МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
Кафедра Комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
Аліна САВЧЕНКО
«_____» _____ 2022р.

КВАЛІФІКАЦІЙНА РОБОТА

(ДИПЛОМНА РОБОТА, ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ **"МАГІСТР"**
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ
"ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ"

Тема: "Засоби протидії викраденню даних в мобільному додатку для ОС Андроїд"

Виконавець: студент групи УС-211М Шемелюк Орест Андрійович

Керівник: к.т.н., доцент Харченко Олександр Григорович

Нормоконтролер: Ігор РАЙЧЕВ

Київ – 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *Кібербезпеки, комп'ютерної та програмної інженерії*

Кафедра *Комп'ютерних інформаційних* технологій

Галузь знань, спеціальність, освітньо-професійна програма: 12 "Інформаційні технології", 122 "Комп'ютерні науки", "Інформаційні управляючі системи та технології"

ЗАТВЕРДЖУЮ
Завідувач випускової кафедри
_____ Аліна САВЧЕНКО
«____»_____2022р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Шемелюка Ореста Андрійовича
(П.І.Б. випускника)

1. **Тема роботи:** «Засоби протидії викраденню даних в мобільному додатку для ОС Андроїд» затверджена наказом ректора від 26.09.2022р. за № 1774/ст
2. **Термін виконання роботи:** 26.09.2022 – 21.11.2022
3. **Вихідні дані роботи:** визначення типів операціних систем на мобільні телефони, їхня відмінність та популярність. Актуальність безпеки персональних даних, способи використання зловмисниками цих даних, типи вразливостей мобільних операційних систем та мобільних додатків.
4. **Зміст пояснювальної записки:** Операційна система Андроїд, її переваги над іншими ОС та популярність. Недоліки ОС Андроїд, типи вразливостей операційної системи. Підхід до розробки програмного забезпечення в команді для уникнення спеціального чи не спеціального вкраплення вразливого коду в мобільний додаток. Перелік порад для написання та підтримки мобільного додатку, щоб запезпечити його від взлому та витоку даних.
5. **Перелік обов'язкового ілюстративного матеріалу:** рисунки, діаграми, графіки, що відображають типи вразливостей ОС Андроїд, їхню популярність та складність.

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Виконання детального аналізу літератури та Інтернет-джерел згідно із темою роботи.	26.09.2022 – 30.09.2022	
2	Розробка та затвердження плану дипломної роботи.	11.10.2022 – 15.10.2022	
3	Огляд вразливостей мобільних додатків до взлому та викраденню даних.	16.10.2022 – 18.10.2022	
4	Розробка шаблонного додатку з найкращими методами протидії викраденню персональних даних.	19.10.2022 – 29.10.2022	
5	Опис порад для забезпечення безпечного користування додатками та його розробкою.	30.10.2022 – 30.11.2022	
6	Розробка звітності.	24.10.2022 – 30.10.2012	
7	Технічне оформлення пояснювальної записки.	01.11.2022 – 08.11.2012	
8	Підготовка до захисту дипломної роботи.	10.11.2022 – 10.11.2022	

7. Дата видачі завдання: «26» вересня 2022 р.

Керівник дипломної роботи _____ Олександр Харченко
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Орест Шемелюк
(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Засоби протидії викраденню даних в мобільному додатку для ОС Андроїд» складається зі вступу, чотирьох розділів, висновку, списку бібліографічних посилань і містить 82 сторінки та 28 рисунок. Список бібліографічних джерел скаладається з 47 найменувань.

Ключові слова: МОБІЛЬНІ ТЕХНОЛОГІЇ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РЕФАКТОРИНГ, ПАТЕРНИ РОЗРОБКИ, МОБІЛЬНІ ПРОЕКТИ, ГЛОБАЛЬНА ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ІНТЕГРОВАНА РОЗРОБКА ЗАСТОСУНКІВ, ЗАПОБІГАННЯ ВИКРАДЕНЮ ДАНИХ.

Актуальність. Нині важко переоцінити актуальність проблеми зі збереженням персональних даних від можливого попадання до рук зловмисників, які можуть використати будь-яку інформацію проти нас.

Мета дипломної роботи полягає в дослідженні вразливостей мобільних додатків під час користування кінцевими користувачами та розробки програмістами.

Основним завданням є створеня набору порад для користувачів та розробників, які допомагають захистити свої дані під час використання та розробки мобільного додатку.

Об’єктом дослідження даної дипломної роботи є розроблений мобільний додаток з використанням найкращих практик захисту інформації.

Предметом дослідження є способи взлому мобільного додатку за допомогою вразливостей в методах розробки ПЗ та у операційній системі.

Для того щоб досягти поставленої цілі дипломного проекту допомогла обробка літературних джерел, відео матеріалів та досвід використання бепечних технологій великомасштабними ІТ-компаніями.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ПЗ	Програмне забезпечення
ПП	Програмний продукт
ЦК	Центральна команда
КР	Командний рівень
МКР	Міжкомандний рівень
CI	Впровадження коду
CD	Постійне впровадження код

ВСТУП

Актуальність даної теми важко переоцінити на сьогоднішній день, адже збереження персональних даних є нагальною проблемою в усіх областях життя людини. Наскільки важливо зберігати особисті дані в таємниці? Точніше кажучи, наскільки небезпечно залишати свою особисту інформацію — своє ім'я, номер соціального страхування, кредитну картку та фінансові дані, навіть адресу електронної пошти та номер телефону — там, де інші люди можуть отримати до них доступ? Хоча може здатися нешкідливим залишати фрагменти особистої інформації в Інтернеті або фізичний документ, який викинули в смітник, це може бути використано для викрадення вашої особи.

Це сумна реальність крадіжки особистих даних. Незважаючи на всі звіти про те, як люди втрачають гроші та навіть репутацію через шахрайство та злам облікових записів, багато людей досі не до кінця розуміють реальні наслідки для безпеки викрадення їхньої особи, представленої обліковими даними та інформацією.

Із постійним розвитком інтелектуальних пристроїв, Інтернету речей (IoT) і того, як звичайні люди використовують Інтернет, дані, залишені мобільними пристроями, онлайн-браузерами та платіжними платформами. Поки це відбувається, ми ставатимемо все більш вразливими до крадіжки особистих даних та інших онлайн-загроз. Єдине, що обмежує технологію, — це етичні та правові обмеження, що робить конфіденційність невід'ємною проблемою для багатьох користувачів. Але чи справді ми турбуємося про свою конфіденційність і чи можемо ми справді отримати вигоду від зміни наших розслаблених звичок в Інтернеті?

Як відбувається крадіжка особистих даних? Крадіжка особистих даних відбувається, коли нашу особисту інформацію викрадають і використовують кіберзлочинці або шахраї, щоб видати себе за вас. Ваші облікові дані можна по суті використовувати для отримання доступу до різних сфер нашого цифрового життя, включаючи банківські рахунки, соціальні мережі та дані кредитної картки. Попутно вони можуть зібрати інші конфіденційні дані з наших облікових записів, а також тих, що належать нашій родині, друзям і колегам. У деяких випадках кіберзлочинці

можуть використовувати наші облікові дані, щоб завдати шкоди нашій репутації або викликати публічне приниження в Інтернеті. Разом вони можуть завдати непоправної шкоди. Крім того, викрадену особу також можна використовувати як ефективне прикриття для кіберзлочинців.

Крадіжка особистих даних зазвичай здійснюється з метою фінансової вигоди. Викрадач особистих даних, якому вдається заволодіти кредитною карткою або номером соціального страхування жертви, може використовувати їх для здійснення покупок або відкриття рахунків, використовуючи викрадені особисті дані. Це також можна зробити за допомогою облікового запису, який більшість людей не має проблем з наданням: адреса електронної пошти.

Оскільки для більшості облікових записів в Інтернеті потрібна адреса електронної пошти як ім’я користувача, а також спосіб перевірки облікових записів і скидання паролів, зламаний обліковий запис електронної пошти може відкрити для зловмисника ряд інших облікових записів. Облікові записи в соціальних мережах можуть бути використані для шкоди репутації, і якщо зловмисник отримає доступ до банківських або торгових облікових записів жертви, то він зможе викрати кошти.

На основі цього, в зв’язку широким використанням мобільних девайсів, була поставлена мета дипломної роботи, яка полягає в дослідженні вразливостей мобільних додатків під час користування кінцевими користувачами та розробки програмістами.

Основним завданням є створеня набору порад для користувачів та розробників, які допомагають захистити свої дані під час використання та розробки мобільного додатку. Об’єктом даної дипломної роботи є розроблений мобільний додаток з використанням найкращих практик захисту інформації.

Для того щоб досягти поставленої цілі дипломного проекту допомогла обробка літературних джерел, відео матеріалів та досвід використання бепечних технологій великомасштабними IT-компаніями.

Дипломна робота складається з чотирьох розділів.

У першому розділі розглянуто теоретичні відомості про операційні системи мобільних пристроїв, їхню популярність а також ступінь захищеності. Обрано найбільш вживану ОС.

Другий розділ описує середовище розробки ПЗ для мобільних додатків на ОС Андроїд. Також розглянуто мови програмування для нативної розробки. Методи тестування додатків на реальних та віртуальних девайсах, відмінності та переваги кожного способу.

У третьому розділі ми розглянули способи протидії викраденню даних на рівні команди розробників ПЗ, тобто саме під час розробки додатку, а не під час використання його кінцевим користувачем. Описали способи взаємодії команди всередені, задля запобігання таких випадків, щоб створити якісний і безпечний продукт.

У четвертому розділі розглянуто особливості створення мобільного додатку з точки зору коду. Описані найбільш часті проблеми по викраденню персональних даних з мобільного додатку встановленого на смартфон користувача, їхні причини всередині коду, а також методи протидії несанкціонованому доступу до персональних даних.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА МОБІЛЬНИХ ОПЕРАЦІЙНИХ СИСТЕМ, ВІДМІННІСТЬ МІЖ НИМИ ТА ЗАХИЩЕНІСТЬ

Операційна система діє як посередник між користувачем комп'ютера та комп'ютерним обладнанням. Метою операційної системи є створення середовища, в якому користувач може зручно та ефективно виконувати програми.

Операційна система — це програмне забезпечення, яке керує апаратним забезпеченням комп'ютера. Апаратне забезпечення повинно забезпечувати відповідні механізми для забезпечення правильної роботи комп'ютерної системи та запобігання втручанню програм користувача в належну роботу системи.

В цьому розділі хотілося б розглянути типи операційних систем, їхні характеристики та популярність. Особливу увагу потрібно звернути на мобільні ОС такі як Андроїд та iOS, поглибити свої знання в них та зрозуміти чому одна ОС більш популярна за іншу.

1.1. Види операційних систем

Операційна система (ОС) — це програмне забезпечення, яке діє як інтерфейс між апаратними компонентами комп'ютера та користувачем. Кожна комп'ютерна система повинна мати принаймні одну операційну систему для запуску інших програм. Такі програми, як браузери, MS Office, блокнот ігор тощо, потребують середовища для роботи та виконання своїх завдань.

ОС допомагає нам спілкуватися з комп'ютером, не знаючи мови комп'ютера. Користувач не може використовувати будь-який комп'ютер або мобільний пристрій без операційної системи.

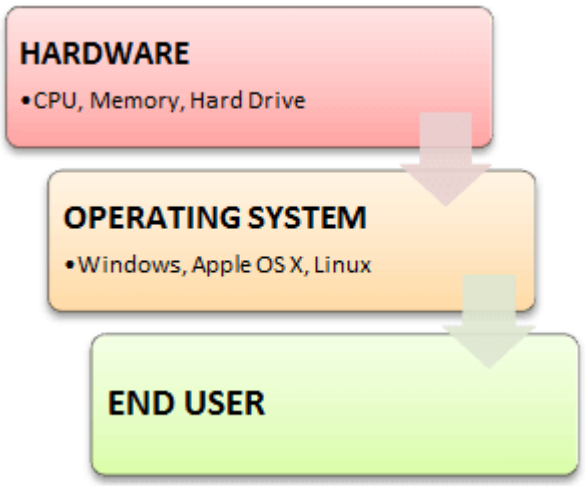


Рис. 1.1. Взаємодія заліза з користувачем через ОС

Операційні системи були вперше розроблені наприкінці 1950-х років для керування стрічковим зберіганням. Дослідницька лабораторія General Motors впровадила першу ОС на початку 1950-х років для свого IBM 701. У середині 1960-х років операційні системи почали використовувати диски.

Наприкінці 1960-х років була розроблена перша версія ОС Unix. Першою ОС, створеною Microsoft, була DOS. Він був створений у 1981 році шляхом придбання програмного забезпечення 86-DOS у компанії з Сіетла. Сучасна популярна ОС Windows вперше з'явилася в 1985 році, коли був створений графічний інтерфейс користувача, який поєднався з MS-DOS.

Деякі комп'ютерні процеси є дуже тривалими та трудомісткими. Щоб пришвидшити той самий процес, завдання з подібними типами потреб об'єднуються разом і виконуються як група.

Користувач пакетної операційної системи ніколи безпосередньо не взаємодіє з комп'ютером. У цьому типі ОС кожен користувач готує свою роботу на автономному пристрої, як на перфокарті, і надсилає її оператору комп'ютера.

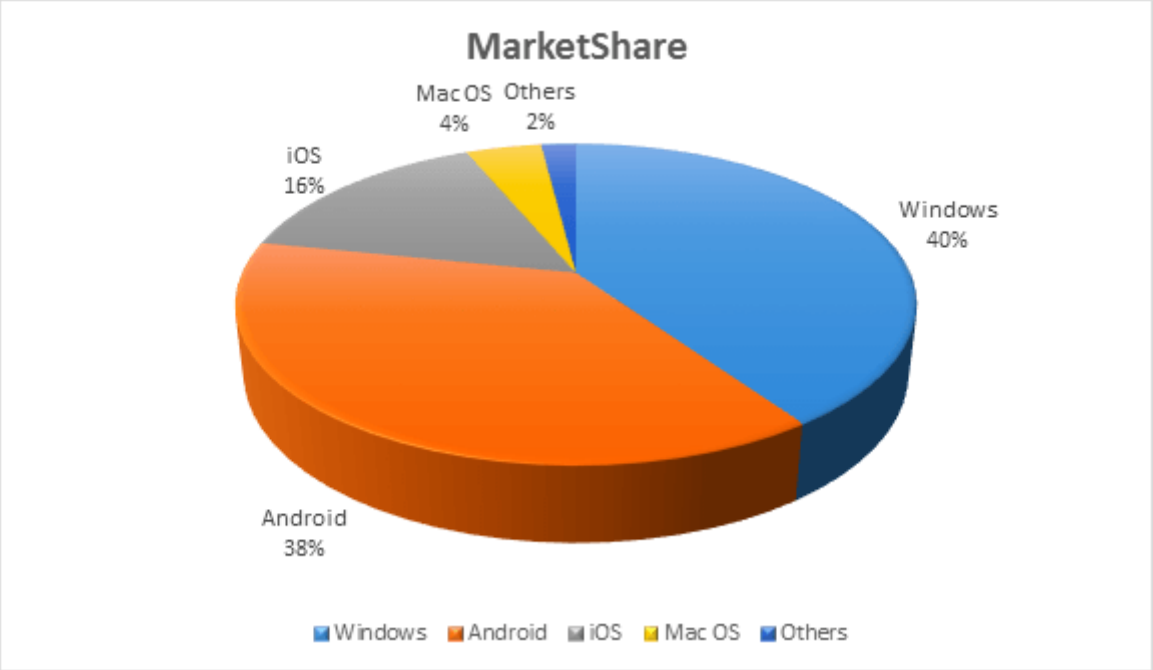


Рис. 1.2. Частка ОС на світовому ринку

Операційна система з розподілом часу дозволяє людям, які знаходяться на різних терміналах (оболонці), використовувати одну комп'ютерну систему одночасно. Процесорний час (CPU), який розподіляється між кількома користувачами, називається розподілом часу.

ОС реального часу. Часовий інтервал операційної системи реального часу для обробки та відповіді на вхідні дані дуже малий. Приклади: військові системи програмного забезпечення, космічні системи програмного забезпечення є прикладом ОС реального часу.

Розподілені системи використовують багато процесорів, розташованих на різних машинах, щоб забезпечити дуже швидкі обчислення своїм користувачам.

Мережева операційна система працює на сервері. Він надає можливість керувати даними, користувачами, групами, безпекою, додатками та іншими мережевими функціями.

Мобільні операційні системи – це ті ОС, які спеціально розроблені для живлення смартфонів, планшетів і пристроїв, що носяться.

Одними з найвідоміших мобільних операційних систем є Android та iOS, але інші включають BlackBerry, Web і watchOS.

Розглянемо найбільш популярні мобільні ОС як Android та iOS.

1.1.1. Операційна система iOS

iOS - це операційна система, розроблена компанією Apple.inc. iOS працює на всіх мобільних пристроях Apple. IOS є другою за величиною операційною системою в світі після Android.

IOS використовує мультисенсорний інтерфейс, в якому пристрій управляється за допомогою простих текстур. Прості жести, як-от переведення пальців у верхній частині екрана пристрою. І щипки пальцем, щоб збільшити екран. Ви можете виконувати всю цю роботу вільно на iOS.

IOS робить датчики свого пристрою дуже потужними. Це полегшує вашу роботу, миттєво виявляючи кінчики пальців. Xeos контролює всі аспекти апаратного забезпечення пристрою Apple. Крім того, він обробляє всі функції програмного забезпечення. Це означає, що iOS надає повний контроль своїм користувачам. Де програмне забезпечення вміло поєднується з апаратним забезпеченням. Що дає користувачеві кращу продуктивність апаратного забезпечення пристрою.

Понад 2 мільйони програм для iOS доступні для завантаження в Apple App Store. Це відомий магазин додатків для всіх пристроїв Apple.

У 2005 році, коли Стів Джобс почав планувати iPhone, у нього було два варіанти: перший — зменшити Mac. Це робочий стіл Macintosh компанії Apple, а другий був для збільшення iPod. Щоб вирішити цю проблему, вони зустрілися з командою, яка створила Mac і iPod, а потім вирішили створити iOS для iPhone.

Рис. 1.3. Зображення IPhone з ОС iOS

Нова операційна система була випущена з iPhone у січні 2006 року. На момент випуску iPhone операційна система називалася iPhone OS. Спочатку на пристрої в ОС iPhone не було дозволено запускати програми сторонніх розробників. Ідея Стіва Джоба полягала в тому, що розробники додатків можуть розробляти веб-додатки через веб-браузер Safari.

У жовтні 2008 року Apple оголосила про оригінальний набір програмного забезпечення, що розробляється для SD. 8 березня 2007 року iPhone SD був готовий. Магазин IOS Apps відкрився 10 липня 2007 року. В якому спочатку було всього 500 додатків. Але з вересня 2007 по 2014 рік його кількість зросла до 2,2 мільйона. Разом ці додатки завантажили понад 130 мільярдів разів. У вересні 2008 року Apple анонсувала iPod, а в січні 2010 року — iPad, який мав більший екран, ніж iPhone та iPod.

У червні 2010 року Apple змінила ОС iPhone на iOS. Раніше операційна система Apple не могла обробляти більше програм iPhone. Наразі основним програмним забезпеченням від Apple є iOS. Який працює на всіх моделях мобільних пристроїв iPhone, iPad, iPod touch і iPad mini. Те саме програмне забезпечення працює на смарт-годиннику Apple. Щоразу, коли Apple додає будь-які нові функції в iOS, це називається оновленням програмного забезпечення.

Apple продовжує щороку випускати нову версію iOS. Наразі останньою версією iOS є IOS 13.1, яка була випущена у вересні 2019 року. У цій новій версії було багато доповнено щодо покращення продуктивності та якості. Основною апаратною платформою для iOS є архітектура ARM. Усі пристрої iOS до iOS 4 можна запускати на пристрої лише з 32-бітним процесором ARM.

Але в 2013 році IOS перестав підтримувати повний 16-бітний процесор. Нова версія iOS 12 від Apple буде доступна на всіх пристроях, які працюють на 14-бітному процесорі. Це особливість iOS. Щоразу, коли до нього додаються нові функції, він негайно дозволяє оновлювати програмне забезпечення на всіх пристроях Apple. У той час як нова мобільна операційна система повинна чекати, поки вона потрапить на свій пристрій, навіть після того, як нові функції будуть ретранслюватися.

Безпека операційної системи ios:

1) Перед повним завантаженням iOS є низькорівневий код, який запускається із завантажувального ПЗУ. Його функція полягає в тому, щоб перевірити, що кореневий CA Apple підписаний відкритим ключем перед запуском низькорівневого завантажувача.

2) Secure Enclave може бути процесором кадрування, який є в пристрої iOS, який містить Touch ID або Face ID. це власний безпечний процес завантаження, щоб переконатися, що він повністю безпечний. До складу цього співпроцесора додатково входить апаратний генератор випадкових чисел. Secure Enclave кожного пристрою має унікальний ідентифікатор, який надається під час його створення та не може бути змінений.

3) Пристрої iOS можуть мати пароль, який не використовується для розблокування пристрою, зміни параметрів системи та шифрування вмісту пристрою.

4) Touch ID може бути сканером відбитків пальців, вбудованим у кнопку «Додому», і, можливо, не розблоковувати пристрій, робити покупки та входити в пристрій для виконання інших завдань.

5) iOS використовує функцію Execute Never (EN) архітектури ARM. Це дозволяє ASLR також як деякі частини пам’яті зупиняти атаки переповнення буфера, включаючи атаки повернення до бібліотеки.

6) Як згадувалося вище, одне використання шифрування в iOS – це пам’ять Secure Enclave. Коли на пристрої iOS використовується код доступу, вміст пристрою шифрується.

7) Двофакторна автентифікація є опцією в iOS, щоб переконатися, що якщо неавторизована особа знає комбінацію Apple ID і пароля, вона не зможе отримати доступ до облікового запису.

8) iOS підтримує TLS як з API низького, так і високого рівня для розробників. Коли Wi-Fi увімкнено, iOS використовує випадкову MAC-адресу, щоб ніхто не міг відстежити пристрій.

IPhone працює на цій системі. Ось чому всі можливості та функції iPhone дуже хороші та має хороший інтерфейс користувача. Завдяки цьому він забезпечує найкращу безпеку, тому ці смартфони дорогі.

1.1.2. Операційна система Linux

Так як операційна система Android побудована на базі Linux, варто спочатку розповісти про нього, щоб детальніше розуміти про що йде мова.

Linux® — це операційна система (ОС) з відкритим кодом. Операційна система — це програмне забезпечення, яке безпосередньо керує апаратним забезпеченням і ресурсами системи, як-от ЦП, пам’ять і сховище. ОС знаходиться

між програмами та апаратним забезпеченням і створює зв’язки між усім вашим програмним забезпеченням і фізичними ресурсами, які виконують роботу.

Linux був розроблений, щоб бути схожим на UNIX, але еволюціонував для роботи на різноманітному апаратному забезпеченні від телефонів до суперкомп’ютерів. Кожна ОС на базі Linux містить ядро Linux, яке керує апаратними ресурсами, і набір програмних пакетів, які складають решту операційної системи.

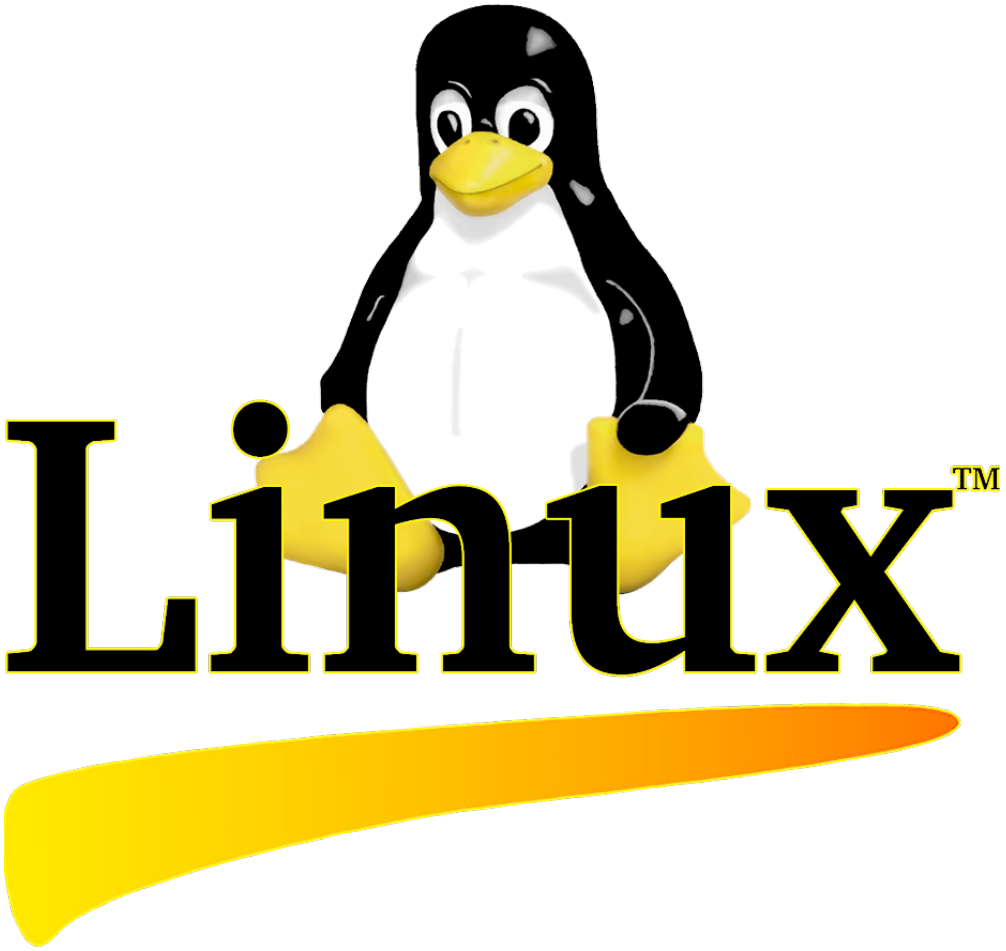


Рис. 1.4. Зображення емблеми Linux

ОС містить деякі загальні основні компоненти, як-от інструменти GNU, серед інших. Ці інструменти дають користувачеві можливість керувати ресурсами, які надає ядро, встановлювати додаткове програмне забезпечення, налаштовувати параметри продуктивності та безпеки тощо. Усі ці інструменти разом складають

функціональну операційну систему. Оскільки Linux є ОС з відкритим кодом, комбінації програмного забезпечення можуть відрізнятися в різних дистрибутивах Linux.

Командний рядок — це ваш прямий доступ до комп'ютера. Тут ви просите програмне забезпечення виконати апаратні дії, які графічний інтерфейс користувача (GUI) просто не може запитати.

Командні рядки доступні в багатьох операційних системах — пропрієтарних або відкритих. Але зазвичай це пов'язано з Linux, оскільки і командні рядки, і програмне забезпечення з відкритим кодом разом надають користувачам необмежений доступ до їхнього комп'ютера.

Linux — це безкоштовна операційна система з відкритим вихідним кодом, випущена згідно з GNU General Public License (GPL). Будь-хто може запускати, вивчати, змінювати та розповсюджувати вихідний код або навіть продавати копії свого зміненого коду, за умови, що він робить це за тією самою ліцензією.

Linux став найбільшим проектом програмного забезпечення з відкритим кодом у світі. Професійні програмісти та програмісти-любителі та розробники з усього світу роблять внесок у ядро Linux, додаючи функції, знаходячи та виправляючи помилки та недоліки безпеки, вносячи виправлення та надаючи нові ідеї — і все це водночас ділиться своїми внесками зі спільнотою.

Однією з найбільш цінних переваг Linux перед іншими платформами є високий рівень безпеки, який він забезпечує. Кожен користувач Linux із задоволенням працює в середовищі, вільному від вірусів, і використовує регулярний час для захисту від вірусів, необхідний під час роботи з іншими операційними системами, для інших більш важливих завдань.

Завдяки дистрибутиву з відкритим вихідним кодом Linux постійно розробляється й оновлюється спільнотою програмістів, що постійно розширюється. Незважаючи на свою динамічність, він повністю повний з точки зору функціональності та інтерфейсу. Усі ці поточні зусилля щодо розробки здійснюються з єдиною метою – підтримувати платформу гнучкою та постійно адаптуватися до мінливого «клімату».

1.1.3. Операційна система Android

ОС Android — це мобільна операційна система на базі Linux, яка в основному працює на смартфонах і планшетах.

Платформа Android включає операційну систему на основі ядра Linux, графічний інтерфейс користувача, веб-браузер і програми для кінцевого користувача, які можна завантажити. Незважаючи на те, що на початкових демонстраціях Android був стандартний смартфон QWERTY і великий екран VGA, операційна система була розроблена для роботи на відносно недорогих телефонах зі звичайними цифровими клавіатурами.

Рис. 1.5. Зображення ОС Android

Android було випущено за ліцензією Apache v2 з відкритим кодом; це дозволяє розробляти багато варіантів ОС для інших пристроїв, таких як ігрові консолі та цифрові камери. Android базується на програмному забезпеченні з відкритим вихідним кодом, але більшість пристроїв Android постачаються з попередньо

встановленим пакетом пропрієтарного програмного забезпечення, наприклад Google Maps, YouTube, Google Chrome і Gmail.

Android розпочав своє життя як стартап у Пало-Альто під назвою Android Inc. у 2003 році. Спочатку компанія мала намір розробити операційну систему для цифрових камер, але відмовилася від цих зусиль, щоб вийти на більш широкий ринок.

У 2005 році Google придбала компанію Android Inc. і її ключових співробітників щонайменше за 50 мільйонів доларів. Google продав ранню мобільну платформу виробникам мобільних телефонів і операторам мобільного зв'язку з такими її основними перевагами, як гнучкість і можливість оновлення.

Google непомітно розробляв ОС Android, коли Apple випустила iPhone у 2007 році. Попередні прототипи телефону Android дуже нагадували BlackBerry з фізичною клавіатурою та без сенсорного екрана. Однак запуск iPhone значно змінив ринок мобільних комп'ютерів і змусив розробників Android посилити підтримку сенсорних екранів. Незважаючи на це, HTC Dream, який був першим комерційно доступним смартфоном під управлінням ОС Android, мав клавіатуру QWERTY і був зустрінутий критично під час випуску в 2008 році.

Наприкінці 2007 року Open Handset Alliance (OHA) оголосив про своє створення. OHA була коаліцією з понад 30 апаратних, програмних і телекомунікаційних компаній, включаючи Google, Qualcomm, Broadcom, HTC, Intel, Samsung, Motorola, Sprint, Texas Instruments і японських бездротових операторів KDDI і NTT DoCoMo. Метою альянсу був внесок у розробку першої платформи з відкритим кодом для мобільних пристроїв.

Google випустив загальнодоступну бета-версію Android 1.0 для розробників приблизно в той самий час, коли було оголошено про альянс, у листопаді 2007 року. Лише коли Google випустив Android 1.5 у квітні 2009 року, Google представив фірмову схему іменування Android на тему десерту; Android 1.5 називався «Cupcake».

Інтерфейс Android за замовчуванням базується на прямих маніпуляціях, таких як торкання, переведення та зведення пальців для ініціювання дій. Пристрій надає

користувачеві тактильний зворотний зв’язок за допомогою сповіщень, наприклад вібрації, у відповідь на дії. Наприклад, якщо користувач натискає кнопку навігації, пристрій починає вібрувати.

Коли користувач завантажує пристрій, ОС Android відображає головний екран, який є основним центром навігації для пристроїв Android і складається з віджетів і піктограм програм. Віджети – це інформаційні дисплеї, які автоматично оновлюють такий вміст, як погода чи новини. Відображення головного екрана може відрізнятися залежно від виробника пристрою, на якому встановлена ОС. Користувачі також можуть вибирати різні теми для головного екрана за допомогою сторонніх програм у Google Play.

Рядок стану у верхній частині головного екрана відображає інформацію про пристрій та його підключення, наприклад мережу Wi-Fi, до якої підключено пристрій, або силу сигналу. Користувачі можуть одним рухом пальця опустити рядок стану, щоб переглянути екран сповіщень.

ОС Android також містить функції для економії використання акумулятора. ОС призупиняє роботу програм, які не використовуються, щоб заощадити заряд батареї та ЦП. Android містить функції керування пам’яттю, які автоматично закривають неактивні процеси, збережені в його пам’яті.

Android використовує ARM для своєї апаратної платформи; пізніші версії ОС Android підтримують архітектури x86 і x86-64. Починаючи з 2012 року, виробники пристроїв випускають Android-смартфони та планшети з процесорами Intel.

Мінімальні вимоги до апаратного забезпечення Android залежать від розміру екрана пристрою, а також типу та щільності ЦП. Спочатку Google вимагав процесор 200 МГц, 32 МБ пам’яті та 32 МБ оперативної пам’яті.

Google випускає документацію з вимогами до апаратного забезпечення, яким мають відповідати виробники оригінального обладнання (OEM), щоб пристрій був «Схвалений Google», що означає, що він постачатиметься з офіційними програмами Google. Однак природа Android з відкритим кодом означає, що він також може працювати на меншому апаратному забезпеченні, і навпаки.

Найсуттєвіша критика користувачів Android полягає в тому, що ОС фрагментована. Гнучка природа Android із відкритим вихідним кодом призводить до багатьох варіантів апаратного та програмного забезпечення. Багато пристроїв працюють на старих версіях Android. Станом на липень 2022 року 29,63% користувачів Android використовують операційну систему версії 11, 21,8% використовують версію 10, 20,86% використовують версію 12 і 10,74% використовують версію 9, за даними Statcounter.

Фрагментація пристроїв створює проблеми для розробників, оскільки важко розробляти програми, які працюють на всіх типах пристроїв і версіях. Фрагментація також є проблемою для бізнесу; IT-персонал не може легко захистити та керувати пристроями, які працюють на різноманітному апаратному та програмному забезпеченні. Google запустив Project Treble як потенційне рішення цієї проблеми. Ця ініціатива відокремлює ОС Android від модифікацій OEM і дозволяє швидше розгортати оновлення програмного забезпечення.

Інша критика ОС Android полягає в тому, що додатки Android можуть бути легко піратськими. Проте з випуском Android Jelly Bean Google запропонував розробникам можливість шифрувати платні програми.

1.2. Порівняння Android та iOS

Творці Android вважали, що ОС буде конкурувати з іншими мобільними операційними системами, такими як Symbian і Microsoft Windows Mobile.

Symbian була закритою ОС з мікроядром і інтерфейсом користувача, який забезпечував графічну оболонку. Багато виробників мобільних телефонів використовували Symbian OS, включаючи Nokia, Samsung і Motorola. Symbian була популярною ОС у всьому світі, але не набула великої популярності в Північній Америці. Однак дизайн Symbian був не таким простим, як Android та iOS, і програмувати ОС було важко. Розробка ОС Symbian була припинена в 2014 році.

Windows Mobile походить від Windows CE, вбудованої ОС, і вперше з’явилася на Pocket PC 2000. Корпорація Майкрософт продала мобільну ОС для бізнесу.

Конкуренція з боку Android та iOS змусила Microsoft внести зміни; компанія замінила Windows Mobile на Windows Phone у 2010 році, орієнтована на споживчий ринок. Microsoft поступово відмовилася від Windows Phone на користь Windows 10 Mobile, але ця ОС також була припинена; Microsoft оголосила про завершення свого життя 14 січня 2020 року.

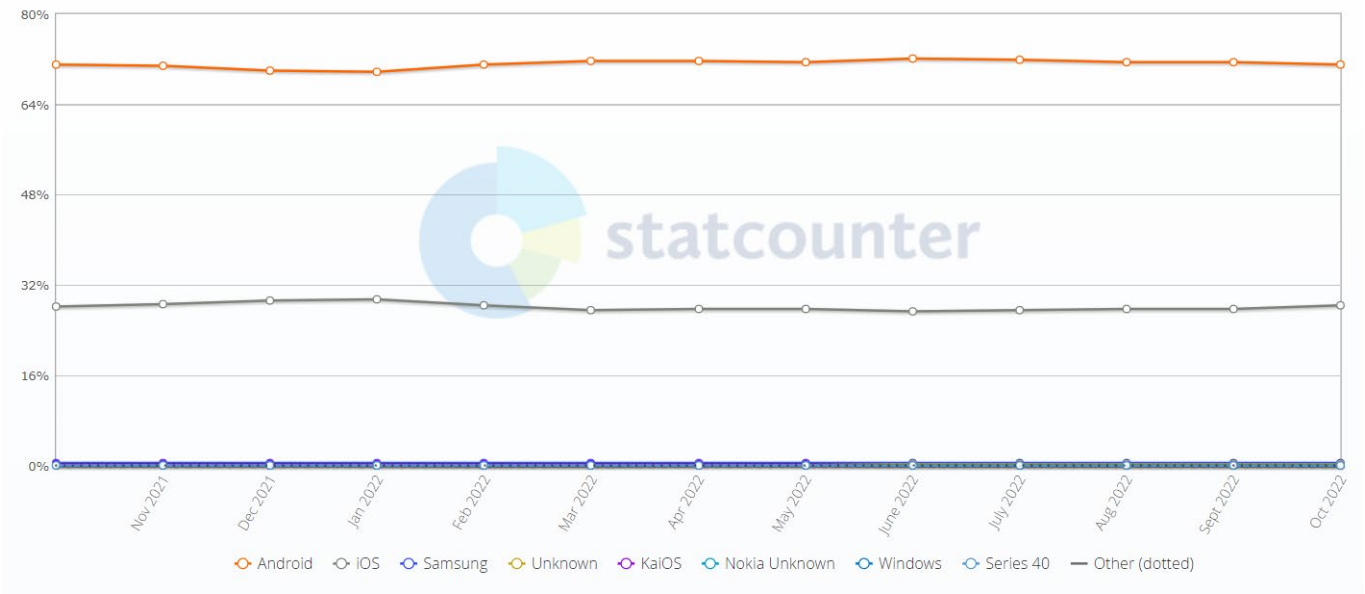


Рис. 1.6. Охоплення ринку ОС для мобільних платформ

Головним конкурентом Android є Apple iOS. І iOS, і ОС Android пропонують порівняльні функції. Apple iOS — це пропрієтарна ОС із фіксованим інтерфейсом, тоді як Android — це ОС з відкритим кодом, яка пропонує більше гнучкості та налаштувань.

Згідно зі звітом Statcounter, частка світового ринку Android у 2022 році становила 71,85%. Частка світового ринку Apple iOS склала 27,5%. У США, однак, Apple домінує на ринку з часткою 55,25%; Android претендує на 44,43%, за ним йдуть Samsung з 0,27% і Windows з 0,02%.

ВИСНОВКИ ДО РОЗДІЛУ 1

В даному розділі ми розглянули основні загальні поняття операційної системи, її особливості, принцип роботи та можливості. Було розглянуто детально про стоворення ОС iOS, її історію, а також особливості використання та рівень безпеки. Наступною операційною системою був Андроїд, який розроблявся спочатку для управління камерою, а згодом був розширений на мобільні девайси.

На основі розглянутих даних між мобільними операційними системами, їхньою популярністю, що Андроїд попри свої недоліки є найбільш використовуваною ОС в світі, з понад 73 відсотками користувачів, а також з своїми суттєвими недоліками в безпеці, хоча і був розроблений на основі Linux, який є доволі безпечним. Було вирішено вибрати Андроїд ОС для подальшого дослідження в даній науковій роботі та написання рекомендацій по розробці мобільних додатків для нього.

РОЗДІЛ 2. ЗАСОБИ РОЗРОБКИ ДОДАТКІВ ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ АНДРОЇД

Даний розділ описує засоби розробки програмного забезпечення для операційної системи Андроїд. Безперечно ви можете використовувати свої засоби досягнення мети, там як інтегроване середовище розробки або навіть свою систему контролю версій. Це не обовязково щоб ваші засоби, збігалися з даними які наведені тут, проте саме вони є рекомендовані при розробці від власника ОС Андроїд Google.

2.1. Інтегроване середовище розробки Android Studio

Android Studio — це інтегроване середовище розробки (IDE) для розробки програм під ОС Android. Android Studio надає безліч функцій, які підвищують продуктивність під час написання програм для ОС Android.

Android Studio було анонсовано 11 травня 2012 року на конференції Google I/O як офіційну IDE для розробки програм для ОС Android.

З 8 травня 2020 року Kotlin є мовою, якій Google віддає перевагу для розробки програм Android. Окрім цього, Android Studio підтримує інші мови програмування.

Особливості Android Studio

- Має гнучку систему збирання на основі Gradle.
- Має швидкий і багатофункціональний емулятор для тестування програм.
- Android Studio має консолідоване середовище, де ми можемо розробляти для всіх пристроїв Android.
- Застосування зміни до коду ресурсів нашої запущеної програми без перезапуску програми.

- Android Studio надає широкі інструменти та фреймворки тестування.
- Підтримує C++ і NDK.
- Забезпечує вбудовану підтримку Google Cloud Platform, що полегшує інтеграцію Google Cloud Messaging і App Engine.

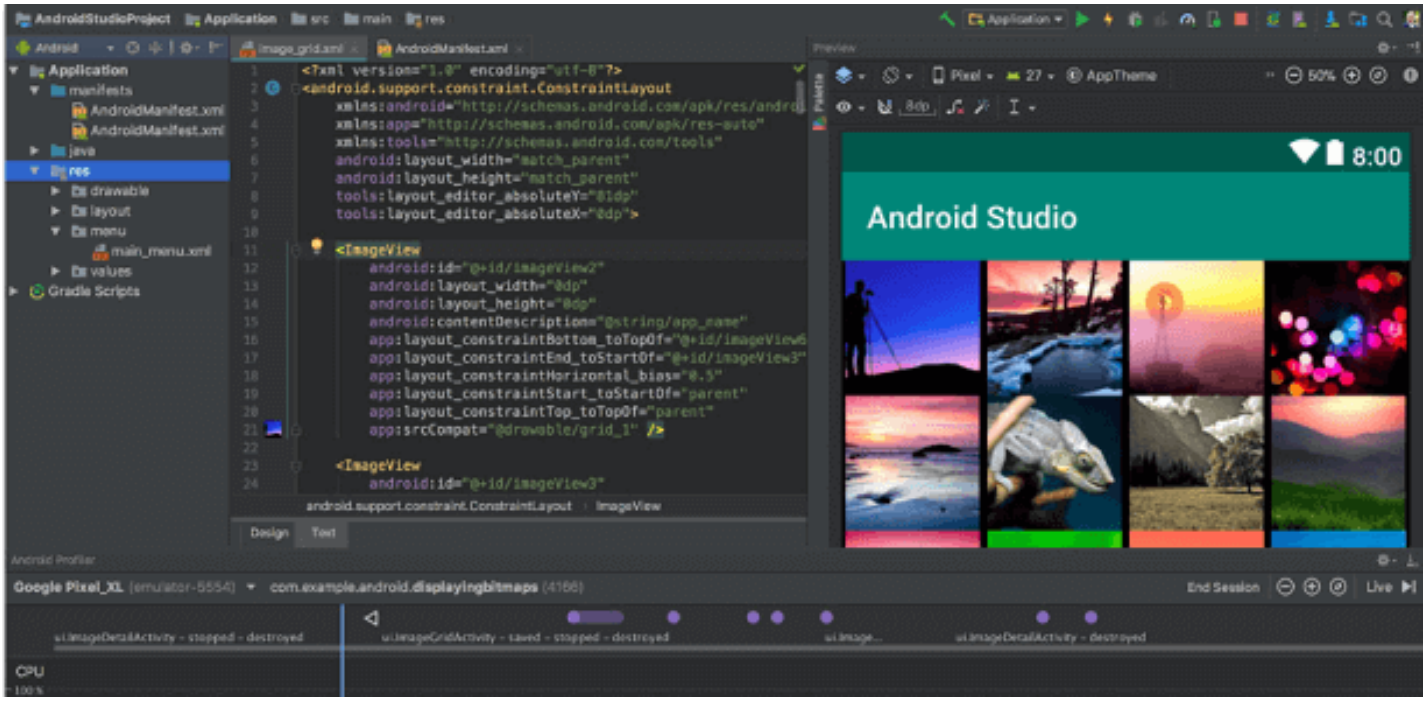


Рис 2.1 Android Studio

2.2. Android емулятор.

Емулятор Android — це віртуальний пристрій Android (AVD), який представляє певний пристрій на ОС Android. Ми можемо використовувати емулятор Android як цільовий пристрій для виконання та тестування нашої програми Android на нашому компютері. Емулятор Android забезпечує практично всі функції реального пристрою.

Ми можемо приймати вхідні телефонні дзвінки та текстові повідомлення. Він також вміє визначати місцезнаходження пристрою та імітує різні швидкості мережі,

можливість робити фото та відео знімки. Емулятор Android імітує обертання та інші апаратні датчики. Він має доступ до магазину Google Play та багато іншого.

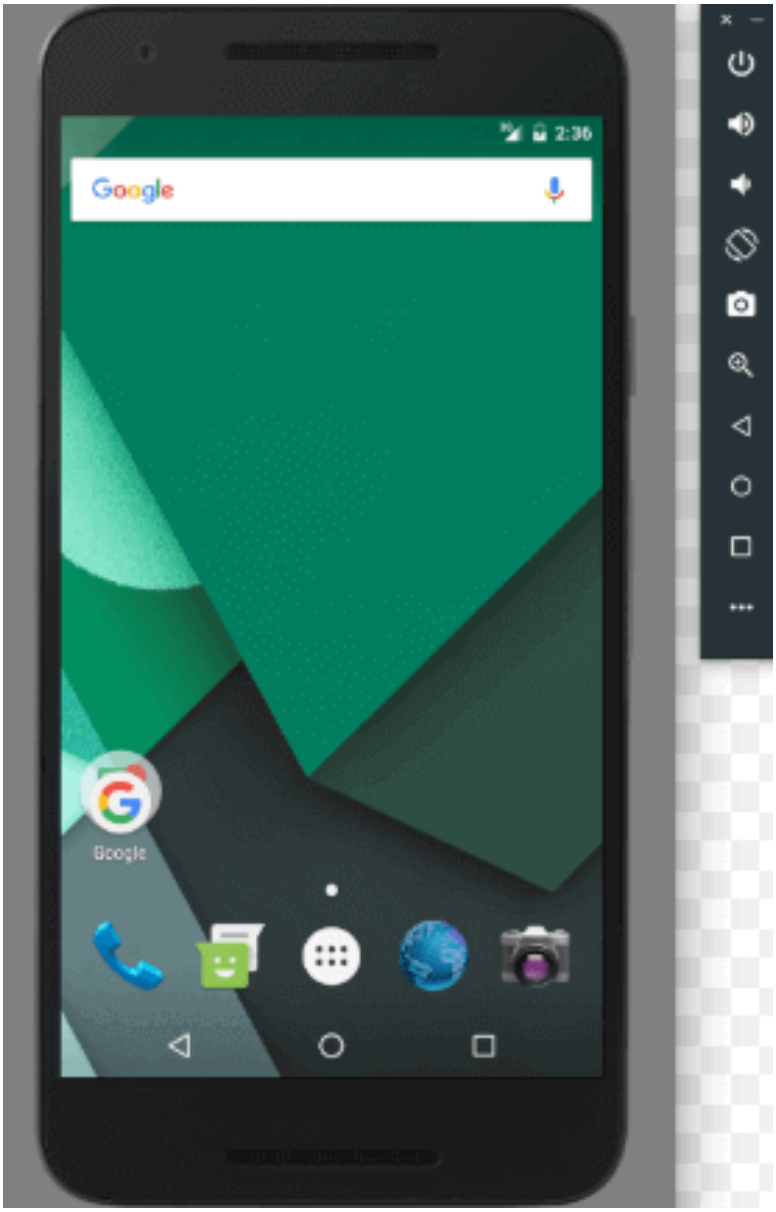


Рис 2.2. Емулятор

Тестування додатків Android на емуляторі іноді швидше та легше, ніж на реальному пристрої. Наприклад, ми можемо передавати дані на емулятор швидше, ніж на реальний пристрій, підключений через USB або Wi-Fi.

Емулятор на ОС Android поставляється з попередньо визначеними конфігураціями для кількох телефонів Android девайсів, Wear OS, планшетів, пристроїв Android TV.

2.3. Мови програмування під ОС Android.

2.3.1. Мова програмування Java

Мова програмування Java була розроблена Sun Microsystems на початку 1990-х років. Незважаючи на те, що Java в основному використовується для Інтернет-програм, вона є простою, ефективною мовою загального призначення. Java спочатку була розроблена спеціально для вбудованих мережевих програм, які працюють на певних платформах. Це портативна, об'єктно-орієнтована, інтерпретована мова.

Рис. 2.3. Емблема Java

Той самий Java-додаток працюватиме однаково на будь-якому комп'ютері, незалежно від апаратного забезпечення чи операційної системи, якщо він має інтерпретатор Java, через це вона була вибрана для ОС Андроїд. Окрім портативності, ще однією з ключових переваг Java є її набір функцій безпеки, які захищають ПК, на якому запущено програму Java, не лише від проблем, викликаних помилковим кодом, але й від шкідливих програм (таких як віруси). Ви можете

безпечно запускати аплет Java, завантажений з Інтернету, оскільки функції безпеки Java запобігають доступу цих типів аплетів до жорсткого диска комп'ютера або мережевих з'єднань. Аплет – це, як правило, невелика програма Java, вбудована в сторінку HTML.

Java є мовою програмування як скомпільованою, так і інтерпретованою мовою, так як її вихідний код спочатку компілюється в двійковий байт-код. Цей байт-код працює на віртуальній машині Java (JVM), яка зазвичай є програмним інтерпретатором. Використання скомпільованого байт-коду дозволяє інтерпретатору (віртуальній машині) бути малим і ефективним (і майже таким же швидким, як центральний процесор, що виконує власний скомпільований код). Крім того, цей байт-код надає Java її переносимість: вона працюватиме на будь-якій JVM, яка правильно реалізована, незалежно від комп'ютерної апаратної чи програмної конфігурації. Більшість веб-браузерів (таких як Microsoft Internet Explorer або Netscape Communicator) містять JVM для запуску Java-аплетів.

Порівняно з C++ (іншою об'єктно-орієнтованою мовою), код Java працює трохи повільніше (через JVM), але він більш портативний і має набагато кращі функції безпеки. Віртуальна машина забезпечує ізоляцію між ненадійною програмою Java і ПК, на якому запущено програмне забезпечення. Синтаксис Java подібний до C++, але мови досить різні. Наприклад, Java не дозволяє програмістам реалізувати перевантаження операторів, тоді як C++ дозволяє. Крім того, Java є динамічною мовою, де ви можете безпечно змінювати програму під час її роботи, тоді як C++ це не дозволяє. Це особливо важливо для мережевих програм, які не можуть дозволити собі будь-які простої. Крім того, усі базові типи даних Java є попередньо визначеними та не залежать від платформи, тоді як деякі типи даних можуть змінюватися залежно від платформи, що використовується в C або C++ (наприклад, тип int).

Програми Java структуровані краще, ніж еквіваленти C++. Усі функції (або методи Java) і виконувані оператори в Java повинні знаходитися в межах класу, тоді як C++ дозволяє визначенням функцій і рядкам коду існувати поза класами (як у програмах у стилі C). Глобальні дані та методи не можуть перебувати поза класом у

Java, тоді як C++ це дозволяє. Ці обмеження, хоч часом і обтяжливі, допомагають підтримувати цілісність і безпеку програм Java і змушують їх бути повністю об’єктно-орієнтованими.

2.3.2. Мова програмування Kotlin

Kotlin черпав натхнення у багатьох мовах програмування, включаючи (але не обмежуючись ними) Java, Scala, C# і Groovy. Однією з основних ідей Kotlin є прагматичність, тобто мова програмування, корисна для повсякденної розробки, яка допомагає користувачам виконувати роботу за допомогою своїх функцій та інструментів. Таким чином, на багато дизайнерських рішень впливало і залишається залежним від того, наскільки ці рішення корисні для користувачів Kotlin.

Kotlin — багатоплатформна, статично типізована мова програмування загального призначення. Наразі, починаючи з версії 1.7, він підтримує компіляцію для наступних платформ.

- JVM (Віртуальна машина Java)
- JS (JavaScript)
- Native (власні двійкові файли для різних архітектур)

Крім того, він підтримує прозору взаємодію між різними платформами за допомогою функції Kotlin Multiplatform Project (Kotlin MPP).

Система типів Kotlin під час компіляції розрізняє типи, що допускають нульові значення, і типи, що не допускають нульові значення, досягаючи нульової безпеки, тобто гарантуючи відсутність помилок виконання, спричинених відсутністю значення (тобто нульового значення). Kotlin також розширює свою статичну систему типів елементами поступової та потокової типізації для кращої

сумісності з іншими мовами та простоти розробки.

Рис. 2.4. Мова програмування Kotlin

Kotlin є об'єктно-орієнтованою мовою, яка також має багато функціональних елементів програмування. З об’єктно-орієнтованої сторони він підтримує номінальне підтипування з обмеженим параметричним поліморфізмом (схожим на дженерики) і дисперсію змішаного сайту. З боку функціонального програмування він має першокласну підтримку функцій вищого порядку та лямбда-літералів.

Ця специфікація охоплює Kotlin/Core, тобто фундаментальні частини Kotlin, які мають функціонувати майже однаково, незалежно від базової платформи. Ці частини включають такі важливі речі, як мовні вирази, оголошення, система типів і вирішення перевантажень.

2.4. Система збірки Android застосунків Gradle.

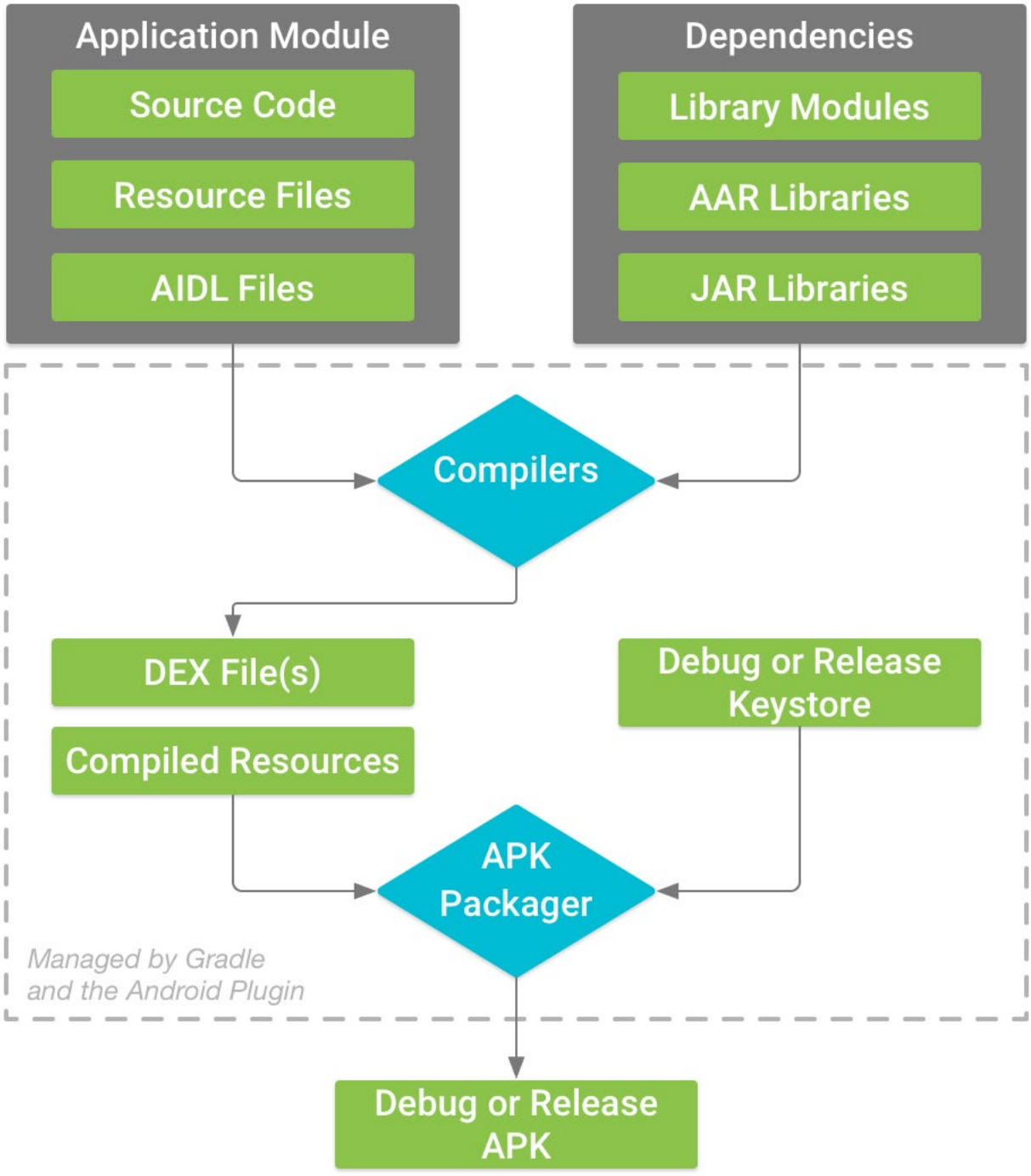
Система збірки Android компілює ресурси програми та вихідний код і пакує їх у файли APK або Android App Bundle, які можна тестувати, розгортати, підписувати та розповсюджувати. Android Studio використовує Gradle, вдосконалений набір інструментів збірки, для автоматизації та керування процесом збирання, дозволяючи нам визначати гнучкі спеціальні конфігурації збірки. Кожна конфігурація збірки може визначати власний набір коду та ресурсів, повторно використовуючи частини, спільні для всіх версій вашої програми. Плагін Android для Gradle працює з набором інструментів для створення процесів і конфігурованих параметрів, які є специфічними для створення та тестування програм Android.

Gradle і плагін Android працюють незалежно від Android Studio. Це означає, що ви можете створювати свої програми для Android з Android Studio, командного рядка на комп’ютері або на машинах, де Android Studio не встановлено (наприклад, сервери безперервної інтеграції). Якщо ви не використовуєте Android Studio, ви можете навчитися створювати та запускати свою програму з командного рядка.

Результат збирання однаковий незалежно від того, створюєте ви проект із командного рядка, на віддаленій машині чи за допомогою Android Studio.

Гнучкість системи збірки Android дає змогу виконувати власну конфігурацію збірки, не змінюючи основні вихідні файли програми.

Процес створення включає в себе багато інструментів і процесів, які перетворюють проект на пакет Android Application Package (APK) або Android App Bundle (AAB). Процес створення дуже гнучкий, тому корисно розуміти дещо з того,



що відбувається під капотом.

Рис. 2.6. Процес збирання типового модуля додатка Android, як показано на малюнку виконує такі загальні кроки:

- Компілятори перетворюють ваш вихідний код у файли DEX (Dalvik Executable), які містять байт-код, що працює на пристроях Android, а все інше — у скомпільовані ресурси.
- Пакетувальник поєднує файли DEX і скомпільовані ресурси в APK або AAB, залежно від вибраної цілі збірки. Перш ніж установити програму на пристрій Android або розповсюдити її в магазині, як-от Google Play, APK або AAB потрібно підписати.
- Пакетувальник підписує ваш APK або AAB за допомогою сховища ключів налагодження або випуску:

Якщо ви створюєте налагоджувальну версію свого додатка (debug), тобто додаток, призначений лише для тестування та профілювання, пакетувальник підписує ваш додаток за допомогою сховища ключів налагодження. Android Studio автоматично налаштовує нові проекти за допомогою сховища ключів налагодження.

Якщо ви створюєте випускну версію своєї програми, яку збираєтеся випустити за межі мережі, пакетувальник підписує вашу програму за допомогою сховища ключів випуску, яке потрібно налаштувати. Щоб створити сховище ключів випуску, прочитайте про підписання програми в Android Studio.

Перед створенням остаточного файлу .apk пакувальник використовує інструмент zipalign, щоб оптимізувати вашу програму, щоб використовувати менше пам’яті під час роботи на пристрої.

Наприкінці процесу складання у вас є файл APK або AAB для налагодження або випуску програми, який можна використовувати для розгортання, тестування чи випуску для зовнішніх користувачів.

2.5. Система контролю версій Git.

Git — це розподілена система керування версіями з відкритим кодом. Система контролю: це фактично означає, що Git є трекером вмісту. Отже, Git можна використовувати для зберігання вмісту — він переважно використовується для зберігання коду завдяки іншим функціям, які він надає.

Рис. 2.7. Емблема Git

Система контролю версій: код, який зберігається в Git, постійно змінюється, коли додається новий код. Крім того, багато розробників можуть додавати код паралельно. Тому система контролю версій допомагає впоратися з цим, зберігаючи історію змін, які відбулися. Крім того, Git надає такі функції, як розгалуження та злиття, про які я розповім пізніше.

Розподілена система контролю версій: Git має віддалене сховище, яке зберігається на сервері, і локальне сховище, яке зберігається на комп'ютері кожного розробника. Це означає, що код не просто зберігається на центральному сервері, а повна копія коду присутня на всіх комп'ютерах розробників. Git — це розподілена

система контролю версій, оскільки код присутній на комп'ютері кожного розробника. Далі в цій статті я поясню концепцію віддалених і локальних сховищ.

Навіщо потрібна система контролю версій, наприклад Git

У реальних проектах, як правило, кілька розробників працюють паралельно. Отже, потрібна система контролю версій, як-от Git, щоб гарантувати відсутність конфліктів коду між розробниками.

Крім того, вимоги в таких проектах часто змінюються. Таким чином, система контролю версій дозволяє розробникам повертатися до старішої версії коду.

Нарешті, іноді кілька проектів, які виконуються паралельно, включають ту саму кодову базу. У такому випадку концепція розгалуження в Git є дуже важливою.

ВИСНОВКИ ДО РОЗДІЛУ 2

Отже, ми розглянули засоби розробки програмного забезпечення, які спеціально заточені під додатки ОС Андроїд. Детально розглянули переваги кожного засобу, спеціальні можливості які він надає . Також освоїли мови програмування які застосовуються при розробці додатків.

РОЗДІЛ 3. ПРОЦЕСИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ЩОБ УНИКНУТИ ВИТОКУ ДАНИХ

В цьому розділі хотілося б розглянути аспекти при розробці програмного забезпечення, в нашому випадку – розробці мобільного додатку під ОС Андроїд, командою розробників. Саме вони мають доступ до відкритого коду, так як потрібно щось видозмінювати або реалізовувати новий функціонал в ньому, а отже можуть явно або неявно зашкодити безпеці проекту. Тому потрібно розглянути методи які можуть теоретично пошкодити код та способи уникнення цьому.

3.2 Розробка ПЗ з використанням VPN

Віртуальна приватна мережа, також відома як VPN, працює шляхом маршрутизації вашого пристрою через сервер третьої сторони, а не через вашого постачальника послуг Інтернету (ISP). Це означає, що коли ваші дані передаються через Інтернет, усі вони надходять із VPN, а не з вашого власного комп’ютера.

Говорячи простою мовою, це захищає ваш комп’ютер або мобільний пристрій від хакерів і шкідливих програм, а також захищає ваші дані та зв’язок. Якщо ви використовуєте його, ваші транзакції залишатимуться анонімними.

Завдяки цій додатковій безпеці розробники можуть переконатися, що вони захищають клієнтські дані та інтелектуальну власність. Вони також можуть досліджувати конкурентів і тестувати веб-сайти з інших країн. VPN допомагають запобігти витоку даних і підвищити загальну конфіденційність в Інтернеті.

Витоки даних — це прикро, але вони трапляються через погані практики кібербезпеки. Досвідчені хакери можуть отримати доступ до конфіденційних даних клієнта, що може завдати шкоди розробникам програм.

Рис. 3.1. Зображення VPN сервісу

VPN шифрує всі дані, які надходять із комп’ютера та пристрою розробника програми, що робить дані марними для хакерів. Цей вид шифрування схожий на те, що використовують банки для захисту клієнтів.

Більшість розробників додатків мають власний дизайн, додатки та іншу роботу, яка є цінною інтелектуальною власністю. Деякі люди можуть захотіти вкрасти цю інформацію.

Тим не менш, коли ви розробник, вам потрібно досліджувати конкурентів, щоб оптимізувати свій продукт. Якщо ви хочете зробити це непомітно, вам слід використовувати VPN. Це особливо стане в нагоді, коли вам потрібно протестувати веб-сайти або веб-додатки з інших країн.

3.2. Проведення Code Review процесу

Експертна перевірка коду може забезпечити послідовний стиль кодування в усьому проекті, завдяки чому вихідний код стане доступним для читання будь-ким, хто може познайомитися з проектом у будь-який момент під час розробки. Один проект може мати кілька фаз розробки, кожна з яких може бути розподілена між кількома розробниками. Якщо кожен розробник має власний характерний стиль кодування і дотримується його, проект неминуче стане роз'єднаним. Незалежно від того, наскільки добре кожен розробник відіграє свою роль, іншим буде складно ефективно включитися в процес розробки.

Використання процесу перевірки для забезпечення узгодженого дизайну та впровадження також сприяє ремонтопридатності та довговічності програмного забезпечення. Зміни неминуче потрібно буде внести, тому важливо подумати про те, хто відповідатиме за внесення цих змін. Чи будуть ті самі розробники на місці, коли знадобляться ці коригування коду? Чи буде проект розвиватися, і чи потрібно буде додавати нових розробників і прискорювати роботу? У процесі перевірки коду слід пам’ятати про ці фактори. Пізніше це заощадить вам і вашій команді дорогоцінний час, дозволяючи зосередитися на інноваційному програмному забезпеченні замість того, щоб з’ясовувати, як працює (чи не працює) код разом.

Обсяг будь-якого проекту програмного забезпечення та його вимоги можуть проходити через руки кількох розробників. Процес перевірки коду може служити перевіркою та балансом проти різних інтерпретацій цього обсягу та вимог порівняно з кодом, який у кінцевому підсумку буде доставлено. Другий набір очей може гарантувати, що ви не потрапите в «яму», яку ви створили на основі власного розуміння того, про що вас запитують, і що щось важливе не було пропущено. Ретельна перевірка коду колегами може заощадити багато часу на «протистоянні» перевірки якості пізніше.

Через брак досвіду деякі молоді розробники можуть не знати про методи оптимізації, які можна застосувати до їх коду. Процес перевірки коду дає можливість цим розробникам набути навичок і підвищити продуктивність свого

коду. Крім того, це дає цим молодим розробникам шанс відточити свої навички та стати експертами у своїй справі.

Реалізуючи необхідні функціональні можливості, багато розробників також намагаються оптимізувати код для лаконічності та ефективності продуктивності. Однак це призведе до більшої складності коду та меншої читабельності. Крім того, модель даних, на якій побудовано код, може змінитися на наступних етапах розробки. Таким чином, така передчасна оптимізація з часом призведе до збільшення вартості обслуговування.

Під час перегляду коду розробники також можуть ділитися один з одним новими технологіями та методами. Обмін техніками виходить за рамки стажу, коли кожен розробник однаково може ділитися, співпрацювати та вдосконалювати свої навички.

З іншого боку, ми шукаємо техніки та методи, щоб максимально автоматизувати процес рецензування та заощадити час кожного. Наприклад, щоб заощадити час на компіляції та розгортанні, ми представили інструмент Travis CI. Після ініціювання запиту на отримання github може синхронізуватися з Travis CI і повідомити, чи можна об’єднати збірку. Згодом Travis CI створить PR і відповідно оновить статус на github, щоб розробники могли своєчасно провести об’єднання.

Хоча ми працюємо над автоматизацією процесу перевірки коду, не все можна автоматизувати, тому ручне тестування іноді є більш ефективним і ретельним.
Це може здатися найбільш очевидною перевагою процесу рецензування коду, але це також одна з найважливіших. Коли ви працюєте в умовах реального тиску часу та бюджету, цей крок легко пропустити. Звісно, ви впевнені у своїй роботі, але навіть найкращі програмісти можуть дивитися на свою роботу схрещеними очима. Огляд коду допомагає по-новому поглянути на виявлення помилок і простих помилок кодування, перш ніж ваш продукт перейде до наступного кроку, що робить процес доставки програмного забезпечення клієнту ефективнішим.

3.3. Статична перевірка коду за допомогою Ktlint, Detekt

Як ми всі знаємо про мову kotlin, яку google рекомендує особливо для розробки додатків для Android, і, звичайно, це полегшило життя розробників додатків для Android. Але якщо ви новачок у цій галузі, ви можете не знати, що вам потрібно писати коди в потрібному форматі. і це безперечно корисно для розробників, але якщо ви не знайомі з тим, як писати чистий код за допомогою Kotlin, тоді не хвилюйтеся, вам не потрібно чогось вчитися, щоб досягти цього, натомість вам потрібно просто використовувати ktlint, який використовується для того ж мета.

Переваги його використання:

- Ktlitn може заощадити ваш час
- Це може заощадити вашу енергію (оскільки вам не потрібно вручну перевіряти стиль коду)
- Це може спростити ваш процес
- Що ми можемо зробити за допомогою ktlint?

Перед цим давайте поговоримо про деякі важливі речі, як-от ktlint розбивається на дві речі.

1) Інструмент лінінгу: інструмент лінінгу базується на стандартному посібнику зі стилю kotlin, і він перевірить і переконається, що ваш код відповідає цьому посібнику зі стилю.

2) Форматувальник: якщо ktlint виявить, що у вашому коді є проблеми, ви можете запустити форматувальник і попросити ktlint автоматично виправити ці проблеми за вас.

Щоб додати ktlint до свого проекту, а точніше, інтегрувати ktlint із плагіном ktlint Gradle,потрібно добавити його в наш gradle файл, це окремий плагін третьої сторони на додаток до основного інструменту ktlint, який робить роботу з ktlint дуже легкою, надаючи готовий Gradle завдання, з якими запускати команди ktlint. Є

кілька варіантів, але найпростіший спосіб — скористатися плагіном ktlint-Gradle, який надає готові завдання Gradle для інструментів ktlint. Для додавання плагіна ktlint-gradle просто додамо плагін ktlint-gradle до файлу build.gradle кореневого рівня [37].

Щоб застосувати плагін до підпроектів, просто застосуйте плагін ktlint-gradle до будь-яких модулів Gradle, які ми хочемо перевірити.

Так як ми використовуємо версію Gradle, яка підтримує плагіни DSL, ми можемо додати ktlint-gradle до свого проекту за допомогою такого коду:

```
// root-level build.gradle
plugins {
    id "org.jlleitschuh.gradle.ktlint" version "7.1.0"
}

// root-level build.gradle
buildscript {
  repositories {
    maven {
      url "https://plugins.gradle.org/m2/"
    }
  }
  dependencies {
    classpath "org.jlleitschuh.gradle:ktlint-gradle:7.1.0"
  }
}
```

Рис. 3.2. Зображення Ktlint залежностей

Для того щоб застосувати плагін ktlint-gradle до різних модулів у вашому проекті. Це можна зробити за допомогою блоку allProjects{} у файлі build.gradle кореневого рівня.

Detekt аналізуватиме код Kotlin за кількома наборами правил і позначатиме код, який порушує будь-яке з його правил. Detekt покращує кодову базу, застосовуючи набори правил, включаючи складність, іменування, продуктивність і потенційні помилки.

Доданий до CI, він також діятиме як мережа безпеки, якщо ви спробуєте об’єднати код, який містить поганий стиль коду Kotlin.

Щоб реалізувати Detekt, почнемо зі створення файлу detekt.gradle у корені проекту.

```
apply plugin: 'io.gitlab.arturbosch.detekt'

detekt {
    config = files("$rootDir/default-detekt-config.yml")
    filters = ".*build.*,.*/resources/.*,.*/tmp/.*"
    //Optional baseline, uncomment & run gradle command detektBaseline to exclude existing issues
    //baseline = file("detekt-baseline.xml")
}
```

Рис. 3.3. Добавлення Detect

Потім ми додаємо його до build.gradle верхнього рівня в розділі allprojects, щоб він застосовувався до всіх наших модулів, а також додаємо плагін Detekt у розділ плагінів.

```
buildscript {
    repositories {
        jcenter()
    }
}

plugins {
    id "io.gitlab.arturbosch.detekt" version "1.0.0-RC10"
}

allprojects {
    apply from: "$rootDir/detekt.gradle"
}
```

Рис. 3.4. Добавлення Detect до всіх модулів

detekt проаналізує код Kotlin, щоб переконатися, що правила не порушуються, інакше він завершиться помилкою.

3.4. Написання тестів, як запорука безпеки

Модульне тестування є потужним інструментом для підвищення якості програмного забезпечення – і воно було десятиліттями. Модильні тести забезпечують фундаментальну перевірку того, що програма відповідає специфікаціям дизайну програмного забезпечення та працює належним чином [38].

Якщо виконано добре, модульні тести:

- зменшувати дефекти та виявляти їх на ранніх стадіях життєвого циклу розробки;
- підвищення читабельності коду;
- дозволити повторне використання коду; і
- покращити швидкість розгортання.
- підвищення безпеки коду

```
import ...

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals( expected: 4,  actual: 2 + 2)
    }
}
```

Рис. 3.5. Зображення модульних тестів

Модульні тести, тип функціонального тесту, досягли більшості; це просто кількість команд розробників, які займаються бізнесом. Тим не менш, компанії, зазвичай використовують модульні тести спорадично та з низьким охопленням. Хоча середовище розробки підтримує модульні тести, і більшість програмістів знають базові фреймворки та інструменти, вони використовують модульні тести лише для простих і малоцінних екземплярів.

3.5. Автоматизація процесів за допомогою CI/CD

CI, скорочення від Continuous Integration, — це практика розробки програмного забезпечення, за якої всі розробники об’єднують зміни коду в центральному сховищі кілька разів на день. CD означає Continuous Delivery, що на додаток до Continuous Integration додає практику автоматизації всього процесу випуску програмного забезпечення.

За допомогою CI кожна зміна коду запускає автоматичну послідовність збірки та тестування для даного проекту, надаючи зворотний зв’язок розробникам, які внесли зміни. Увесь цикл зворотного зв’язку CI має пройти менше ніж за 10 хвилин.

Безперервна доставка включає надання інфраструктури та розгортання, яке може відбуватися вручну та складатися з кількох етапів. Важливо те, що всі ці процеси повністю автоматизовані, кожен запуск повністю реєструється та видимий для всієї команди.

Конвеєр CI/CD може здатися накладним, але це не так. По суті, це робоча специфікація кроків, які повинен виконати будь-який розробник, щоб доставити нову версію програмного продукту. За відсутності автоматизованого конвеєра інженерам все одно потрібно було б виконувати ці кроки вручну, а отже, це було б набагато менш продуктивно.

Більшість випусків програмного забезпечення проходять кілька типових етапів:

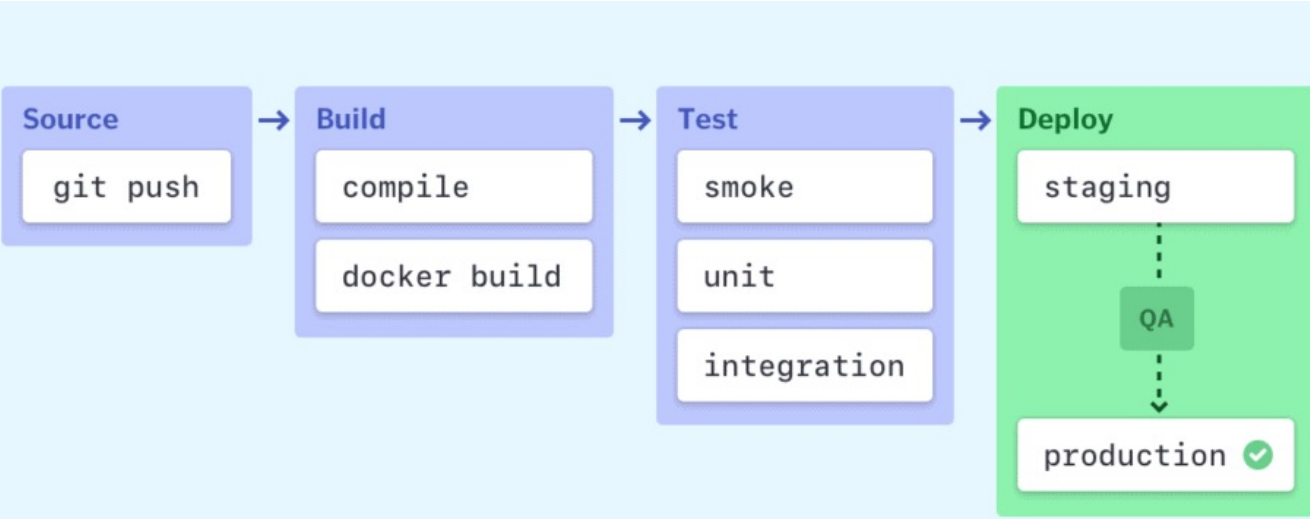


Рис. 3.6. Пайплайн на CI

Збій на кожному етапі зазвичай викликає сповіщення (через електронну пошту, Slack тощо), щоб повідомити відповідальних розробників про причину. В іншому випадку вся команда отримує сповіщення після кожного успішного розгортання у виробництві.

У більшості випадків запуск конвеєра ініціюється репозиторієм вихідного коду. Зміна коду запускає сповіщення інструменту CI/CD, який запускає відповідний конвеєр. Інші поширені тригери включають автоматично заплановані або ініційовані користувачем робочі процеси, а також результати інших конвеєрів.

Ми поєднуємо вихідний код і його залежності, щоб створити робочий екземпляр нашого продукту, який потенційно можемо надіслати нашим кінцевим користувачам. Програми, написані такими мовами, як Java, C/C++ або Go, потребують компіляції, тоді як програми на Ruby, Python і JavaScript працюють без цього етапу.

Незалежно від мови, хмарне програмне забезпечення зазвичай розгортається з Docker, і в цьому випадку на цьому етапі конвеєра CI/CD будуються контейнери Docker.

Непрохідність етапу збірки є індикатором фундаментальної проблеми в конфігурації проекту, і найкраще вирішити її негайно.

На цьому етапі ми запускаємо автоматизовані тести, щоб перевірити правильність нашого коду та поведінку нашого продукту. Тестовий етап діє як запобіжна сітка, яка запобігає потраплянню помилок, які легко відтворюються, до кінцевих користувачів.

Відповідальність за написання тестів лягає на розробників. Найкращий спосіб писати автоматизовані тести — робити це під час написання нового коду в розробці, керованій тестуванням або поведінкою.

Залежно від розміру та складності проекту цей етап може тривати від секунд до годин. Багато великомасштабних проектів проводять тестування в кілька етапів, починаючи з димових тестів, які виконують швидку перевірку працездатності, і закінчуючи наскрізними інтеграційними тестами, які перевіряють всю систему з точки зору користувача. Широкий набір тестів зазвичай розпаралелюється, щоб скоротити час виконання.

Помилка на етапі тестування виявляє проблеми в коді, які розробники не передбачили під час написання коду. На цьому етапі важливо швидко надати зворотний зв’язок розробникам, поки проблема ще свіжа в їхній пам’яті, і вони можуть підтримувати стан потоку.

Коли ми створимо придатний для виконання екземпляр нашого коду, який пройшов усі попередньо визначені тести, ми готові до його розгортання. Зазвичай існує кілька середовищ розгортання, наприклад, «бета-версія» або «поетапне» середовище, яке використовується внутрішньо командою продукту, і «виробниче» середовище для кінцевих користувачів.

Команди, які прийняли гнучку модель розробки, керуючись тестами та моніторингом у реальному часі, зазвичай розгортають незавершену роботу вручну в проміжне середовище для додаткового ручного тестування та перегляду, а також автоматично розгортають затверджені зміни з головної гілки до робочої.

3.6. Зберігання ключів доступу в безпечних сховищах

Зберігання ключів API або будь-якої іншої конфіденційної інформації в репозиторії git — це те, чого слід уникати будь-якою ціною. Навіть якщо сховище приватне, ви не повинні сприймати його як безпечне місце для зберігання конфіденційної інформації.

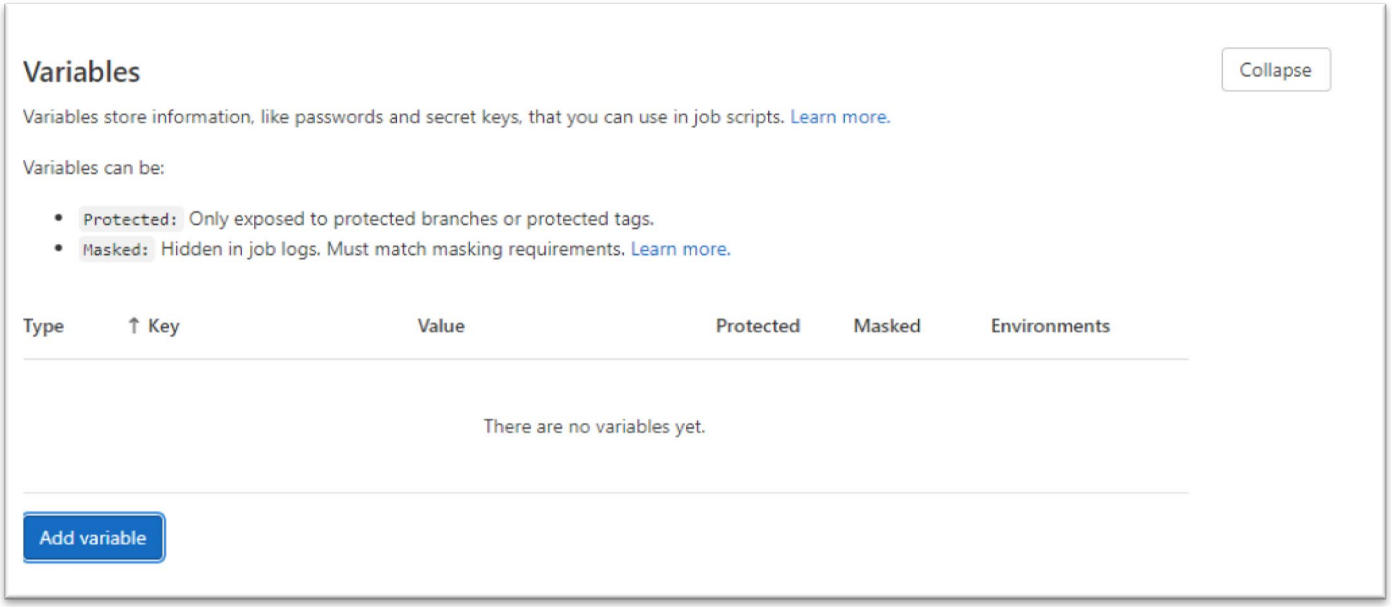


Рис. 3.6. Збереження ключів в GitHub

Давайте почнемо з того, що розглянемо, чому це погана ідея зберігати ключі API у загальнодоступних репозиторіях git.

За своєю природою публічний репозиторій git може отримати доступ будь-хто. Іншими словами, будь-хто, хто має підключення до Інтернету, може отримати доступ до вмісту публічного репозиторію git. Мало того, вони також можуть переглядати весь код у сховищі та, можливо, навіть запускати його. Якщо ви зберігаєте ключ API у загальнодоступному сховищі, ви публікуєте його у відкритому доступі, щоб його міг побачити кожен.

Нещодавній пошук client_secret на GitHub виявив, що існує понад 30 000 комітів, які потенційно розкривають ключ і секрет API. У деяких випадках для негайного доступу до API потрібно лише скопіювати та вставити код.

Ця проблема стає настільки важливою, що деякі компанії інвестують у ресурси, щоб переконатися, що немає витоку ключів і секретів API.

Минулого року Slack почав шукати відкриті маркери API та завчасно анулювати їх. Ця дія запобігає зловмисному доступу до облікових записів Slack, але не може знайти всі виточені токени.

Отже, це відбувається в публічних сховищах Git. А як щодо приватних? Чому це проблема?

Приватні репозиторії Git, розміщені в таких службах, як GitHub, GitLab і Bitbucket, наражаються на інший тип ризику. Коли ви інтегруєте програму третьої сторони з однією зі згаданих служб, ви можете відкривати свої приватні сховища для цих третіх сторін. Ці програми матимуть доступ до ваших приватних сховищ і читатимуть інформацію, що міститься в них.

Хоча це само по собі не створює ризику, уявіть, якщо одна з цих програм стане вразливою для зловмисників. Отримавши несанкціонований доступ до одного з цих сторонніх додатків, зловмисники можуть отримати доступ до ваших конфіденційних даних, включаючи ключі та секрети API.

Є багато альтернатив для безпечного зберігання ключів і секретів API. Деякі з них дозволяють використовувати ваш репозиторій Git і шифрувати конфіденційні дані. Інші інструменти є більш складними та розшифровують конфіденційну інформацію як частину робочого процесу розгортання. Давайте розглянемо деякі з доступних рішень.

git-remote-gcrypt

Перше рішення дозволяє зашифрувати ціле сховище Git. git-remote-gcrypt робить це шляхом додавання функціональності до віддалених помічників Git, щоб став доступним новий зашифрований транспортний рівень. Користувачам потрібно лише налаштувати новий зашифрований пульт і ввести в нього код.

git-секрет

git-secret — це інструмент, який працює на вашій локальній машині та шифрує певні файли перед тим, як ви надішлете їх у свій репозиторій. За лаштунками git-secret — це сценарій оболонки, який використовує GNU Privacy Guard (GPG) для

шифрування та дешифрування файлів, які можуть містити конфіденційну інформацію.

git-crypt

Іншим рішенням є git-crypt. Він дуже схожий на git-secret за принципом роботи, але має деякі цікаві відмінності.

Перше, на що слід звернути увагу щодо git-crypt, це те, що це двійковий виконуваний файл, а не сценарій оболонки, як git-secret. Будучи двійковим виконуваним файлом, це означає, що для його використання вам спочатку потрібно скомпілювати його, або вам потрібно знайти двійковий дистрибутив для вашої машини.

3.7. Типи бекдорів, способи уникнення

Кожна комп'ютерна система має офіційні засоби, за допомогою яких користувачі мають отримати до неї доступ. Часто це включає систему автентифікації, коли користувач надає пароль або інший тип облікових даних, щоб продемонструвати свою особу. Якщо користувач успішно пройшов автентифікацію, йому надається доступ до системи з дозволами, обмеженими тими, що призначені для його конкретного облікового запису.

Хоча ця система автентифікації забезпечує безпеку, вона також може бути незручною для деяких користувачів, як законних, так і нелегітимних. Системному адміністратору може знадобитися отримати віддалений доступ до системи, яка не призначена для цього. Зловмисник може захотіти отримати доступ до сервера бази даних компанії, незважаючи на відсутність для цього облікових даних. Виробник системи може включити обліковий запис за умовчанням, щоб спростити налаштування, тестування та розгортання оновлень системи.

У цих випадках у систему може бути вставлений бекдор. Наприклад, системний адміністратор може налаштувати веб-оболонку на сервері. Коли вони хочуть отримати доступ до сервера, вони відвідують відповідний сайт і можуть надсилати команди безпосередньо на сервер без необхідності автентифікації або

налаштування корпоративної політики безпеки для прийняття безпечного протоколу віддаленого доступу, наприклад SSH.

Бекдор забезпечує доступ до системи в обхід звичайних механізмів автентифікації організації. Кіберзлочинці, які теоретично не мають доступу до законних облікових записів у системах організації, можуть використовувати його для віддаленого доступу до корпоративних систем. Завдяки такому віддаленому доступу вони можуть викрадати конфіденційні дані, розгортати програму-вимагач, шпигунське програмне забезпечення чи інше зловмисне програмне забезпечення та виконувати інші шкідливі дії в системі.

Часто бекдори використовуються, щоб надати зловмиснику початковий доступ до середовища організації. Якщо системний адміністратор або інший законний користувач створив бекдор у системі, зловмисник, який виявить цей бекдор, може використати його у власних цілях. Крім того, якщо зловмисник виявляє вразливість, яка дозволить йому розгорнути власний бекдор у системі, він може використати бекдор, щоб розширити свій доступ і можливості в системі.

Бекдори можуть мати різні форми. Деякі з найпоширеніших типів включають:

Трояни: більшість бекдор-зловмисного програмного забезпечення розроблено для того, щоб пройти повз захист організації, забезпечуючи зловмиснику опору на системи компанії. З цієї причини вони зазвичай є троянськими програмами, які видають себе за безпечний або бажаний файл, але містять шкідливі функції, такі як підтримка віддаленого доступу до зараженого комп’ютера.

Вбудовані бекдори: виробники пристроїв можуть включати бекдори у вигляді облікових записів за замовчуванням, незадокументованих систем віддаленого доступу та подібних функцій. Хоча ці системи, як правило, призначені лише для використання виробником, вони часто розроблені таким чином, що їх неможливо вимкнути, і жоден бекдор не залишається секретом назавжди, відкриваючи ці діри безпеки для зловмисників.

Веб-оболонки: веб-оболонка — це веб-сторінка, призначена для прийому введених даних користувачами та виконання їх у системному терміналі. Ці бекдори

зазвичай встановлюють системні та мережеві адміністратори, щоб полегшити віддалений доступ до корпоративних систем і керування ними.

Експлойти ланцюга поставок: веб-програми та інше програмне забезпечення часто включають сторонні бібліотеки та код. Зловмисник може включити бекдор-код у бібліотеку в надії, що він буде використаний у корпоративних програмах, надаючи бекдор-доступ до систем, на яких працює програмне забезпечення.

Деякі найкращі методи захисту від використання бекдорів включають:

- Зміна облікових даних за замовчуванням: облікові записи за замовчуванням є одними з найпоширеніших типів бекдорів. Під час налаштування нового пристрою вимкніть облікові записи за замовчуванням, якщо це можливо, а якщо ні, змініть пароль на інший, ніж стандартний.
- Розгортання рішень безпеки кінцевих точок: бекдори зазвичай реалізуються як троянське програмне забезпечення. Рішення безпеки кінцевої точки може виявляти та блокувати відомі шкідливі програми або ідентифікувати нові загрози на основі незвичайної поведінки.
- Моніторинг мережевого трафіку: бекдори призначені для надання віддаленого доступу до систем за допомогою альтернативних засобів, які обходять системи автентифікації. Моніторинг незвичайного мережевого трафіку може дозволити виявити ці приховані канали.
- Сканування веб-додатків: бекдори можуть бути розгорнуті як веб-оболонки або інтегровані в сторонні бібліотеки чи плагіни. Регулярне сканування вразливостей може допомогти виявити ці бекдори у веб-інфраструктурі організації.

ВИСНОВОК ДО РОЗДІЛУ 3

Отже, при розробці програмного продукту в великій компанії, де залучена велика кількість розробників, потрібно звертати уваги не лише на програмну складову вразливостей коду, але й на людський фактор.

В даному випадку ми розглянули як уникнути ненавмисному пошкодженню коду за допомогою процесу код рев'ю, при якому вся команда розробників передивляється написаний код іншого розробника і намагається найти якусь несумісність якщо така є.

Також ми розглянули статичні аналізатори коду, для мови програмування Kotlin, так як вона зараз є основною для розробки під ОС Андроїд, які допомагають нам виявити баги та прогалини в безпеці під час збірки проекту, тобто на ранньому етапі розробки, що до тестування коду тестувальниками чи перевірки його на вразливість.

РОЗДІЛ 4. МЕТОДИ ПРОТИДІЇ ВИКРАДЕННЮ ПЕРСОНАЛЬНИХ ДАНИХ ПРИ РОЗРОБЦІ ПЗ НА ОС АНДРОЇД

Безліч витоків персональної інформації відбуваються через недосконаліть програмного забезпечення написаного програмістами які не приділяють безпеці мобільним додаткам належної уваги. Не можна недооцінювати хакерів, які можуть знайти спосіб щоб заволодіти банківськими карточками користувача і викрасти в нього гроші. В такому випадку, це вкрай негативно вплине на рейтинг додатку в магазинах додатків під платформу, також безперечно на імідж бізнесу. Як результат бізнес може залишитися без клієнтів і збанкротувати.

Так як більшіть взломів телефонів відбуваються через недосконаліть самої операційної системи, на що розробники саме мобільних додатків не зможуть повпливати, а також через необачність і необізнаність користувача (наприклад фішингова атака). Але все ж є способи захистити персональні дані користувача від викрадення, які ми розглянемо і реалізуємо в цьому розділі.

4.1. Безпечне з’єднання з сервером з використанням сертифікатів SSL та TSL

Сертифікати SSL, також відомі як сертифікат рівня захищених сокетів, які використовуються для захисту з’єднань між браузером і сервером. Простіше кажучи, коли ви здійснюєте онлайн-транзакцію, цей SSL забезпечує безпечний тунель або з’єднання. Щоб ваша конфіденційна інформація була захищена від атак хакерів/зловмисників.

Залежно від використання, існують типи сертифікатів SSL на вибір. Але більшість людей не знають, як працює сертифікат SSL. Щоб зрозуміти роботу сертифіката SSL. Шифрування та автентифікація є основною частиною роботи

сертифіката SSL. Ось основні принципи, які необхідно зрозуміти, щоб зрозуміти, як працює SSL/TLS:

Захищене спілкування починається з рукостискання TLS, під час якого дві сторони, що спілкуються, відкривають безпечне з'єднання та обмінюються відкритим ключем. Під час рукостискання TLS дві сторони генерують ключі сеансу, а ключі сеансу шифрують і розшифровують усі повідомлення після рукостискання TLS

Різні ключі сеансу використовуються для шифрування зв'язку в кожному новому сеансі TLS гарантує, що сторона на стороні сервера або веб-сайт, з яким взаємодіє користувач, насправді є тим, за кого себе видає. TLS також гарантує, що дані не були змінені, оскільки код автентифікації повідомлення (MAC) додається до передачі

За допомогою TLS шифруються дані HTTP, які користувачі надсилають на веб-сайт (клацаючи, заповнюючи форми тощо), і дані HTTP, які веб-сайти надсилають користувачам. Зашифровані дані мають бути розшифровані одержувачем за допомогою ключа.

Сеанс зв'язку TLS починається з рукостискання TLS. У рукостисканні TLS використовується так зване асиметричне шифрування, тобто два різні ключі використовуються на двох кінцях розмови. Це можливо завдяки техніці, яка називається криптографією з відкритим ключем.

У криптографії з відкритим ключем використовуються два ключі: відкритий ключ, який сервер робить загальнодоступним, і закритий ключ, який зберігається в таємниці та використовується лише на стороні сервера. Дані, зашифровані відкритим ключем, можна розшифрувати лише закритим ключем.

Під час рукостискання TLS клієнт і сервер використовують відкритий і закритий ключі для обміну випадково згенерованими даними, і ці випадкові дані використовуються для створення нових ключів для шифрування, які називаються ключами сеансу.

На відміну від асиметричного шифрування, у симетричному шифруванні дві сторони в розмові використовують той самий ключ. Після рукостискання TLS

обидві сторони використовують однакові ключі сеансу для шифрування. Коли ключі сеансу використовуються, відкритий і закритий ключі більше не використовуються. Ключі сеансу – це тимчасові ключі, які не використовуються знову після завершення сеансу. Для наступного сеансу буде створено новий випадковий набір ключів сеансу.

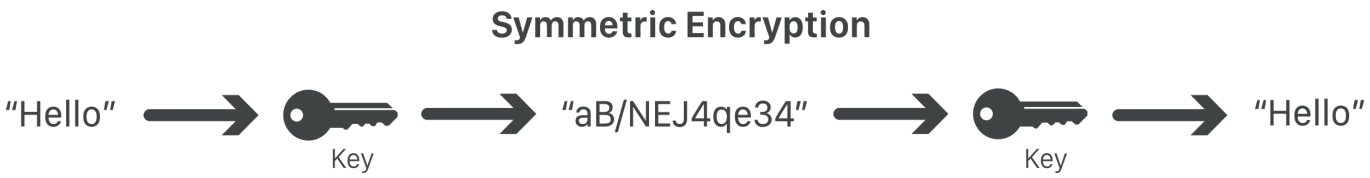


Рис. 4.1. Симетричне шифрування

Зв’язок TLS із сервера включає код автентифікації повідомлення, або MAC, який є цифровим підписом, який підтверджує, що повідомлення походить із фактичного веб-сайту. Це автентифікує сервер, запобігаючи атакам на шляху та спуфінгу домену. Це також гарантує, що дані не були змінені під час передачі.

Сертифікат SSL – це файл, встановлений на початковому сервері веб-сайту. Це просто файл даних, що містить відкритий ключ та особу власника веб-сайту, а також іншу інформацію. Без сертифіката SSL трафік веб-сайту не можна зашифрувати за допомогою TLS.

Технічно будь-який власник веб-сайту може створити власний сертифікат SSL, і такі сертифікати називаються самопідписаними сертифікатами. Однак браузери не вважають самопідписані сертифікати такими ж надійними, як сертифікати SSL, видані центром сертифікації.

Власникам веб-сайтів потрібно отримати сертифікат SSL від центру сертифікації, а потім установити його на своєму веб-сервері (часто веб-хостинг може впоратися з цим процесом). Центр сертифікації — це зовнішня сторона, яка може підтвердити, що власник веб-сайту — це той, за кого себе видає. Вони зберігають копії сертифікатів, які видають.

Багато центрів сертифікації стягують плату за сертифікати SSL. Щоб зробити Інтернет більш безпечним, Cloudflare пропонує безкоштовні сертифікати SSL. Cloudflare була першою компанією з безпеки та продуктивності в Інтернеті, яка це зробила. Cloudflare також працював над оптимізацією продуктивності SSL/TLS, щоб веб-сайти, які переходять з HTTP на HTTPS, не вплинули на продуктивність. Щоб отримати додаткові відомості про параметри SSL із Cloudflare, перегляньте нашу документацію для розробників.

S у "HTTPS" означає "безпечний". HTTPS — це просто HTTP з SSL/TLS. Веб-сайт із адресою HTTPS має законний сертифікат SSL, виданий центром сертифікації, і трафік до та з цього веб-сайту автентифікується та шифрується за допомогою протоколу SSL/TLS.

Щоб спонукати весь Інтернет перейти на безпечніший HTTPS, багато веб-браузерів почали позначати веб-сайти HTTP як «небезпечні» або «небезпечні». Таким чином, HTTPS не тільки необхідний для захисту користувачів і даних користувачів, він також став важливим для побудови довіри між користувачами. Перевірте веб-сайт на наявність проблем із SSL/HTTPS.

Перевірка домену SSL (DV): Вважається, що протокол DV SSL має низький рівень надійності, і його найлегше отримати — все, що вам потрібно підтвердити, це те, що ви є власником домену, на який ви подаєте заявку на SSL. Цей метод перевірки достатньо безпечний, однак якщо хтось отримує контроль над доменом, яким він не володіє (тобто доступ до електронної пошти, доступ до ftp або доступ до DNS), SSL може бути видано не тій особі. Чудово підходить для особистого користувача, програми Facebook або сторінки входу.

Розширена перевірка SSL (EV): EV SSL є найбільш захищеними від хакерів сертифікатами в галузі, тому вважаються найбільш надійними. Щоб отримати EV, необхідно підтвердити юридичне існування компанії, історію діяльності та фізичне існування. Усі ці підтверджені дані відображаються в деталях сертифіката в адресному рядку браузерів, щоб вказати назву вашої компанії та легітимність існування. Такі сертифікати неможливо підробити, тому вони отримують оцінку «дуже висока гарантія».

Перевірка організації SSL (OV): OV SSL також вимагає підтвердження юридичного існування організації, однак процес перевірки організації вимагає менше кроків, тобто не має значення, як довго працює компанія, центр сертифікації не перевіряє особу, яка подає заявку на OV від імені компанії, ані юридичну адресу перевірено через публічні джерела. Реквізити перевіреної компанії вказані в деталях сертифіката.

4.2. Використання лише безпечного з'єднання в мобільному додатку Андроїд

По замовчуванню, при використанні http з'єднання без SSL сертифікату – буде виникати помилка при запуску мобільного додатку, так як це продумали при розробці п'ятої версії операційної системи Андроїд. Але бувають випадки коли нам все ж потрібно відправити запит на сервер без SSL сертифікату (як приклад при старті проекту, коли розробка почалася, але ще не встановили сертифікат). Для цього нам потрібно добавити файл в якому ми впишемо налаштування для успішної роботи додатку при цих умовах.

Функція конфігурації мережевої безпеки дає змогу налаштувати параметри мережевої безпеки програми в безпечному декларативному файлі конфігурації, не змінюючи код програми. Ці параметри можна налаштувати для певних доменів і для певної програми. Основні можливості цієї функції:

- Спеціальні прив'язки довіри: налаштуйте, яким центрам сертифікації (CA) можна довіряти для безпечних з'єднань програми. Наприклад, довіряти певним самопідписаним сертифікатам або обмежувати набір загальнодоступних ЦС, яким програма довіряє.
- Перевизначення лише для налагодження: безпечно налагоджуйте безпечні з'єднання в програмі без додаткового ризику для встановленої бази.
- Відмова від трафіку відкритого тексту: захистіть програми від випадкового використання трафіку відкритого тексту (незашифрованого).

- Закріплення сертифіката: обмежте безпечне з’єднання програми певними сертифікатами.

Спеціальні прив’язки довіри: налаштування, якими центр сертифікації (ЦС) можна довіряти для безпечних з’єднаних програм. Наприклад, довіряти певним самопідписаним сертифікатам або обмежувати набір загальнодоступних ЦС, якою програма довіряє.

Перевизначення лише для накладання: безпечно налагоджуйте безпечні з’єднання в програмі без додаткового ризику для встановленої бази.

Відмова від трафіку відкритого тексту: захистити програму від випадкового використання трафіку відкритого тексту (незашифрованого).

Закріплення сертифіката: обмежте безпечне з’єднання програми певними сертифікатами.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Рис. 4.2. Налаштування безпеки в Андроїд

Ви можете захотіти, щоб ваша програма довіряла спеціальному набору ЦС замість стандартної платформи. Найпоширеніші причини цього:

- Підключення до хосту за допомогою спеціального ЦС, наприклад, самопідписаного ЦС або виданого всередині компанії.
- Обмеження набору ЦС лише тими ЦС, яким ви довіряєте, замість кожного попередньо встановленого ЦС.
- Довіра додаткових ЦС, не включених до системи.

За замовчуванням безпечні з’єднання (з використанням таких протоколів, як TLS і HTTPS) від усіх додатків довіряють попередньо встановленим системним центрам сертифікації, а програми, націлені на Android 6.0 (рівень API 23) і нижчі, також за умовчанням довіряють доданому користувачем ЦС. Ви можете налаштувати підключення програми за допомогою base-config (для налаштування всієї програми) або domain-config (для налаштування для кожного домену).

Конфігурація для обмеження набору довірених ЦС подібна до довіри спеціальному ЦС для певного домену, за винятком того, що в ресурсі надається кілька ЦС. Наведений нижче уривок коду демонструє, як обмежити набір довірених центрів сертифікації вашої програми в res/xml/network_security_config.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">secure.example.com</domain>
        <domain includeSubdomains="true">cdn.example.com</domain>
        <trust-anchors>
            <certificates src="@raw/trusted_roots"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Рис. 4.3. Шифрування домейнів

4.3. Постійне оновлення залежностей (бібліотек) проекту

В будь якому проекті, в незалежності чи це мобільний додаток чи серверний є певні бібліотеки які виконують певну рутинну роботу. Це нормальна практика їх використання. Для Андроїд проектів, існують певні хмарні сервери на яких зберігаються дані залежності, такі як Maven, JCentral або ж Google. Щоб використати ці бібліотеки в себе в проекті, потрібно лише прописати шлях до цієї бібліотеки в файлі build.gradle (модульної видимості).

```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.3.2'
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

}
```

Рис. 4.4. Залежності проекту

Здавалося б, що при звичайному використанні стороніх бібліотек, які є доволі поширені та підтримуються топовими компаніями такі як Google, IntelliJ та інші є також деякі небезпеки.

По перше, зловмисники можуть замінити бібліотеки які ми вказали вище, на свої під час оновлення залежностей проекту. Така атака має назву Man In The Middle, і щоб уникнути даної ситуації, потрібно використовувати VPN який провадить нас до нашого репозиторію з збереженими залежностями (рис 4.5)

```
repositories {
    mavenLocal()
    maven {
        url "${artifactory_url}/mobile-generic-virtual"
        credentials { PasswordCredentials it ->
            username = "${artifactory_user}"
            password = "${artifactory_password}"
        }
    }
}
```

Рис. 4.5. Добавлення VPN для Maven

де artifactory_url – це частина посилання на наш maven репозиторій, arifactory_user artifactory_password – логін та пароль розробника за допомогою якого він отримає доступ до цього репозиторія. Варто звернути увагу, що логін та пароль не прописуються в build.gradle проекту, так як вони можуть потрапити в ремоут репозиторій і ваші сенсетивні дані можуть побачити всі інші розробники. Для цього вони виносяться в локальний файл для налаштукань Gradle який буде лежати лише на вашому компютері.

По друге, оновлення бібліотек не тільки дає нам новий покращений функціонал, але й більш безпечний код. Без сумніву, бібліотеки також мають якісь баги, через які зловмисники можуть зломати наш застосунок, тому потрібно бути обережним з цим. Рішення для цієї проблеми є простим: завжди потрібно оновлювати версії бібліотек до найбільш актуальної, так як скоріш за все в ній вже знаходяться фікси для багів по безпеці коду, які було виявлено раніше.

4.4. Використання Timber бібліотеки

Розробляючи додатки для Android, ми, як розробники, маємо тенденцію вводити багато логів із різними пріоритетами, для цього Android SDK має службовий клас Log, у якому є методи для реєстрації наших повідомлень із різними пріоритетами, найпоширеніші методи, які ми використовуємо для різних випадків, наприклад, ми використовуємо Log.e(TAGGER, "message") у випадках, коли ми хочемо показати якусь помилку (це зазвичай відображається червоним кольором у вікні logcat android studio), або Log.d(TAG, "message"), коли ми хочемо надрукувати якесь повідомлення з метою налагодження, наприклад значення деяких даних.

Зазвичай, коли розробка програми завершена, і настав час випустити програму в Play Store, нам потрібно видалити всі оператори журналу з програми, щоб жодні дані програми, такі як інформація про користувача, приховані дані програми, токени

авторизації доступні для користувача в logcat як звичайний текст, і це стає громіздким завданням, а також є ризик що ми пропустимо видалення logcat і будь який користувач зможе прочитати наші залоговані дані. Для вирішення даної проблеми, ми можемо використовути Timber бібліотеку.

Щоб підлючити Timber, потрібно прописати її в build.gradle, а також добавити ініціалізуючий код в Application class Андроїд фреймворку

```
implementation 'com.jakewharton.timber:timber:4.7.1'
```

```
package com.death.timberdemo;

import android.app.Application;

import timber.log.Timber;

public class ApplicationController extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        if(BuildConfig.DEBUG){
            Timber.plant(new Timber.DebugTree());
        }
    }
}
```

Рис. 4.6. Добавлення Timber

В цьому коді ми добаваляємо DebugTree та підключаємо його лише в debug версії додатку. Тобто при розробці додатку в нас будуть логи, а при релізній версії, їх не буде і ми будемо в безпеці від ненавмисного витоку залогованих даних.

4.5. Використання WebView

Якщо ви хочете доставити веб-програму (або просто веб-сторінку) як частину клієнтської програми, ви можете зробити це за допомогою WebView. Клас WebView є розширенням класу View Android, який дозволяє відображати веб-сторінки як частину вашого макета діяльності. Він не містить жодних функцій повністю розробленого веб-браузера, таких як елементи керування навігацією чи адресний рядок. Все, що робить WebView за замовчуванням, це показує веб-сторінку.

Поширений сценарій, коли використання WebView є корисним, коли ви хочете надати інформацію у своїй програмі, яку вам може знадобитися оновити, наприклад угоду з кінцевим користувачем або посібник користувача. У програмі для Android ви можете створити дію, яка містить WebView, а потім використовувати її для відображення документа, розміщеного в Інтернеті.

Інший сценарій, у якому WebView може допомогти, полягає в тому, що ваша програма надає дані користувачеві, якому завжди потрібне підключення до Інтернету для отримання даних, наприклад електронної пошти. У цьому випадку ви можете виявити, що простіше створити WebView у вашій програмі для Android, який показуватиме веб-сторінку з усіма даними користувача, замість того, щоб виконувати мережевий запит, потім аналізувати дані та відтворювати їх у макеті Android. Натомість ви можете створити веб-сторінку, адаптовану для пристроїв Android, а потім реалізувати WebView у своїй програмі Android, яка завантажує веб-сторінку.

Якщо веб-сторінка, яку ви плануєте завантажити у свій WebView, використовує JavaScript, ви повинні ввімкнути JavaScript для свого WebView. Після ввімкнення JavaScript ви також можете створювати інтерфейси між кодом програми та кодом JavaScript.

JavaScript вимкнено у WebView за замовчуванням. Ви можете ввімкнути його через WebSettings, прикріплені до вашого WebView. Ви можете отримати

WebSettings за допомогою getSettings(), а потім увімкнути JavaScript за допомогою setJavaScriptEnabled().

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

Рис. 4.7. Добавлення WebView

Використання addJavascriptInterface() дозволяє JavaScript контролювати вашу програму Android. Це може бути дуже корисною функцією або небезпечною проблемою безпеки. Якщо HTML у WebView є ненадійним (наприклад, частина або весь HTML надано невідомою особою чи процесом), тоді зловмисник може включити HTML, який виконує ваш клієнтський код, і, можливо, будь-який код за вибором зловмисника. Таким чином, ви не повинні використовувати addJavascriptInterface(), якщо ви не написали весь HTML і JavaScript, який відображається у вашому WebView. Ви також не повинні дозволяти користувачеві переходити на інші веб-сторінки, які не є вашими, у вашому WebView. Натомість дозвольте браузеру користувача за замовчуванням відкривати іноземні посилання — за замовчуванням веб-браузер користувача відкриває всі URL-посилання, тому будьте обережні, лише якщо ви обробляєте навігацію сторінкою, як описано в наступному розділі).

4.6. Збереження логіну та паролю в мобільному додатку

Для того, щоб користувач зміг повторно зайти в наш додаток, без рутинного введення логіну та пароля, ми можемо зберегти дане значення в пам’яті телефону. Операційна система Андроїд має кілька варіантів збереження даних: такі як база

даних SQLite, обгортка над SQLite бібліотека Room, збереження даних у файловому форматі у внутрішній (захищеній) директорії, у зовнішній, на карті пам’яті та спосіб збереження у вигляді ключа – значення SharedPreferences . Найкраще та найбезпечніше для наших завдань підходить останній варіант. Хоча і зберігає дані в SharedPreferences дуже швидко та зручно, зловмисникам також легко переглядати дані, які зберігаються в SharedPreferences. Тож потрібно буди обережним з тим, що ми туди додаємо, і, можливо, почати думати про спосіб зберігання даних у зашифрованому форматі.

Одним із способів є використання EncryptedSharedPreferences, яке робить усі шифрування під капотом. EncryptedSharedPreferences є надійним і простим для Android 6.0 і новіших версій. Він має два великих плюси: це не вимагає від нас жорсткого коду будь-чого в нашому коді. Він просто використовує для нас Android Keystore, використовуючи біль від роботи з ним. Користувачеві не потрібно встановлювати блокування екрана. EncryptedSharedPreferences працює так само добре без блокування екрана. Це майже додаткова заміна SharedPreferences.

Іншим способом є шифрування даних вручну. Шифрування — це метод, за допомогою якого інформація перетворюється на секретний код, який одержує справжнє значення інформації. Наука про шифрування та дешифрування інформації називається криптографією. В обчислювальній техніці незашифровані дані також називають відкритим текстом, а зашифровані дані називають зашифрованим текстом. Формули, які використовують для кодування та декодування повідомлення, називаються алгоритмами шифрування або шифрами. Щоб бути ефективним, шифр включає в себе змінну як весь алгоритм. Змінна, яка називається ключем, робить вихід шифру унікальним. Якщо зашифроване повідомлення перехоплюється за допомогою неавторизованої організації, зловмисник повинен здогадатися, який зашифрував відправник для шифрування повідомлення, а також ключ, який використовувався як змінний. Час і труднощі введення цієї інформації зробити шифрування таким цінним інструментом безпеки. Шифрування є давнім способом захисту конфіденційної інформації. Історично він використовувався військовими та урядами. У наш час шифрування використовують для захисту даних, що

зберігаються на комп’ютерах і запам’ятованих пристроях, а також даних, що передаються по мережах.

Тож з написанням своїх функцій шифрування\дешифрування ми можемо легко зберегти конфіденційні дані в мобільному пристрої без страху, що вони можуть бути прочитані зловмисниками.

4.7. Створення надійного функціоналу для аутентифікації користувача

Одним із головних екранів у багатьох застосунках є екранна аутентифікація для подальшого використання додатку. Цей екран дозволяє перевірити чи справді та людина, яка має відношення до цього облікового запису, використовуючи цей додаток. зазвичай він має наступний вигляд (рис 4.8).

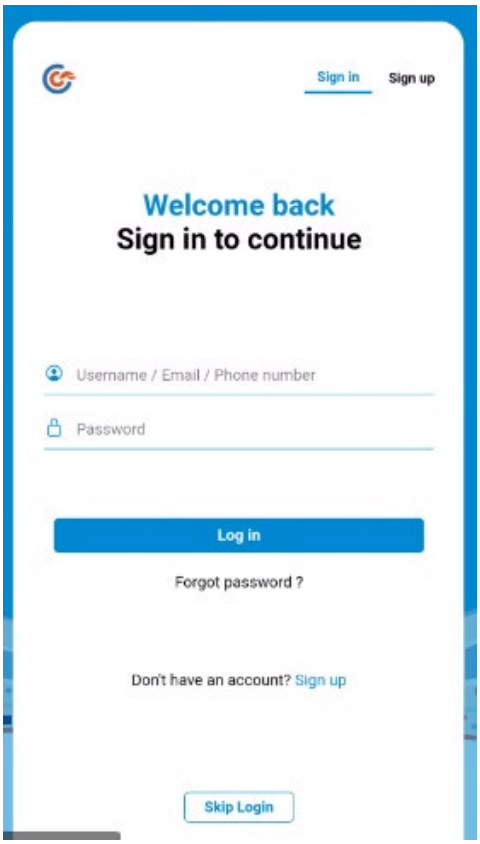


Рис. 4.8. Екран логіну

Безумовно найкращим і найнадійнішим способом для цього є просто введення звичайного текстового пароля в поле, але для покращення зручності використання додатка можна розмістити інші способи автентифікації в додатку. Для цього візьмемо бібліотеку, яку рекомендує компанія використовувати Google – BiometricManager. Біометрія пропонує більш зручний, але менш безпечний спосіб підтвердження вашої особи за допомогою пристрою. Згідно з багаторівневою моделлю автентифікації, первинна автентифікація (тобто модальності на основі фактора знань, такі як PIN-код, шаблон і пароль) забезпечує найвищий рівень безпеки. Біометрія знаходиться на вторинному рівні автентифікації, забезпечуючи баланс зручності та безпеки. Android CDD визначає три класи біометричної надійності: клас 3 (більш сильний), клас 2 (більш слабкий) і клас 1 (більш зручність). Кожен клас має набір передумов, привілеїв і обмежень. Усім трьом класам дозволено інтегруватись із Lockscreen, але лише сильні та слабкі автентифікатори дозволено інтегрувати з android.hardware.biometrics API. Платформа Android включає підтримку біометричної автентифікації обличчя та відбитків пальців. Android можна налаштувати для підтримки інших біометричних модальностей (наприклад, Iris). Однак біометрична інтеграція залежить від біометричної безпеки, а не від модальності. Якщо ваш пристрій підтримує кілька біометричних даних, користувач повинен мати можливість вказати параметри для замовчування в налаштуваннях. Ваша реалізація BiometricPrompt має віддавати перевагу біометрії класу 3 (раніше Strong) для замовчування, якщо явно не замінити її, тоді відображати попередження має з поясненнями ризиків, пов’язаних з біометром (наприклад, ваша фотографія може розблокувати ваш пристрій).

Звідси, задля найбільш безпечного використання додатком, ми як розробники повинні забезпечити найвищий рівень безпеки (тобто Class 3), це є звичайний пароль, який бажано робити складним, і біометрію так як відбиток пальця чи сканувати обличчя. Надійність біометрії лягає на ліцензійного вендора андроїд девайсів.

4.8. Захист інтелектуальної власності коду

На жаль, у багатьох програмах все ж таки є властивості, які зв’язані з поганим написанням коду. І цю вразливість хакери можуть знайти лише, якщо заглянути в початковий код програми. Для цього їм також не потрібно викрасити репозиторій проекту чи дезберігається код, достатньо лише провести кілька маніпуляцій із зворотним проектуванням і декомпілювати мобільний застосунок. Якщо ви один із розробників, які вважають, що жорстке кодування секретних ключів або навіть збереження їх у файлі build.gradle запобіжить його подальшою рукою хакерів чи інших розробників, ви помиляєтеся.

Безпека ніколи не була легкою, і перше правило — ніколи не довіряти безпеці на стороні клієнта. Клієнтська сторона не є середовищем, яке ми контролюємо, тому мені не потрібно розміщуватися на ньому, жорстко кодуючи або зберігаючи секрети, які можуть зруйнувати роботу наших систем. Тож найкращий спосіб гарантувати, що вас не спіймають розробники та хакери, — це самостійно перепроектувати програму та, якщо можливо, виправити проблеми.

Перший крок - Нам знадобиться APK програми, яку ми хочемо перепроектувати. Є багато способів зробити це, але розглянемо тут простий спосіб. Завантажуємо програму Apk Extractor на наш пристрій і вибираємо програму зі списку всередині програми. Після цього відкриємо будь-який Провідник файлів і перейдемо до папки ExtractedApks, яка знаходиться в каталозі внутрішнього сховища. Там ми знаходимо APK. Скопіюємо цей APK у нашій системі, і виконуємо наведені нижче дії.

Другий крок — Отримавши файл APK, скасуйте його, щоб дізнатися та переглянути код. Це підтверджує структуру коду та має виявлення про те, які заходи вони вжили, щоб уникнути атак зворотного проектування безпеки.

Тут ми перемінюємо файл {app}.apk на {app}.zip і розпаковуємо його. У розпакованій папці знайдено файл classes.dex, який містить код програми.

Файл DEX — це виконуваний файл, який містить скомпільований код і працює на платформі Android.

Використовуємо файл classes.dex, який взяли з zip-файлу Тепер APK, і перетворимо його на JAR. Для цього можна скористатися інструментом з відкритим кодом «dex2jar», доступним тут. Перейдіть до розділу випуску, завантажте найновіший доступний файл zip і розпакуйте його. Скопіюємо витягнутий файл classes.dex і вставимо його в каталог «dex2jar-x.x».

Відкриємо термінал на вашій машині та перейдемо до каталогу «dex2jar-x.x». Тепер ми виконаємо команду -

d2j-dex2jar.bat classes.dex

Це перетворити файл classes.dex у файл JAR, який можна переглянути за допомогою будь-якої декомпіляції.

Третій крок — будемо використовувати JD JUI, який є простим інструментом Java Decompiler. Запустимо jd-gui.exe, відкриємо файл dex і готово, у нас є повний робочий код нашої програми, який ми можемо використовувати в різних цілях: будемо шукати вразливості в програмному коді і використовувати їх для викрадання персональних даних у майбутньому або ж ми просто заволоділи інтелектуальною власна компанія, яка випустила даний застосунок.

Для того, щоб зберегти код від зворотного проектування, потрібно використовувати ProGuard. Щоб зробити ваш додаток якомога меншим, вам слід увімкнути стискання у збірці випуску, щоб видалити невикористаний код і ресурси. Увімкнувши стиснення, ви також отримуєте переваги від обфускації, яка скорочує назви класів і членів вашої програми, і оптимізації, яка застосовує більш агресивні стратегії для подальшого зменшення розміру вашої програми. На цій сторінці описано, як R8 виконує ці завдання під час компіляції для вашого проекту та як ви можете їх налаштувати.

Коли ви створите свій проект за допомогою плагіна Android Gradle 3.4.0 або новішої версії, плагін більше не використовує ProGuard для оптимізації коду під час компіляції. Натомість плагіна працює з компілятором R8 для виконання наступних завдань під час компіляції:

Згортання коду (або тремтіння дерева): ви показує та безпечно видаляє невикористовувані класи, поля, методи та атрибути з вашої програми та її бібліотечних залежностей (що робить її цінним інструментом для роботи з межею посилань у 64 КБ). Наприклад, якщо ви використовуєте лише кілька API залежно від бібліотеки, натиснення може ідентифікувати код бібліотеки, яку ваша програма не використовує, і видалити цей код із вашої додатки. Щоб дізнатися більше, перейдіть до розділу про те, як зменшити код.

Скорочення ресурсів: видаляє невикористані ресурси з пакетної програми, виключно з невикористаними ресурсами в залежностях бібліотеки програми. Він працює в поєднанні зі стисненням коду, тому після видалення невикористаного коду всі ресурси, на які більше не посилаються, також можна безпечно видалити. Щоб дізнатися більше, перейдіть до розділу про те, як зменшити свої ресурси.

Обфускація: скорочує назви класів і членів, що призводить до зменшення розмірів файлів DEX. Щоб дізнатися більше, перейдіть до розділу про обфускацію коду.

Оптимізація: перевіряє та переписує ваш код, щоб більше зменшити розмір файлів DEX вашої програми. Наприклад, якщо R8 виявляє, що гілка else {} для даного оператора if/else ніколи не потребується, R8 видає код для гілки else {}. Щоб дізнатися більше, перейдіть до розділу про оптимізацію коду.

Під час створення випускної версії вашої програми для замовчування R8 автоматично виконується для вас описані вище завдання під час компіляції. Однак ви можете отримати певні завдання або налаштувати поведінку R8 за допомогою файлів правил ProGuard. Насправді R8 працює з усіма наявними файлами за правилами ProGuard, тому оновлення плагіна Android Gradle для використання R8 не вимагається від зміни існуючих правил.

Хоча обфускація не видаляє код із вашої програми, значну економічну величину можна побачити в програмах із файлами DEX, які індексують багато класів, методів і полів. Однак інша обфускація змінює різні частини вашого коду, певні завдання, як така перевірка трасування стека, потребують додаткових

інструментів. Щоб зрозуміти трасування стека після обфускації, прочитайте розділ про те, як декодувати трасування стека.

Крім того, ваш код розміщується на передбачуваних іменах для методів і класів вашої додатки, під час використання відображення, ви повинні розглядати ці підписи, наприклад, якщо точки входу та вказати для них правила збереження, як описано в розділі про те, як на налаштованому коді для бути. Ці правила збереження наказують R8 лише не зберігають цей код в остаточному DEX вашої додатки, але й зберігають його оригінальне найменування.

Саме зміна імені класів, функцій та назви параметрів, що дає їм ім'я без будь-якої змістової навантаження, майже неможливе прочитання коду після його декомпіляції та зберігає всі його секрети.

4.9. Перевірка девайсу на рутованість

Багато людей хочуть рутувати смартфони Android, щоб вони могли встановлювати різноманітні програми сторонніх розробників або подолати певні системні обмеження, які зазвичай встановлюють виробники обладнання та оператори.

Хоча деякі телефони можуть отримати рутований доступ, більшість із них – ні. Є кілька простих і безкоштовних способів перевірити, чи є у вас рутований телефон. У цій статті ви знайдете три способи, два з яких надійні, а один може виникнути в певній ситуації залежно від моделі вашого телефону.

Не плутати з джейлбрейком (на пристроях iOS), це метод розблокування пристрою Android, щоб надати користувачеві привілейований контроль або root-доступ. Це дуже схоже на наявність прав адміністратора в ОС Windows або Linux.
API цілісності Play допомагає захистити ваші програми та ігри від потенційно ризикованих і шахрайських взаємодій, таких як шахрайство та несанкціонований доступ, дозволяючи вам реагувати відповідними діями, щоб запобігти атакам і зменшити зловживання.

Коли ваша програма чи гра використовується на пристрої з ОС Android 4.4 (рівень API 19) або новішої версії, API цілісності Play надає підписану та зашифровану відповідь, яка містить таку інформацію:

Справжній двійковий файл програми: визначте, чи взаємодієте ви зі своїм немодифікованим двійковим файлом, який розпізнає Google Play.

Genuine Play install: визначте, чи поточний обліковий запис користувача має ліцензію, що означає, що користувач встановив або оплатив вашу програму чи гру в Google Play.

Справжній пристрій Android: це вказує вам, чи працює ваша програма на оригінальному пристрої Android із сервісами Google Play.

Коли ви вмикаєте перевірку програми для служби та включаєте клієнтський SDK у свою програму, періодично відбувається таке:

Ваша програма взаємодіє з обраним вами постачальником, щоб отримати підтвердження автентичності програми чи пристрою (або обох, залежно від постачальника).

Атестація надсилається на сервер App Check, який перевіряє дійсність атестації за допомогою параметрів, зареєстрованих у програмі, і повертає вашій програмі маркер App Check із часом закінчення дії. Цей маркер може зберігати деяку інформацію про матеріал атестації, який він перевірив.

Клієнтський SDK App Check зберігає маркер у вашій програмі, готовий до надсилання разом із будь-якими запитами, які ваша програма надсилає до захищених служб. Служба, захищена App Check, приймає лише запити, які супроводжуються поточним дійсним маркером App Check.

App Check покладається на потужність своїх постачальників атестації для визначення автентичності програми чи пристрою. Це запобігає деяким, але не всім векторам зловживань, спрямованих на ваші серверні частини. Використання App Check не гарантує усунення всіх зловживань, але, інтегрувавши App Check, ви робите важливий крок до захисту ваших серверних ресурсів від зловживань.

4.10. Проблема обфускації при використанні kotlin nullable

Кожна серйозна програма повинна використовувати певну форму інструменту для стиснення коду та обфускації. В Android поширеними інструментами є ProGuard і нещодавно R8.

Ці інструменти повністю підтримуються Kotlin без додавання спеціальних правил (крім деяких особливих випадків, які не розглядатимуться в цій публікації)
Це означає, що ви можете запустити ProGuard у своїй програмі Kotlin, і все має працювати як задумано без збоїв. Зверніть увагу, що згенерований код викликає клас під назвою Intrinsics. Ці виклики в основному є твердженнями про різні обмеження Kotlin, як у цьому прикладі, параметри не повинні бути нульовими або повинні бути ініціалізовані перед доступом. Змінні та імена методів розширення все ще присутні!
Ці імена можуть розкрити нашу бізнес-логіку стороннім особам, і ця інформація може бути використана у шкідливий спосіб. Це не головна проблема безпеки, але все ж проблема безпеки. Правила ProGuard можна використовувати не тільки для запобігання обфускації чи видаленню коду, їх також можна використовувати для видалення коду. Додавання наступного правила ProGuard до вашого файлу proguard-rules.pro призведе до видалення всіх наступних викликів методів до класу Intrinsics.

```
-assumenosideeffects class kotlin.jvm.internal.Intrinsics {
  public static void checkExpressionValueIsNotNull(java.lang.Object, java.lang.String);
  public static void checkFieldIsNotNull(java.lang.Object, java.lang.String);
  public static void checkFieldIsNotNull(java.lang.Object, java.lang.String, java.lang.String);
  public static void checkNotNull(java.lang.Object);
  public static void checkNotNull(java.lang.Object, java.lang.String);
  public static void checkNotNullExpressionValue(java.lang.Object, java.lang.String);
  public static void checkNotNullParameter(java.lang.Object, java.lang.String);
  public static void checkParameterIsNotNull(java.lang.Object, java.lang.String);
  public static void checkReturnedValueIsNotNull(java.lang.Object, java.lang.String);
  public static void checkReturnedValueIsNotNull(java.lang.Object, java.lang.String, java.lang.Str
  public static void throwUninitializedPropertyAccessException(java.lang.String);
}
```

Рис. 4.9. Проперті в Proguard

ВИСНОВКИ ДО РОЗДІЛУ 4

Отже ми розглянули найбільш актуальні способи взлому мобільних додатків або ж несанкіонованого доступу до даних. Як ми можемо розуміти ми як розробники ПЗ повинні виконувати всі вищенаведені вимоги по захисту мобільного додатку від взлому.

ВИСНОВКИ

З появою операційних систем, відкрилася нова епоха програмування, яка дала поштовх в багато нових напрямків. В нас з'явилася можливість транспортувати, здавалося б доволі велику програму, яка потребує потужного заліза для підтримки, таку як операційну систему на маленькі девайси (смартфони) які можна легко взяти з собою в будь яку точку світу і користуватися.

Але з кожним новим проривом в програмуванні, потрібно бути обачним, так як його можуть використати зловмисники в своїх цілях задля наживи. Щоб запустити нову операційну систему, яка є доволі громіздкою програмою, перш за все потрібно продумати її внутрішню безпеку, щоб зробити її максимально корисною для користувача. На жаль, ОС Андроїд поспішив з релізом своєї системи, так як на той час, коли саме зароджувалися розумні телефони (смартфони), вже виступив на ринок доволі значний гравець як Apple з своїм новим iPhone, який приніс революцію і нові ідеї в світ телефонів. Тому Андроїд, щоб не втратити ринок остаточно, вирішив зарелізити свою версію ОС для мобільних телефонів, яка доволі бистро набрала популярність через свою простоту та дешевизну в порівняні з Apple. Тому ринок Андроїд значно швидше поширився і зайняв майже 73 відсотки світого ринку ОС для мобільних смартфонів. Хоча це чудово що Андроїд поширився так, проте він є дуже не безпечним в порівняні з Apple через його сирий вигляд в релізі. Прикладом цього є непродумана безпека Андроїда, яка дозволяла робити все що завгодно користувачу (хоча так само і зробив Linux, батько Андроїд), але і можливість задіяти шкоди своєму телефону чи даним. Також ми можемо знайти недотримання одного з правил яке було описане в другому розділі цієї дипломної роботи про код рев'ю в частинах AOSP (Android Open Source Project) відкритий код Андроїда, де є спеціально створені так звані бекдори для несанкціонованого доступу до даних телефону. На щастя, з кожним виходом нового Андроїда, розробники посилюють безпеку ОС шляхом урізання певних прав доступу які були надані в минулих версіях, але, як на мене, це варте того щоб дані користувача були в безпеці.

В цій дипломній роботі було розглянуто методи розробки ПЗ в команді, для того щоб забезпечити код від викрадення сторонніми особами і заволодінню інтелектуальною власністю. Також досліджено вразливість додатків для ОС Андроїд, причини їх винекнення, а також способи вирішення. Однією з мети було сторення перелік найкращих практик по розробці безпечних та захищених від проникнення мобільних додатків для ОС Андроїд які наведені нижче:

1) Проведення код рев'ю
2) Розробка ПЗ з використанням VPN
3) Використання статичних код аналізаторів Ktlint, Detekt
4) Написання модульних тестів
5) Використання практик CI/CD
6) Зберігання ключів доступу в GitHub (або в іншому віддаленому репозиторію, але не в проекті)
7) Запобігання сторення бекдорів програмістами (за допомогю код рев'ю)
8) Використання лише безпечного зєднання при restful запитах
9) Постійне оновлення до останніх стабільних бібліотек
10) Timber – бібліотека для логування
11) Бути обережним з WebView компонентоз з вкрапленням JS коду
12) Постійне (!) використання proguard з включеною обфускацією і шрінкінгом
13) Винесення важливого коду в C++ шар архітектури
14) Потрібно перевіряти чи девайс не рутований
15) Збереження логіну та паролю в безпечному місці таке як SavedSharedPrefs

Це основні правила для розроки ПЗ для мобільного додатку Андроїд. Звичайно є і інші способи викрасти дані користувача, так як жодна система не є досконалою і це є вічна гонка між розробниками ОС та хакерами. Вона заключається в тому хто швидше знайте певний баг в Андроїд та використати його для взлому мобільного телефону чи додатку. Іншими словами, це є холодна війна, між хакером і розробником, які завжди використовують різні способи виграти раунд в один одного, і вона ніколи не закінчиться через стрімкий розвиток технологій.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ

1. Operational System: Frontier. [Електронний ресурс]. Режим доступу: https://www.guru99.com/operating-system-tutorial.html (дата звернення 01.10.2022). – Назва з екрана.
2. Linux Software Development. [Електронний ресурс]. Режим доступу: https://digitaltechakshay.medium.com/what-is-the-ios-operating-system-b19c5d19f5bc (дата звернення 05.10.2022). – Назва з екрана.
3. iOS Development. [Електронний ресурс]. Режим доступу: https://www.makeuseof.com/tag/what-is-ios/ (дата звернення 10.10.2022). – Назва з екрана.
4. Henrik Kniberg. Scrum and XP from the trenches. — C4Media, 2007. — С. 140. — ISBN 978-1-4303-2264-1.
5. Linux . [Електронний ресурс]. Режим доступу: https://www.redhat.com/en/topics/linux/what-is-linux (дата звернення 15.11.2022). – Назва з екрана.
6. Extreme programming. [Електронний ресурс]. Режим доступу: http://www.xprogramming.com/xpmag/whatisxp.htm (дата звернення 17.10.2022). – Назва з екрана.
7. Linux Redhut. [Електронний ресурс]. Режим доступу: https://www.ntchosting.com/encyclopedia/hosting/linux-operating-system/ (дата звернення 18.10.2022). – Назва з екрана.
8. What is Android? [Електронний ресурс]. Режим доступу: https://www.techtarget.com/searchmobilecomputing/definition/Android-OS (дата звернення 26.10.2022). – Назва з екрана.
9. Mobile Market. [Електронний ресурс]. Режим доступу: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення 28.10.2022). – Назва екрана.

10. Android Studio. [Електронний ресурс]. Режим доступу: https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio (дата звернення 29.10.2022). – Назва з екрана.
11. Android Studio. [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/android-studio (дата звернення 30.10.2022). – Назва з екрана.
12. Android Emulator. [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/android-emulator (дата звернення 30.10.2022). – Назва з екрана.
13. Git. [Електронний ресурс]. Режим доступу: https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/ (дата звернення 30.10.2022). – Назва з екрана.
14. Android Studio Build. [Електронний ресурс]. Режим доступу: https://developer.android.com/studio/build (дата звернення 30.10.2022). – Назва з екрана
15. Kotlin Programming Language [Електронний ресурс]. Режим доступу: https://kotlinlang.org/spec/introduction.html (дата звернення 25.10.2022). – Назва з екрана.
16. Git essential [Електронний ресурс]. Режим доступу: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git (дата звернення 26.10.2022). – Назва з екрана.
17. VPN. [Електронний ресурс]. Режим доступу: https://messapps.com/allcategories/development/why-developers-need-vpn/ (дата звернення 27.10.2020). – Назва з екрана.
18. Code review [Електронний ресурс]. Режим доступу: https://www.3pillarglobal.com/insights/the-importance-of-code-reviews/ (дата звернення 28.10.2022). – Назва з екрана.
19. Code review process. [Електронний ресурс]. Режим доступу: https://www.brightspot.com/learn/articles/5-reasons-why-the-code-review-process-is-critical-for-developers (дата звернення 23.11.2022). – Назва з екрана.

20. Coexisting Plan-driven and Agile Methods: How Tensions Emerge and Are Resolved Completed Research Paper. [Електронний ресурс]. Режим доступу: https://www.researchgate.netPlan-driven_and_Agile_Methods (дата звернення 01.11.2022). – Назва з екрана.
21. Quality code review. [Електронний ресурс]. Режим доступу: https://www.softwaretestinghelp.com/code-quality-tools/ (дата звернення 23.11.2020). – Назва з екрана.
22. Muhammad, Ali Babar, Alan W. Brown, Ivan Mistrik. Agile software architecture: aligning agile processes and software architectures. - Morgan Kaufmann, 2013. – с. 432. – ISBN 9780124077720.
23. Using Social Network Analysis to Investigate the Collaboration Between Architects and Agile Teams. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/Social_Network_Analysis_Adile (дата звернення 10.11.2022). – Назва з екрана.
24. Застосування архітектурного проектування в гнучких методах розробки програмних продуктів. [Текст] / Харченко О.Г., Боднарчук І.О., Галай І.О., Лісовий В. // Інженерія програмного забезпечення. – 2012. – № 3–4 (11–12). – с. 5–11.
25. Лавріщева К.М. Програмна інженерія. / К.М. Лавріщева –К.: Видавничий дім "Академперіодика", 2008.– 319 с.
26. Харченко О.Г. Проектування архітектури web-застосувань на основі моделі якості. /О.Г. Харченко, І.О. Галай, І.О. Боднарчук, В.В. Яцишин// Інженерія програмного забезпечення. № 4, 2010. – С. 26 – 34.
27. Garlan, David. An Introduction to Software Architecture: Technical Report/ David Garlan, Mary Shaw. – Carnegie Mellon University Pittsburgh, PA, USA, 1994. – 42 p.
28. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. – СПб. : Питер, 2010. – 366 с.

29. Руководство Microsoft по проектированию архитектуры приложений (2-е издание) / – Корпорация Майкрософт, 2009. – 528 с.
30. Saaty T. Decision Making with the Analytic Network Process./ Saaty T. Vargas L.// – N.Y.: Springer, 2006. 278 p
31. Harchenko Alexandr, Bodnarchuk Ihor, Halay Iryna. Stability of the Solutions of the Optimization Problem of Software Systems Architecture // Proceeding of VIIth International Scientific and Technical Conference CSIT 2012. pp. 47–48, Lviv, 2012
32. Coordinating Knowledge Work in Multi-Team Programs: Findings from a Large-Scale Agile Development Program. [Електронний ресурс]. Режим доступу:https://www.researchgate.net/publication/Coordinating_Knowledge_Work_in_Multi-Team_Programs (дата звернення 16.11.2020). – Назва з екрана.
33. Studying Onboarding in Distributed Software Teams: A Case Study and Guidelines. [Електронний ресурс]. Режим доступу: https://dl.acm.org/Studying_Onboarding (дата звернення 19.11.2020). – Назва з екрана.
34. Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. [Електронний ресурс]. Режим доступу: https://ieeexplore.ieee.org/Coordination Challenges (дата звернення - 03.11.2020). – Назва з екрана.
35. Методы классической и современной теории автоматического управления. Учебник в 3-х т. Т.3: Методы современной теории автоматического управления / Ред. Н.Д. Егупов. М.: Изд-во МГТУ им. Н.Э. Баумана, 2000. 748 с.
36. Ларичев О.И., Мошкович Е.М. Качественные методы принятия решений. Вербальный анализ решений. М.: Наука. Физматлит, 1996. 208 с.
37. Применение теории координации для управления качеством технологических процессов со слабо формализуемыми критериями. [Електронний ресурс]. Режим доступу: https://cyberleninka.ru/article/primenenie-teorii-koordinatsii (дата звернення 18.11.2020). – Назва з екрана.

38. Боулдинг К. Общая теория систем – скелет науки // Исследования по общей теории систем. М.: Прогресс, 1969. С. 106–124.
39. Хакен Г. Синергетика. Иерархия неустойчивостей в самоорганизующихся системах и устройствах. М.: Мир, 1985.
40. Месарович М., Мако Д., Такахара И. Теория иерархических многоуровневых систем. М.: Мир, 1973. 344 с.
41. Проектирование автоматизированной системы контроля и управления доступом на предприятии [Електронний ресурс] – Режим доступу до ресурсу: https://knowledge.allbest.ru/programming/3c0b65625a3bd69a5d53b89521316c37_0.html.
42. Офіційна документація бази MySQL [Електронний ресурс]. – 2009. – Режим доступу до ресурсу: https://dev.mysql.com/doc/.
43. TDD [Електронний ресурс] 2019 – Режим доступу до ресурсу: https://www.manning.com/books/junit-in-action-third-edition?query=tdd
44. Testing Java [Електронний ресурс] 2018 – Режим доступу до ресурсу: https://www.manning.com/books/testing-java-microservices?query=java
45. Java 8 in Action [Електронний ресурс] 2018 – Режим доступу до ресурсу: https://www.manning.com/books/java-8-in-action?query=java
46. Web Components in Action [Електронний ресурс] 2019 – Режим доступу до ресурсу: https://www.manning.com/books/web-components-in-action?query=web%20archi
47. SQL in Motion [Електронний ресурс] 2017 – Режим доступу до ресурсу: https://www.manning.com/livevideo/sql-in-motion?query=mysql