 0.

dHR0dGlldmFodG5vZGllc3VhY2VibGxlaHNhdG5hd2k

Після того, як ми позбудемося цих анти-механізмів, ми можемо легко з'ясувати, через відлагодження, що програма бере 5 ~ 38 байт вводу, реверсує їх, кодує в base64, а потім порівнює закодований результат з наступним: Отже, ми можемо просто розкодувати цей рядок base64 і повернути його в початковий порядок, тоді прапорець отримаємо наступний результат.

Iwantashellbecauseidonthaveitttt

Розділ 4. Практична реалізація

4.1 Метод факторизації матриць

4.1.1 Загальна ідея методу

Факторизація (розкладання) матриці - це метод, який дає змогу спростити роботу з матрицею шляхом її розкладання на дві скорочені матриці. Цей метод широко застосовується в комп'ютерних обчислювальних системах, оскільки дає змогу спростити та прискорити роботу як над матрицями великого розміру, так і над сильно розрідженими матрицями, тобто такими, що містять більшість нульових елементів.

На практиці існує безліч методів розкладання матриць, які застосовуються як для розв'язання задач лінійної алгебри, так і для інших задач, наприклад, для задачі побудови моделей машинного навчання.

У цій роботі найбільший інтерес представляє розкладання Холецького. Цей розклад може застосовуватися до симетричної позитивно визначеної матриці, а результатом розкладу буде нижня трикутна матриця і матриця, що має вигляд її транспонування. Приклад такого розкладання наведено на Рисунку 3. Однак, основна перевага цього методу полягає в тому, що під час розкладання можна сильно скоротити розмір результуючих матриць без втрати точності.

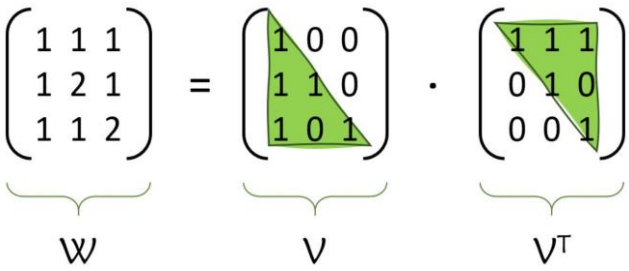


Рис 4.1 Приклад розкладання Холецького симетричної позитивно визначеної матриці

4.1.2 Машини факторизації

У стандартних моделях нейронних мереж для виявлення шкідливих застосунків за вектором характеристик безпеки не враховується так звана "взаємодія" характеристик. Очевидно, якщо в застосунку присутні кілька

небезпечних характеристик, об'єднання деяких із них може однозначно вказувати на шкідливість застосунку - саме в цьому і полягає сенс урахування "взаємодії" характеристик безпеки застосунків.

"Взаємодія" характеристик представлена їхнім попарним впливом одна на одну за принципом кожна-з-кожною, таким чином формується матриця розміром n^2, де n - кількість характеристик. Оскільки характеристик велика кількість, то тільки зберігання матриці розміром їхньої кількості у квадраті потребує великих витрат пам'яті, а з огляду на те, що на кожному кроці навчання мережі елементи матриці мають змінюватися, тобто відбувається постійна взаємодія з такою великою структурою, навчання потребуватиме не тільки великих витрат пам'яті, а й часу.

Машини факторизації були представлені Штеффеном Рендлом [33] у 2010 році як клас нових нейромережевих моделей, що поєднують у собі переваги машини опорних векторів (SVM) і факторизаційних моделей. Машини факторизації можуть виконувати завдання регресії, класифікації та ранжування. Особливість машин факторизації полягає в опрацюванні будь-яких вхідних векторів, що представляють об'єкти реального світу, та оцінюванні парних взаємодій їхніх елементів навіть за умови сильної розрідженості цих векторів, на відміну від SVM, які не можуть працювати з сильно розрідженими даними, або можуть, але видають низьку точність.

Ще одна перевага машин факторизації - лінійна складність $O(kn)$, де k - гіперпараметр, що визначає розмірність факторизації, а $n-$ кількість ознак описуваного об'єкта (розмір вхідного вектора).

Найбільшої популярності машини факторизації набули в рекомендаційних системах, які мають враховувати парні взаємодії, при цьому маючи в своєму розпорядженні сильно розріджені дані. Однак, більшість даних, що описують об'єкти реального світу, є сильно розрідженими через наявність безлічі можливих ознак, але містять у собі лише малу частину всіх цих ознак. Так само і Android застосунки - всі разом мають безліч, залежно від вибору, від десятків до

сотень тисяч, характеристик безпеки, але окремі застосунки, найчастіше, містять лише кілька характеристик і в найрідкісніших випадках близько тисячі характеристик, що все одно є малою частиною від їхньої загальної кількості.

Математично, ідея взаємодії компонентів x з вагами w записується у вигляді поліноміальної регресії другого порядку:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j$$

де W - матриця ваг взаємодії x_i и x_j.

Як було описано вище, робота з такою матрицею не є можливою, однак, матриця W може бути розкладена методом Холецького:

$$W = VV^T.$$

Якщо уявити v_i в якості $i-$ ой рядок V, нейронна мережа тренуватиме прихований вектор v_i для кожного x_i і ваги моделі парної взаємодії w_{ij} будуть представлені як скалярні добутки відповідних прихованих векторів для x' и x_j:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \qquad (1)$$

де скалярний добуток векторів v розмірності k рахується за такою формулою:

$$\langle v_i, v_j \rangle = \sum_{m=1}^{k} v_{i,m} v_{j,m}$$

Формула (1) і є математичне представлення моделі машини факторизації. У моделі машини факторизації параметри w_0, w_i, v_i тренуються і оновлюються з використанням стохастичних методів, які виконують псевдовипадкові зміни величин ваг, зберігаючи ті зміни, що ведуть до поліпшень. На практиці найчастіше застосовується метод градієнтного спуску.

Приховані вектори v_i формуються в процесі навчання мережі і найчастіше їхня розмірність k у багато разів менша за n - розміру вхідного вектора xчерез його розрідженість. Це гарантує те, що не доведеться зберігати велику кількість

непотрібних даних і працювати з матрицею розміру $n \times n$. Завдяки даному поданню складність методу і знижується з $O(n^2)$ до $O(kn)$.

Машини факторизації можуть бути розширені і до поліноміальної регресії більшого порядку для дослідження взаємодії більш ніж двох призноків об'єкта:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=2}^{d} \sum_{i_1=1}^{n} \dots \sum_{i_l=i_{l-1}+1}^{n} \left(\prod_{j=1}^{l} x_{i_j} \right) \left(\sum_{f=1}^{k_l} \prod_{j=1}^{l} v_{i_j,f}^{(l)} \right)$$

У такому разі складність дорівнює $O(k_d n^d)$, але при математичному перетворенні рівняння (2) складність зводиться все до тієї ж лінійної $O(kn)$. Доказ цьому наведено в роботі [33].

4.1.3 Машини факторизації з урахуванням специфіки полів

Моделі, що працюють за принципом машин факторизації з урахуванням полів, є розширенням стандартної моделі машин факторизації. Вони були запропоновані Рендлом і Шмідом-Тімом [34] у 2010 році, а також Андреасом Тешером та іншими на конкурсі KDD Cup у 2012 році. Однак, конкретна концепція машин факторизації з урахуванням полів була сформована Ючін Цзюанем та іншими [35] у 2016 році.

Для машин факторизації з урахуванням специфіки полів окрім характеристик, що використовуються в стандартній моделі, вводиться додатковий параметр поле. Поле - це клас, група або будь-яке інше об'єднання характеристик. Так, наприклад конкретні дозволи Android додатків
(READ_CONTACTS, INTERNET тощо) є характеристиками, а полем для них усіх може бути "Роздільна здатність". На відміну від досить конкретних характеристик, поля можуть задаватися довільно, наприклад, роздільності можуть бути поділені на такі поля, як "Компоненти пристрою" (CAMERA, BLUETOOTH і т. д.), "Мобільний зв'язок" (CALL_PHONE, READ_SMS і т. д.), та інші, залежно від потрібної гранулярності.

На відміну від стандартної машини факторизації, де з кожною характеристикою був пов'язаний тільки один прихований вектор v_i у розширеній моделі з урахуванням специфіки полів, під час навчання, для кожної характеристики

створюється стільки прихованих векторів, скільки введено полів, і поліноміальна регресія другого порядку матиме такий вигляд:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_{i,f_s}, v_{j,f_r} \rangle x_i x_j$$

де f_s - поле характеристики x_j, а f_r - поле характеристики x_i.

Для кращого розуміння цієї концепції можна розглянути приклад на основі наведених вище полів і характеристик. Припустімо, модель побудована на трьох характеристиках із відповідними їм полями, наведеними в Таблиці 2.

Табл. 4.1- Характеристики Android додатків, розділені на відповідні поля

Компоненти пристрою (КУ)	Мобільний зв'язок (МС)	Місцезнаходження (МП)
КАМЕРА (CAM)	CALL_PHONE (CP)	ACCESS_FINE_LOCATION (AFL)

Тоді для стандартної моделі машини факторизації взаємодія характеристик для довільного додатка буде описуватися таким виразом:

$$\langle v_{CAM}, v_{CP} \rangle x_{CAM} x_{CP} + \langle v_{CAM}, v_{AFL} \rangle x_{CAM} x_{AFL} + \langle v_{AFL}, v_{CP} \rangle x_{AFL} x_{CP},$$

де x_n дорівнюватиме 1 , якщо характеристика n міститься в додатку, і 0 в іншому випадку.

Відповідно, для машини факторизації з урахуванням специфіки полів взаємодія характеристик матиме такий вигляд:

$$\langle v_{CAM,\text{МС}}, v_{CP,\text{КУ}} \rangle x_{CAM} x_{CP} + \langle v_{CAM,\text{М}}, v_{AFL,\text{КУ}} \rangle x_{CAM} x_{AFL} + \langle v_{AFL,\text{МС}}, v_{CP,\text{М}} \rangle x_{AFL} x_{CP}$$

Тут приховані вектори v зовсім інші - якщо в першому випадку для характеристики CAMERA прихований вектор v_{CAM} був одним для взаємодій x_{CAM} с x_{CP} и x_{CAM} с x_{AFL}, то в другому випадку, у зв'язку з тим, що x_{CP} и x_{AFL} належать

різним полям, для характеристики CAMERA створюється два приховані вектори $-v_{CAM,\text{мс}}$ и $v_{CAM,\text{м}}$.

4.2 Порівняння моделей

Незважаючи на те, що модель машин факторизації з урахуванням специфіки полів відрізняється від стандартної моделі лише збільшеною кількістю прихованих векторів, що навчаються, вони по-різному проявлятимуть себе залежно від розв'язуваного завдання. Тоді вибір найбільш підходящої моделі для відповідного завдання гарантуватиме підвищення точності класифікації, регресії та ранжування. Тому важливо виділити умови, за яких варто використовувати ту чи іншу модель машин факторизації. Плюси та мінуси кожної моделі наведено в Таблиці 3.

Табл. - Порівняння моделей машин факторизації

	Стандартна модель	Модель з урахуванням специфіки полів
Час навчання і розпізнавання (за однакового параметра k)	Менше	Більше
Використовувана пам'ять (за умови однакового ковому параметрі k)	Менше	Більше

Табл - Порівняння моделей машин факторизації (продовження)

	Стандартна модель	Модель з урахуванням специфіки полів	

Розмірність k прихованих векторів, за якої досягається найбільша точність	Більше	Менше	
Набір даних	Щільність	Будь-який	Сильно розріджений-ний
	Роздільність	Будь-який	На велику кількіс-ство класів (полів)
	Тип даних	Будь-який	Категоріальні

Розглядаючи це питання з боку використовуваних ресурсів, тобто займаної пам'яті та часу, що витрачається на тренування і розпізнавання, машини факторизації з урахуванням специфіки полів поступаються стандартній моделі за однакових параметрів моделей у зв'язку з тим, що стандартна модель навчає і зберігає лише один прихований вектор для кожної характеристики, а розширена - по вектору на кожне поле.

З іншого боку, у зв'язку з тим, що машини факторизації з урахуванням специфіки полів навчають приховані вектори щодо окремих полів, а не на основі всіх характеристик, як у стандартній моделі, розмірність k прихованих векторів може бути в рази меншою за тієї самої точності розпізнавання на відміну від звичайних машин факторизації.

Крім того, в оригінальній роботі [35] доведено, що машини факторизації з урахуванням специфіки полів показуватимуть кращі результати за умови достатньої розрідженості даних, у яких може бути виділено досить велику кількість полів. У

цій же роботі модель було протестовано як на категоріальних, так і на числових даних і зроблено висновок, що для числових даних результати роботи машин факторизації з урахуванням специфіки полів не кращі за стандартну модель.

Таким чином, метод факторизації матриць і засновану на ньому модель машинного навчання - машини факторизації, а також її розширення - машини факторизації з урахуванням специфіки полів є досить ефективними. Математичною основою моделей, крім факторизації, є поліноміальна регресія другого порядку для дослідження парної взаємодії характеристик і вищого порядку для, відповідно, взаємодій вищого порядку. Було проведено порівняння моделей і зроблено висновок, що розширену модель слід застосовувати в разі сильної розрідженості даних, які мають бути категоріальними, а також поділюваними на достатню кількість полів (класів). Дані Android додатків підходять під перераховані умови, з чого можна зробити висновок, що при застосуванні моделі машин факторизації з урахуванням полів точність виявлення шкідливих додатків повинна бути вищою, ніж у стандартної моделі.

4.3 Практична реалізація алгоритмів

4.3.1 Датасет drebin

Реалізовані системи виявлення шкідливих Android-додатків є бінарними класифікаторами, оскільки проблема виявлення ґрунтується тільки на двох класах - шкідливих і безпечних додатках. Таким чином, не потрібно використання складних моделей, що виконують множинну класифікацію.

Увесь код написаний мовою програмування Python з огляду на його простоту і велику кількість реалізованих модулів.

Для систем виявлення шкідливих Android додатків датасетами є самі додатки. Найбільшу складність становить те, що більшість наборів є закритими, зважаючи на вміст у них шкідливих прикладів, і до них складно отримати доступ.

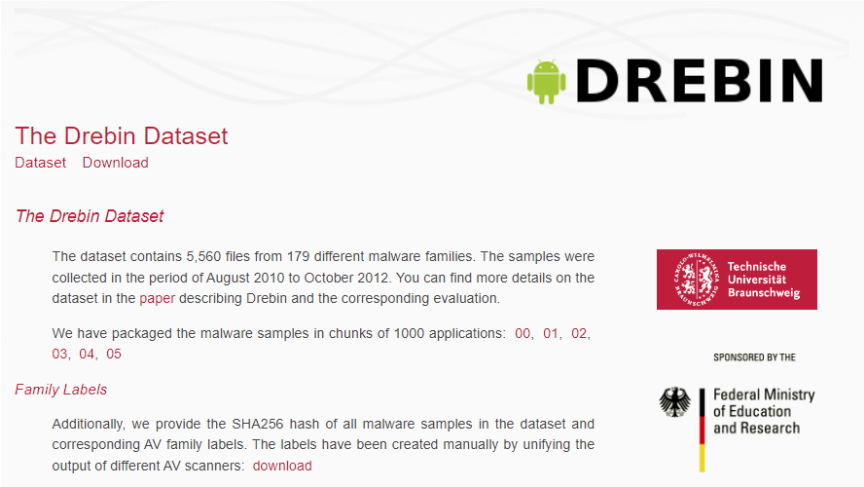


Рис.4.2 Датасет Drebin

Одним із найпопулярніших наборів даних є DREBIN [16]. Цей набір містить зразки шкідливих програм, зібраних з 2010 по 2012 роки. До нього включено 5560 додатків зі 179 сімейств шкідливих програм. DREBIN є закритим набором даних і отримати до нього доступ можна тільки зв'язуючись з його власниками поштою.

Для ще двох популярних закритих датасетів, Genome [27] і AMD [36], зовсім припинили видачу дозволів на доступ, і отримати зараз їх не представляється можливим. Однак це були великі та якісні набори. Genome містив понад 1200 застосунків, що покривають більшість існуючих на той момент родин шкідливих програм і зібраних із 2010 до 2011 року. До AMD входило 24553 зразки, розділених на 135 різновидів серед 71 сімейства шкідливих програм, зібраних у період з 2010 по 2016 рік.

Найбільший на даний момент з відомих наборів даних - Andro-Zoo [37]. Він постійно розширюється і на кінець травня 2021 року містить понад 15 мільйонів додатків. AndroZoo збирає додатки з кількох джерел, зокрема, з офіційного магазину Android додатків Google Play. Після потрапляння в базу даних AndroZoo застосунок аналізується десятками різних антивірусів для однозначного визначення того, чи є застосунок безпечним.

VirusShare - великий закритий репозиторій, що містить зразки шкідливих додатків різних сімейств під основні системи. Виявлені додатки під ОС Android

розділені на роки згідно з їхньою публікацією. Загалом у репозиторії міститься понад 180 тисяч шкідливих Android додатків. Доступ до них можна отримати шляхом надсилання запиту електронною поштою.

Набори даних Канадського інституту кібербезпеки UNB перебувають у відкритому доступі та містять велику кількість репозиторіїв, створених на базі інституту. Репозиторії містять як самі додатки - шкідливі і безпечні, так і сформовані CSV файли, що містять вектори характеристик додатків і готові до подачі в системи машинного навчання.

Крім того, додатки містяться і в магазинах Android застосунків - Google Play та на альтернативних платформах для розповсюдження застосунків. Однак під час їхнього ручного збирання не можна бути впевненим у їхній безпеці, потрібно буде організовувати роботу з їхньої перевірки та формування баз безпечних і шкідливих застосунків. Такий спосіб найчастіше є зайвим, оскільки перераховані раніше репозиторії, найімовірніше, вже зберігають усі зразки, які можна знайти у відкритому доступі.

Ми будемо використовувати датасет Drebin у форматі csv, який містить інформацію про мобільні додатки, розділені на дві категорії: шкідливі та безпечні. Цей датасет є частиною дослідження з безпеки мобільних додатків, де метою є ідентифікація та класифікація шкідливих програм.

Завантаження даних ¶

Використовуємо відкритий датасет

```
[4]: data = pd.read_csv("drebin-215-dataset-5560malware-9476-benign.csv", encoding="utf-8", low_memory=False, na_values="?")
     data.head()
```

[4]:

	transact	onServiceConnected	bindService	attachInterface	ServiceConnection	android.os.Binder	SEND_SMS	Ljava.lang.Class.getCanonicalName	Ljava.lang.Class.getMethods	Ljava.lang.Class.cast	...	READ_CONTACTS	DEVI(
0	0	0	0	0	0	0	1	0	0	0	...	0	
1	0	0	0	0	0	0	1	0	0	0	...	0	
2	0	0	0	0	0	0	1	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	1	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 216 columns

Рис. 4.3 Датасет

data = pd.read_csv("drebin-215-dataset-5560malware-9476-benign.csv", encoding="utf-8", low_memory=False, na_values="?")

data.head()

4.3.2 Робота з JSON

Оскільки набори додатків доволі великі та їхній розбір займає багаточасу, крім того, можливі виняткові випадки, коли робота раптово переривається, потрібно організувати проміжне збереження результатів. У зв'язку з тим, що в Python зручно організована робота з json форматом за допомогою однойменного стандартного модуля, було вирішено зберігати дані саме в цьому форматі.

Формат і приклад заповнення JSON-файлу наведено на Рисунку 4. Файл складається з 5 основних полів:

1. all_features - список, що містить імена всіх характеристик безпеки розібраних додатків.
2. list_of_vectors - список, що містить списки, які представляють бінарні вектори, що відповідають списку all_features. Якщо n-а характеристика з all_features міститься в додатку, то на n-ій позиції вектора буде знаходитися ділитися 1, інакше - 0.
3. results - список, що показує, яким є застосунок - шкідливим або безпечним. Якщо n-ий вектор списку vectors представляє шкідливий додаток. носний додаток, то на n-ій позиції списку results перебуватиме -1, якщо ж безпечне, то 1.
4. stage - етап, на якому було виконано останнє збереження (malware або benign).
5. num - кількість оброблених додатків на відповідному етапі.

```
import json
# Завантаження JSON даних з файлу
with open("data.json", "r", encoding="utf-8") as file:
    data = json.load(file)
# Перегляд завантажених даних
print(data)
```

4.3.3 Формат CSV

Файл у форматі CSV (Comma-Separated Values) містить табличні дані, де кожен рядок представляє собою запис, а значення розділені комами. Кожен стовпець відповідає окремому атрибуту або характеристиці запису.

Перший рядок CSV файлу зазвичай містить заголовки стовпців, які описують, які дані містяться в кожному стовпці.

Кожен наступний рядок містить значення, розділені комами, які відповідають заголовкам стовпців.

Для датасету, який ми використовуємо в роботі, "drebin-215-dataset-5560malware-9476-benign.csv" структура є наступною:

app_id,permission_1,permission_2,api_call_1,api_call_2,network_activity,label

com.example.app1,1,0,2,1,3,benign com.example.app2,0,1,1,0,2,malware com.example.app3,1,1,2,2,1,benign

- app_id : Унікальний ідентифікатор додатку.
- permission_1, permission_2, : Дозволи, запитувані додатком (наприклад, доступ до камери, контактів тощо).
- api_call_1, api_call_2, : Виклики API, що використовуються додатком.
- network_activity : Інформація про мережеву активність (наприклад, кількість відправлених/отриманих пакетів).
- label : Мітка, яка вказує, чи є додаток шкідливим (malware) або безпечним (benign).

Завантаження та обробку CSV файлу можна виконувати за допомогою pandas

import pandas as pd

Для завантаження даних з файлу виконуємо наступне:

data = pd.read_csv("drebin-215-dataset-5560malware-9476-benign.csv", encoding="utf-8", low_memory=False, na_values="?")

- `na_values="?"` : Вказує, що значення `?` у файлі слід розглядати як відсутні дані (NA).

Після завантаження даних можна виконувати різні операції:

- Перегляд перших рядків : `data.head()`
- Інформація про датасет : `data.info()`
- Статистичний аналіз : `data.describe()`
- Фільтрація даних : `data[data['label'] == 'malware']`
- Обробка відсутніх значень : `data.fillna(0)` або `data.dropna()`

Формат CSV є дуже поширеним та зручним для зберігання та обміну табличними даними, особливо для подальшого аналізу та обробки.

4.3.4 Деякі аспекти реалізації класифікатора

В роботі реалізовано класифікатор, який зчитуєнабір вхідних APK файлів і переводить їхні характеристики у список бінарних векторів. Для цього насамперед було сформовано два каталоги з безпечними та шкідливими додатками. Таким чином, спочатку зчитуються імена додатків із каталогів, шляхи до яких записуються в коді у змінні datasets_folder (загальначастина шляху), benign_folder і malware_folder, і формуються відповідні списки.

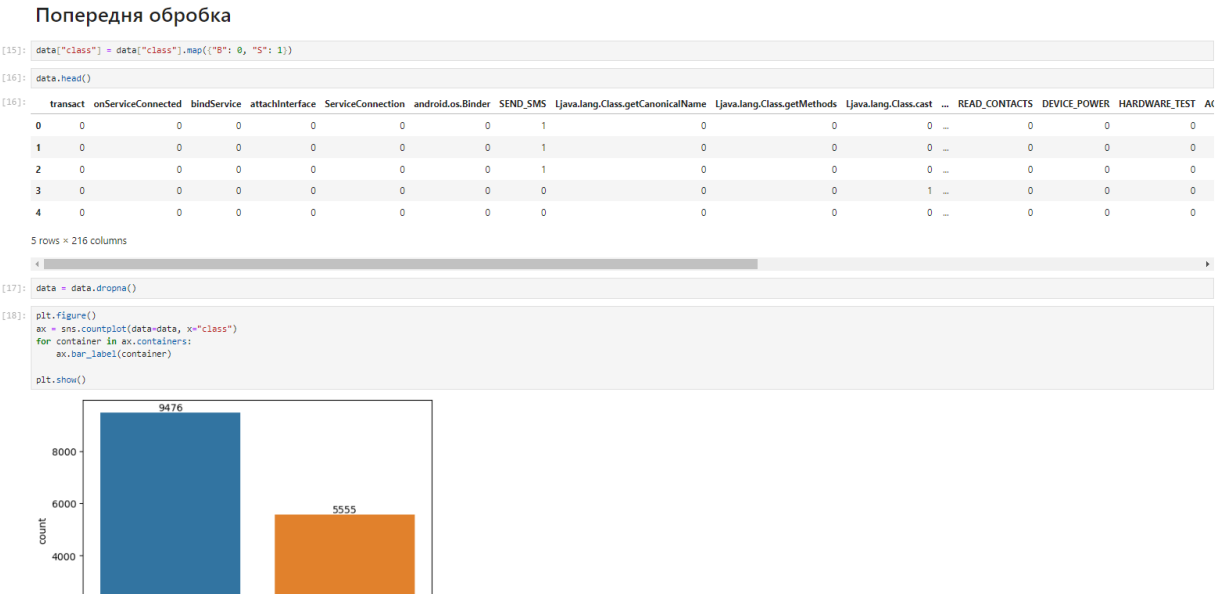


Рис. 4.4 Попередня обробка даних

trust = sorteded([f for f in listdir(datasets_folder + benign_folder) if isfile(join(datasets_folder + benign_folder,f))])

malware = sorted([f for f in listdir(datasets_folder + malware_folder) if isfile(join(datasets_folder + malware_folder,f))])

Після цього, окремо для шкідливих і безпечних додатків, викликаємо функцію, що відповідає за перебір файлів.

get_features_from_files(trust, datasets_folder + be- nign_folder)

У ній запускається цикл за всіма отриманими раніше іменами файлів:

```
while i < len(listfiles):
    filename = listfiles[i]i += 1
    features = get_features(directory + '/' + filename)
    if not features:
        continue list_of_vectors.append(get_features_vector(features))
```

Тут listfiles - переданий список імен додатків, який, утворюючи повний шлях, передається у функцію get_features. Ця функція за допомогою засобів модуля Androguard декодує APK-файл, дизасемблює вихідні коди і повертає потрібні характеристики програми шляхом виклику відповідних функцій. Крім того, перед опрацюванням застосунку файл APK перевіряється на те, що він є ZIP-файлом, і організовується опрацювання винятків для коректного продовження роботи навіть у разі проблем з APK-файлом.

```
def get_features(apk_name):
    features = []
    if zipfile.is_zipfile(apk_name):try:
        apk, dalvik, analysis = AnalyzeAPK(apk_name) features +=
        apk.get_permissions()
        features += apk.get_declared_permissions() features
        += apk.get_features()
        features += apk.get_libraries() features
        += apk.get_providers() features +=
        apk.get_receivers()
```

features += apk.get_requested_aosp_permissions() features += apk.get_uses_implied_permission_list()del apk

Таким чином, за допомогою модуля Androguard формується список наступних характеристик: дозволи, оголошені в маніфесті додатка; дозволи, запитувані під час роботи програми; використовувані апаратні засоби; використовувані бібліотеки; використовувані постачальники контенту; підписки на широкомовні повідомлення; привілейовані дозволи системи Android;

Якщо додаток було вдало опрацьовано, то результат роботи функції get_features подається на вхід функції get_features_vector, яка, своєю чергою, кодує отримані характеристики в бінарний вектор, що відповідає конкретному додатку.

Насамперед формується список feature_vector, що містить нулі на всіх позиціях. Розмір цього списку дорівнює розміру списку all_features, який містить усі назви раніше зчитаних характеристик.

feature_vector = [0] len(all_features)

Далі, кожну характеристику потрібно розібрати так, щоб отримати назву характеристики, яка не залежить від конкретного пакета програми.

feature = rawfeature[rawfeature.rfind('.')+1:]

Далі, якщо отримане ім'я характеристики міститься у списку all_fea- tures, то на відповідній позиції feature_vector встановлюється 1.

if feature in all_features: feature_vector[all_features.index(feature)] = 1

Якщо характеристика не міститься в all_features, то її додають до all_features, а в кінець кожного вектора, що відповідає додатку, закодованому раніше, додають елемент, що містить 0.

```
all_features.append(feature)  for  vector
in list_of_vectors:
    vector.append(0)
feature_vector.append(1)
```

Результатом функції є закодований вектор характеристик конкретного Android-додатка, який додається до результуючого списку векторів list_of_vectors.

list_of_vectors.append(get_features_vector(features))

У результаті роботи коду, що відповідає за формування навчальної вибірки, отримуємо список list_of_vectors, що містить навчальні вектори, а також список all_features, який містить імена всіх отриманих раніше характеристик із навчальних додатків. Крім того, формується список results, у якому зберігаються результати еталонної класифікації. Також, з огляду на те,що опрацювання застосунків для формування навчальної вибірки - найтрудомісткіше завдання в контексті реалізованої системи (деякі застосунки можна опрацьовувати приблизно за 10 хвилин, і цей час пропорційний до розміру застосунку), щоб не опрацьовувати вже закодовані раніше файли знову, як було описано раніше, результати записують у файл result.json кожні dump_every_n опрацьованих застосунків. Мною було виставлено значення в 5 додатків.

Загальну схему кодування додатків у вектори та збереження їх у вигляді JSON файлу представлено на Рисунку 4.

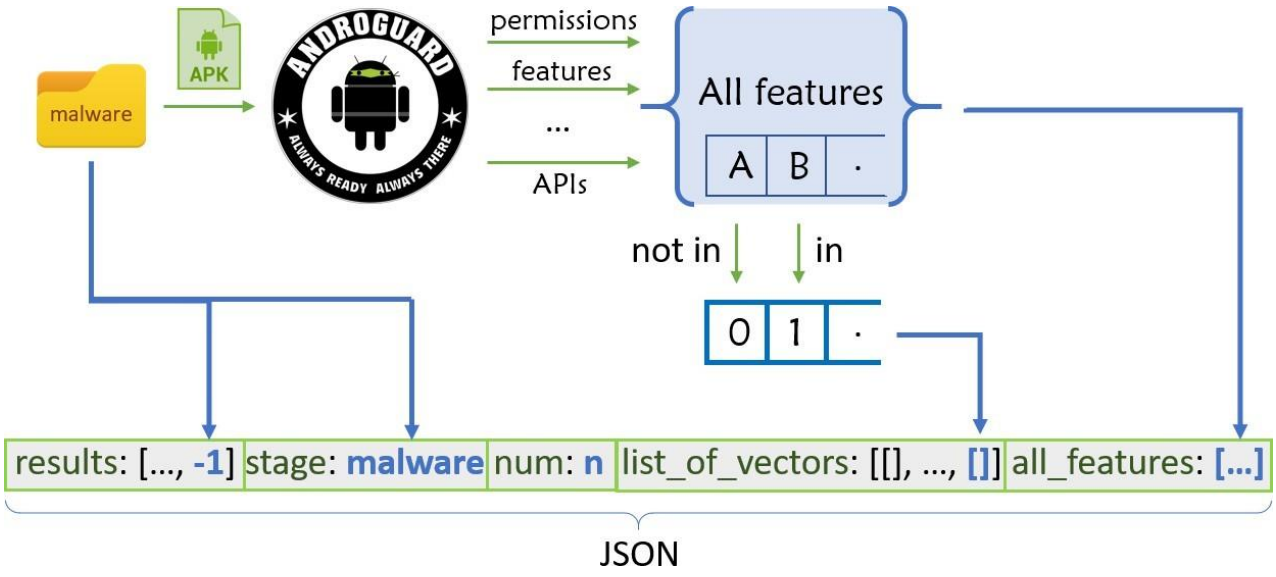


Рисунок 4.5 - Схема обробки Android додатків

4.3.4 Logistic Regression

Напишемо код, який демонструє процес оцінки моделі логістичної регресії (Logistic Regression) на навчальній та тестовій вибірках, а також обчислення різних метрик для моделі.

Спершу виконуємо прогнозування на навчальній та тестовій вибірках:

```
logreg_pred_train = logreg.predict(X_train)
logreg_pred_test = logreg.predict(X_test)
```

Потім виконуємо оцінку точності (accuracy) моделі на навчальній та тестовій вибірках :

```
logreg_train_score = accuracy_score(logreg_pred_train, y_train)
logreg_test_score = accuracy_score(logreg_pred_test, y_test)
print("Logistic Regression Train Score:", logreg_train_score)
print("Logistic Regression Test Score:", logreg_test_score)
```

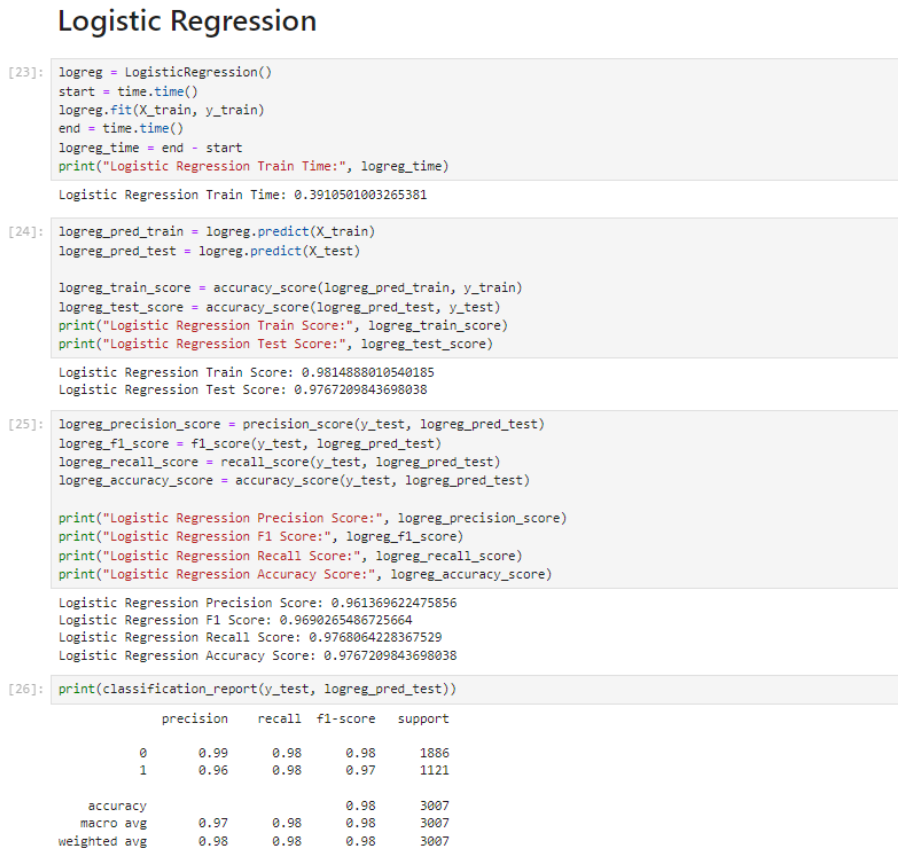
accuracy_score` вимірює частку правильних передбачень. Обчислення інших метрик для тестової вибірки виконується наступним чином:

```
logreg_precision_score = precision_score(y_test, logreg_pred_test)
logreg_f1_score = f1_score(y_test, logreg_pred_test)
logreg_recall_score = recall_score(y_test, logreg_pred_test)
logreg_accuracy_score = accuracy_score(y_test, logreg_pred_test)
print("Logistic Regression Precision Score:", logreg_precision_score)
print("Logistic Regression F1 Score:", logreg_f1_score)
print("Logistic Regression Recall Score:", logreg_recall_score)
print("Logistic Regression Accuracy Score:", logreg_accuracy_score)
```

precision_score вимірює частку правильних позитивних передбачень серед усіх позитивних передбачень. recall_score` вимірює частку правильних позитивних

передбачень серед усіх фактичних позитивних випадків. `f1_score` є гармонійним середнім між precision та recall.

В коді отримали такі результати:



Logistic Regression

```
[23]: logreg = LogisticRegression()
      start = time.time()
      logreg.fit(X_train, y_train)
      end = time.time()
      logreg_time = end - start
      print("Logistic Regression Train Time:", logreg_time)

      Logistic Regression Train Time: 0.3910501003265381

[24]: logreg_pred_train = logreg.predict(X_train)
      logreg_pred_test = logreg.predict(X_test)

      logreg_train_score = accuracy_score(logreg_pred_train, y_train)
      logreg_test_score = accuracy_score(logreg_pred_test, y_test)
      print("Logistic Regression Train Score:", logreg_train_score)
      print("Logistic Regression Test Score:", logreg_test_score)

      Logistic Regression Train Score: 0.9814888010540185
      Logistic Regression Test Score: 0.9767209843698038

[25]: logreg_precision_score = precision_score(y_test, logreg_pred_test)
      logreg_f1_score = f1_score(y_test, logreg_pred_test)
      logreg_recall_score = recall_score(y_test, logreg_pred_test)
      logreg_accuracy_score = accuracy_score(y_test, logreg_pred_test)

      print("Logistic Regression Precision Score:", logreg_precision_score)
      print("Logistic Regression F1 Score:", logreg_f1_score)
      print("Logistic Regression Recall Score:", logreg_recall_score)
      print("Logistic Regression Accuracy Score:", logreg_accuracy_score)

      Logistic Regression Precision Score: 0.961369622475856
      Logistic Regression F1 Score: 0.9690265486725664
      Logistic Regression Recall Score: 0.9768064228367529
      Logistic Regression Accuracy Score: 0.9767209843698038

[26]: print(classification_report(y_test, logreg_pred_test))

                   precision    recall  f1-score   support

                0       0.99      0.98      0.98      1886
                1       0.96      0.98      0.97      1121

         accuracy                           0.98      3007
        macro avg       0.97      0.98      0.98      3007
     weighted avg       0.98      0.98      0.98      3007
```

Рис. 4.6 Результати

Logistic Regression Train Score: 0.9814888010540185

Logistic Regression Test Score: 0.9767209843698038

Це показує високу точність як на навчальній, так і на тестовій вибірках, що свідчить про узгодженість моделі.

Модель логістичної регресії показує відмінні результати на обох вибірках, демонструючи високу точність, precision, recall і f1-score. Ці метрики вказують на те, що модель добре справляється із завданням класифікації додатків як шкідливих, так і безпечних.

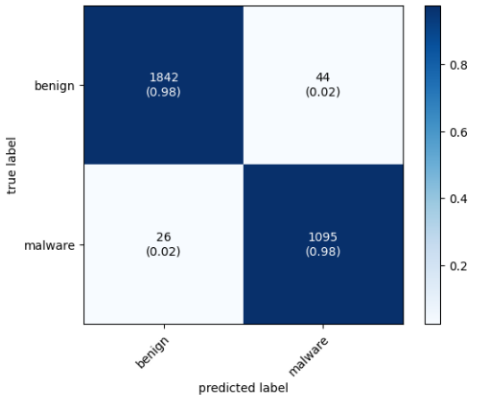


Рис. 4.7 Модель логістичної регресії

4.3.5 Random Forest Classifier

Розглянемо приклад створення, навчання та оцінки моделі `RandomForestClassifier` на даних, а також обчислення відповідних метрик.

Виконуємо імпорт необхідних бібліотек :

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

Створюємо екземпляр моделі RandomForestClassifier

rf = RandomForestClassifier()

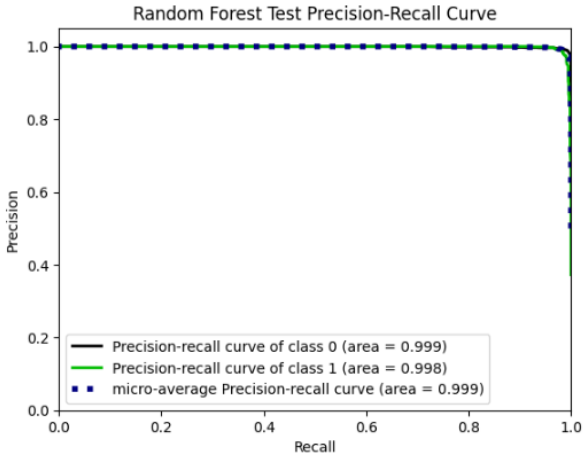


Рис.4.8 Екземпляр моделі

Рис. 4.9 Екземпляр моделі

Прогнозування на навчальній та тестовій вибірках :

rf_pred_train = rf.predict(X_train)

rf_pred_test = rf.predict(X_test)

Оцінка точності (accuracy) моделі на навчальній та тестовій вибірках :

rf_train_score = accuracy_score(rf_pred_train, y_train)

rf_test_score = accuracy_score(rf_pred_test, y_test)

print("Random Forest Train Score:", rf_train_score)

print("Random Forest Test Score:", rf_test_score)

```
XGBClassifier Precision Score: 0.9855725879170424
XGBClassifier F1 Score: 0.9802690582959641
XGBClassifier Recall Score: 0.9750223015165032
XGBClassifier Accuracy Score: 0.985367475889591
```

```
print(classification_report(y_test, xgb_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1886
           1       0.99      0.98      0.98      1121

    accuracy                           0.99      3007
   macro avg       0.99      0.98      0.98      3007
weighted avg       0.99      0.99      0.99      3007
```

```
xgb_cm = confusion_matrix(y_test, xgb_pred_test)
fig, ax = plot_confusion_matrix(conf_mat=xgb_cm, show_absolute=True, show_normed=True, colorbar=True, class_names=["benign", "malware"])
plt.show()
```

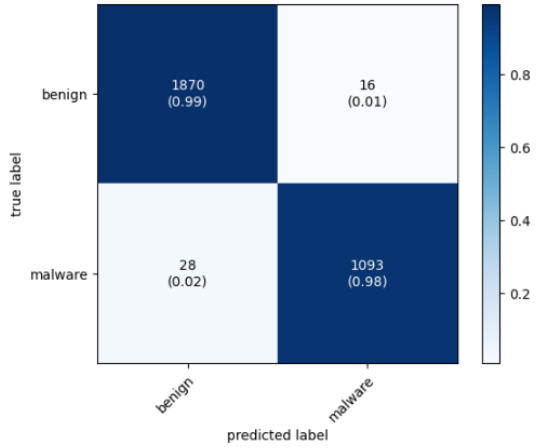


Рис. 4.10 Результати погнозування

Результати

```
labels = ["LR", "DT", "RF", "XGB"]
scores = [logreg_test_score, dt_test_score, rf_test_score, xgb_test_score]
```

```
def plot_model_plot(labels, scores):
    plt.figure()
    ax = sns.barplot(x=labels, y=scores)
    ax.set_title("Trained Models Accuracy")
    for container in ax.containers:
        ax.bar_label(container)
```

```
plot_model_plot(labels, scores)
plt.show()
```

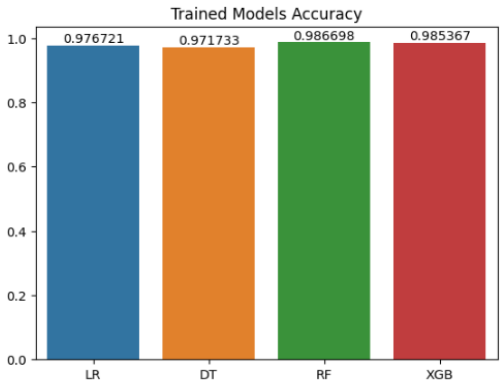


Рис. 4.11 Результати тренування мережі

Висновки

Таким чином, в ході виконання бакалаврської роботи розглянуто низку важливих питань, що пов’язані з безпекою Android-додатків. Зокрема, класифікацію характеристик безпеки, ефективні методи пошуку вразливостей, проблеми безпеки.

Розробка системи для виявлення вразливостей неможлива без вивчення організації системи Android, зокрема, відповыдних компонент, структури APK-файлів, процесів. В роботі було розглянуто реверс-інжиніринг APK файлів, зокрема JEB, IDA, Xposed Hook, Frida Hook. Особлива увага приділялась питанням APK Unpacking.

Практичним результатом роботи є розробка архітектури та навчання спеціальної нейронної мережі, для аналізу вразливостей APK-файлів.

Було проведпено експериментальне дослідження мережі. Воно показало високу точність як на навчальній, так і на тестовій вибірках, що свідчить про узгодженість моделі.

Модель логістичної регресії показує відмінні результати на обох вибірках, демонструючи високу точність, precision, recall і f1-score. Ці метрики вказують на те, що модель добре справляється із завданням класифікації додатків як шкідливих, так і безпечних.

Список використаних джерел

1. Number of smartphone users worldwide від 2016 to 2026. [Електронний ресурс]. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. -

2. Number of smartphones sold to end users worldwide від 2007 to 2021. [Електронний ресурс]. URL: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. - (Дата звернення: 10.05.2021).

3. Mobile operating systems' market share worldwide від January 2012 to January 2021. [Електронний ресурс]. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. -

4. Число доступних applications в Google Play Store з грудня 2009 до грудня 2020. [Електронний ресурс]. URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

5. 10 Google Play Apps Found Containing Banking Malware. [Електронний ресурс]. URL: https://www.infosecurity-magazine.com/news/ten-google-play-apps-banking/. –

6. Google: Android Malware Threat є Vastly Exaggerated. [Електронний ресурс]. URL: https://www.infosecurity-magazine.com/news/google-android-malware-threat-is-vastly/. -

7. AV-TEST Malware. [Електронний ресурс]. URL: https://www.av-test.org/en/statistics/malware/. -

8. Павленко Є. Ю., Дремов А. С. Виявлення шкідливих ділянок коду Android-додатків на основі аналізу графів потоків управління та графіків потоків даних // Проблеми інформаційної безпеки. Комп'ютерні системи. - 2017. - №. 2. - С. 109-126.

9. Павленко Є. Ю., Ігнатьєв Г. Ю., Зегжда П. Д. Статичний аналіз безпеки Android-додатків // Проблеми інформаційної безпеки. Комп'ютерні системи. - 2017. - №. 4. - С. 73-79.

10. Venugopal D., Hu G. Efficient signature based malware detection on mobile devices //Mobile Information Systems. - 2008. - Т. 4. - №. 1. - С. 33-49.

11. Павленко Є. Ю., Ярмак А. В., Москвин Д. А. Застосування методів кластеризації для аналізу безпеки Android-додатків // Проблеми інформаційної безпеки. Комп'ютерні системи. - 2016. - №. 3. - С. 119-126.

12. Chen T. та ін. TinyDroid: lightweight і efficient model для Android malware detection and classification //Mobile information systems. - 2018. - Т. 2018.

13. Павленко Є. Ю., Ігнатьєв Г. Ю. Виявлення шкідливих Android-додатків з використанням згорткової нейронної мережі // Проблеми інформаційної безпеки. Комп'ютерні системи. - 2018. - №. 3. - С. 107-119.

14. 13. Павленко Є. Ю., Суслов С. М. Виявлення шкідливих додатків для операційної системи Android з використанням капсульної нейронної мережі // Проблеми інформаційної безпеки. Комп'ютерні системи. - 2019. - №. 1. - С. 100-111.

15. Wu DJ et al. Droidmat: Android malware detection через manifest і api calls tracing //2012 Seventh Asia Joint Conference on Information Security. - IEEE, 2012. - С. 62-69.

16. Arp D. та ін. Drebin: Effective і explainable detection android mal-ware в вашому пакеті //Ndss. - 2014. - Т. 14. - С. 23-26.

17. Yang W. та ін. Appcontext: Diferentiating malicious and benign mobile app behaviors using context //2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. - IEEE, 2015. - Т. 1. - С. 303-313.

18. Gascon H. та ін. Structural detection of android malware за допомогою embedded call graphs //Proceedings of the 2013 ACM workshop on Artificial intelligence and security. - 2013. - С. 45-54.

19. Arzt S. et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps //Acm Sigplan Notices. – 2014. – Т. 49. – №. 6. – С. 259-269.

20. Octeau D. et al. Composite constant propagation: Application to android inter-component communication analysis //2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. – IEEE, 2015. – Т. 1. – С. 77-88.

21. Malware categories. [Электронный ресурс]. URL: https://developers.google.com/android/play-protect/phacategories. – (дата обращения: 12.05.2021).

22. Schmidt A. D. et al. Static analysis of executables for collaborative mal-ware detection on android //2009 IEEE International Conference on Communications. – IEEE, 2009. – С. 1-5.

23. Schmidt A. D. et al. Enhancing security of linux-based android devices //Proceedings of 15th International Linux Kongress. – Lehmann, 2008. – С. 1-16.

24. Bläsing T. et al. An android application sandbox system for suspicious software detection //2010 5th International Conference on Malicious and Unwanted Software. – IEEE, 2010. – С. 55-62.

25. Burguera I., Zurutuza U., Nadjm-Tehrani S. Crowdroid: behavior-based malware detection system for android //Proceedings of the 1st ACM workshop on Se-curity and privacy in smartphones and mobile devices. – 2011. – С. 15-26.

26. Shabtai A. et al. “Andromaly”: a behavioral malware detection framework for android devices //Journal of Intelligent Information Systems. – 2012. – Т. 38. – №. 1. – С. 161-190.

27. Zhou Y., Jiang X. Dissecting android malware: Characterization and evo-lution //2012 IEEE symposium on security and privacy. – IEEE, 2012. – С. 95-109.

28. Sahs J., Khan L. A machine learning approach to android malware detec-tion //2012 European Intelligence and Security Informatics Conference. – IEEE, 2012. – С. 141-147.

29. Saracino A. et al. Madam: Effective and efficient behavior-based android malware detection and prevention //IEEE Transactions on Dependable and Secure Computing. – 2016. – Т. 15. – №. 1. – С. 83-97.

30. Li C. et al. Android malware detection based on factorization machine //IEEE Access. – 2019. – Т. 7. – С. 184008-184019.

31. Arora A., Peddoju S. K., Conti M. Permpair: Android malware detection using permission pairs //IEEE Transactions on Information Forensics and Security. – 2019. – T. 15. – C. 1968-1982.

32. Mahindru A., Sangal A. L. MLDroid—framework for Android malware detection using machine learning techniques //Neural Computing and Applications. – 2021. – T. 33. – №. 10. – C. 5183-5240.

33. Rendle S. Factorization machines //2010 IEEE International Conference on Data Mining. – IEEE, 2010. – C. 995-1000.

34. Rendle S., Schmidt-Thieme L. Pairwise interaction tensor factorization for personalized tag recommendation //Proceedings of the third ACM international conference on Web search and data mining. – 2010. – C. 81-90.

35. Juan Y. et al. Field-aware factorization machines for CTR prediction //Proceedings of the 10th ACM conference on recommender systems. – 2016. – C. 43-50.

36. Li Y. et al. Android malware clustering through malicious payload mining //International symposium on research in attacks, intrusions, and defenses. – Springer, Cham, 2017. – C. 192-214.

37. Allix K. et al. Androzoo: Collecting millions of android apps for the re-search community //2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). – IEEE, 2016. – C. 468-471.

38. Rendle S. Factorization machines with libfm //ACM Transactions on In-telligent Systems and Technology (TIST). – 2012. – T. 3. – №. 3. – C. 1-22.

39. Bayer I. fastfm: A library for factorization machines //The Journal of Ma-chine Learning Research. – 2016. – T. 17. – №. 1. – C. 6393-6397.

40. Get Started with xLearn ! [Электронный ресурс]. URL: https://xlearn-doc.readthedocs.io/en/latest/python_api/index.html. – (дата обращения: 15.05.2021).

41. Lashkari A. H. et al. Toward developing a systematic approach to generate benchmark android malware datasets and classification //2018 International Carnahan Conference on Security Technology (ICCST). – IEEE, 2018. – C. 1-7.

42. APK datasets for machine learning. [Электронный ресурс]. URL: https://github.com/maxherobrine/android_apk_datasets/. – (дата обращения: 10.05.2021).

43. Zhao C. et al. Quick and accurate android malware detection based on sensitive APIs //2018 IEEE International Conference on Smart Internet of Things (SmartIoT). – IEEE, 2018. – С. 143-148.

44. Sandeep H. R. Static analysis of android malware detection using deep learning //2019 International Conference on Intelligent Computing and Control Sys-tems (ICCS). – IEEE, 2019. – С. 841-845.

45. Zhao C., Wang C., Zheng W. Android malware detection based on sensi-tive permissions and APIs //International Conference on Security and Privacy in New Computing Environments. – Springer, Cham, 2019. – С. 105-113.

46. Wu S. et al. Effective detection of android malware based on the usage of data flow APIs and machine learning //Information and software technology. – 2016. – Т. 75. – С. 17-25.