**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Обеспечение информационной безопасности мобильных устройств»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

«Основы анализа защищенности мобильных приложений»

**Выполнили:**

Ахраров Али, студент группы N3350

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Федоров Иван Романович

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(отметка о выполнении)

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Содержание

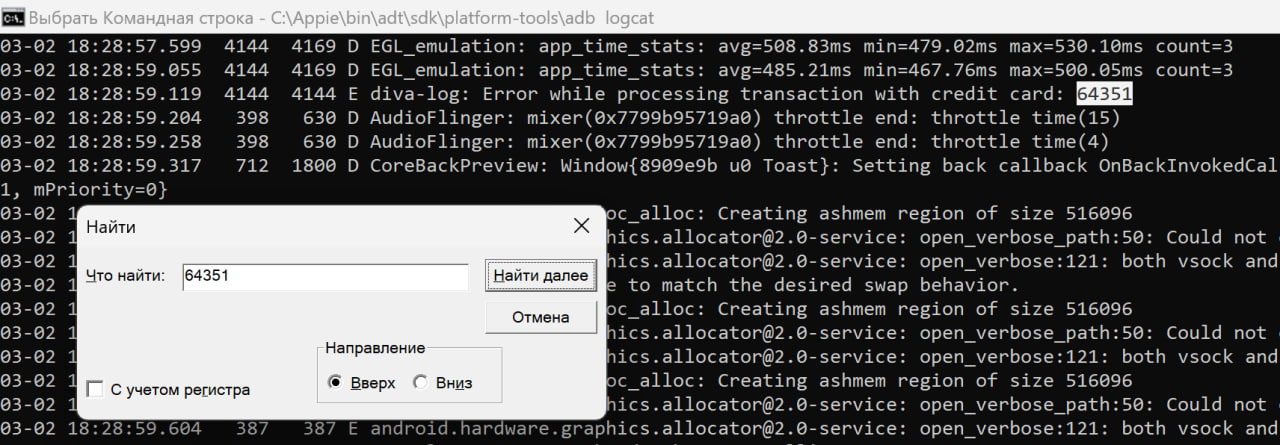
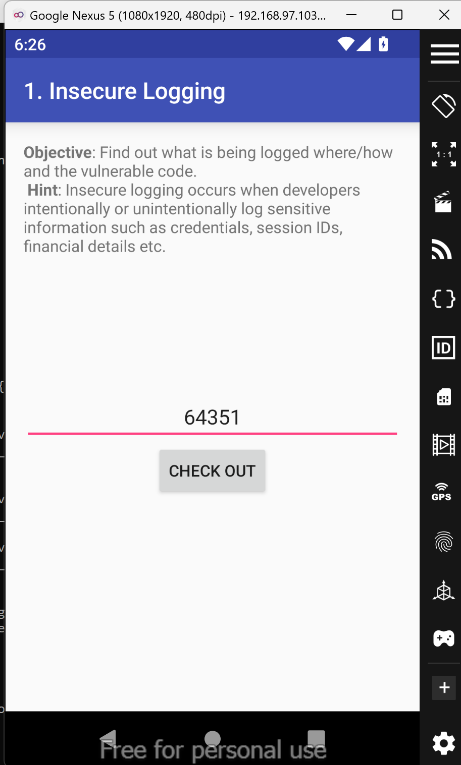
[Поиск уязвимостей 3](#_Toc191995556)

[Безопасность хранения и реализации 13](#_Toc191995557)

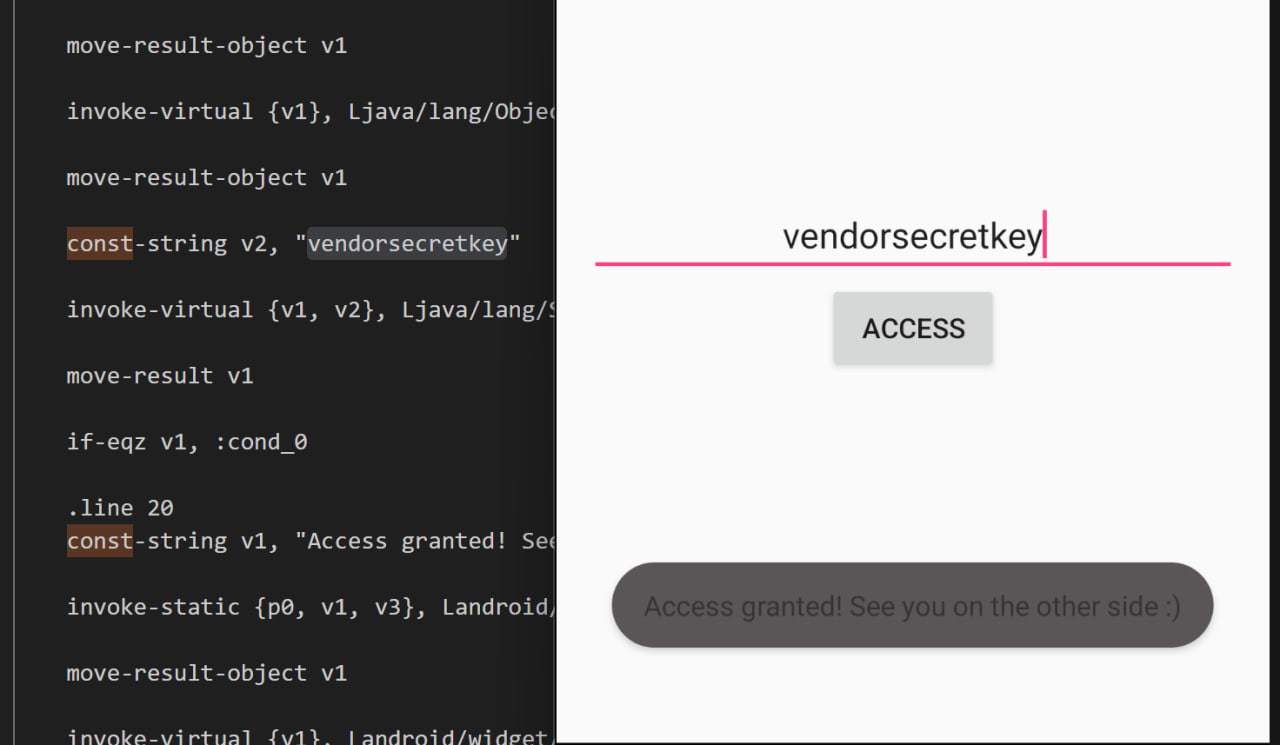
[Выводы и рекомендации 14](#_Toc191995558)

# Поиск уязвимостей

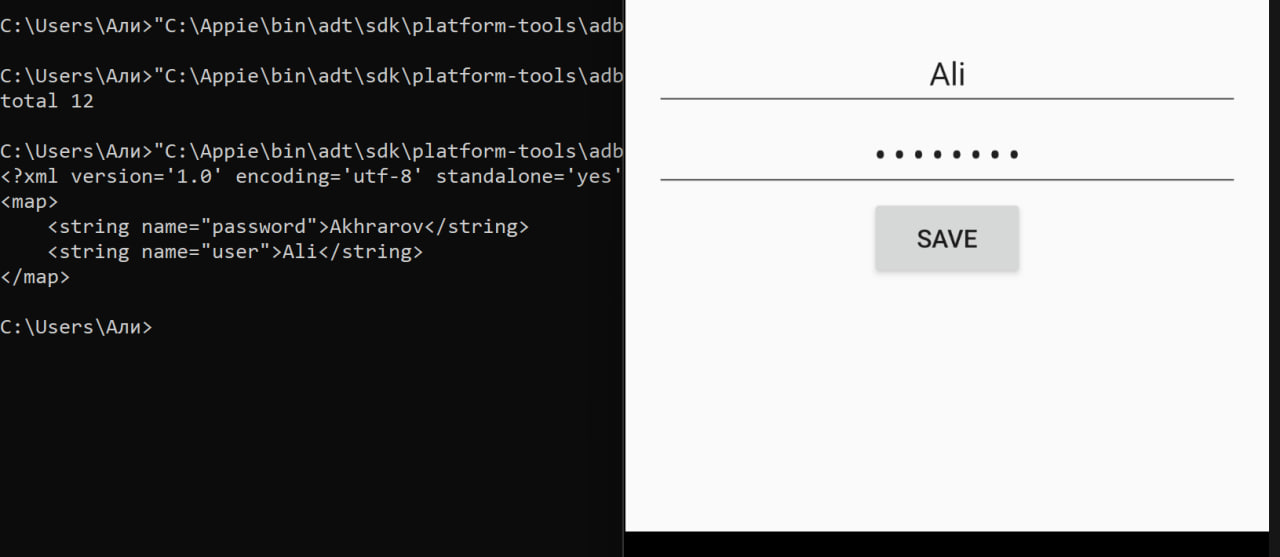
1. **Insecure Logging.** Приложение записывает конфиденциальные данные (например, номер кредитной карты) в системные логи. При вводе произвольного числа (например, 64351) в поле и нажатии кнопки «Check out», в logcat фиксируется строка вида «Error while processing transaction with credit card: 64351». Любой, имеющий доступ к logcat, может прочитать эти данные.

  
*Рисунок 1 – Пример вывода logcat, где отображён номер карты.*

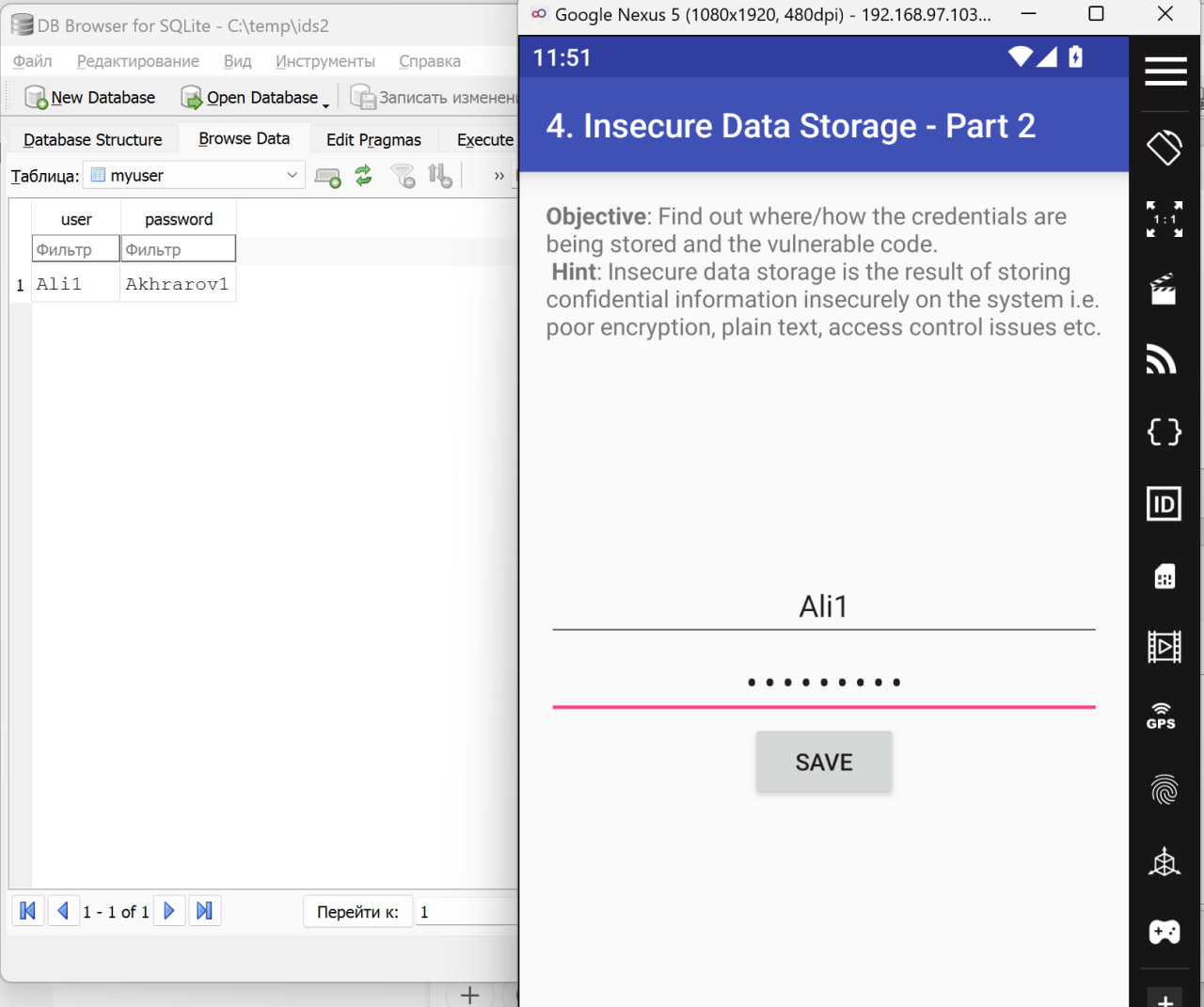
1. **Hardcoding Issues – Part 1.** В активности APICredsActivity жёстко прописаны данные (API Key, логин, пароль), которые сразу выводятся в TextView. При декомпиляции кода видно, что метод onCreate() напрямую устанавливает текст «API Key: 123secretapikey123…». Запустив эту активность извне или анализируя smali, злоумышленник получает ключи.

  
*Рисунок 2 – Экран APICredsActivity с зашитыми API-ключами.*

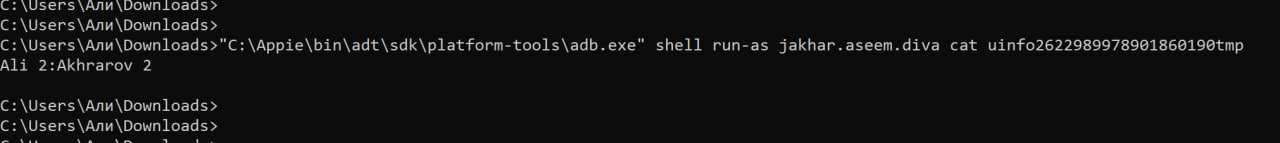
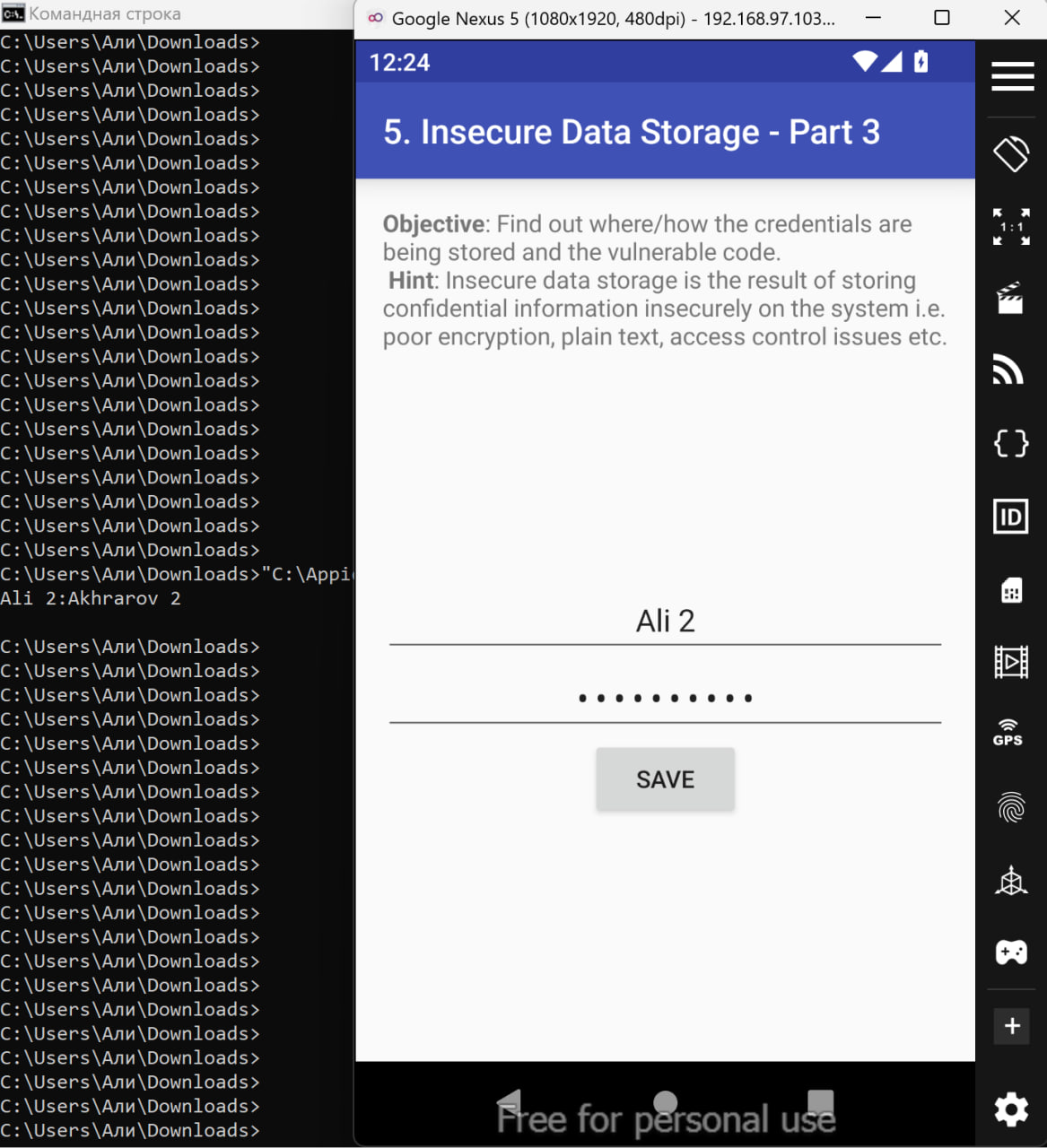
1. **Insecure Data Storage – Part 1.** Приложение хранит третьесторонние учётные данные (например, Ali, Akhrarov) в Shared Preferences без шифрования. Через команды adb shell run-as можно вывести содержимое jakhar.aseem.diva\_preferences.xml и увидеть <string name="user">Ali</string>, <string name="password">Akhrarov</string>. Это значит, что злоумышленник, имея доступ к устройству, прочитает пароли.

  
*Рисунок 3 – Содержимое Shared Preferences.*

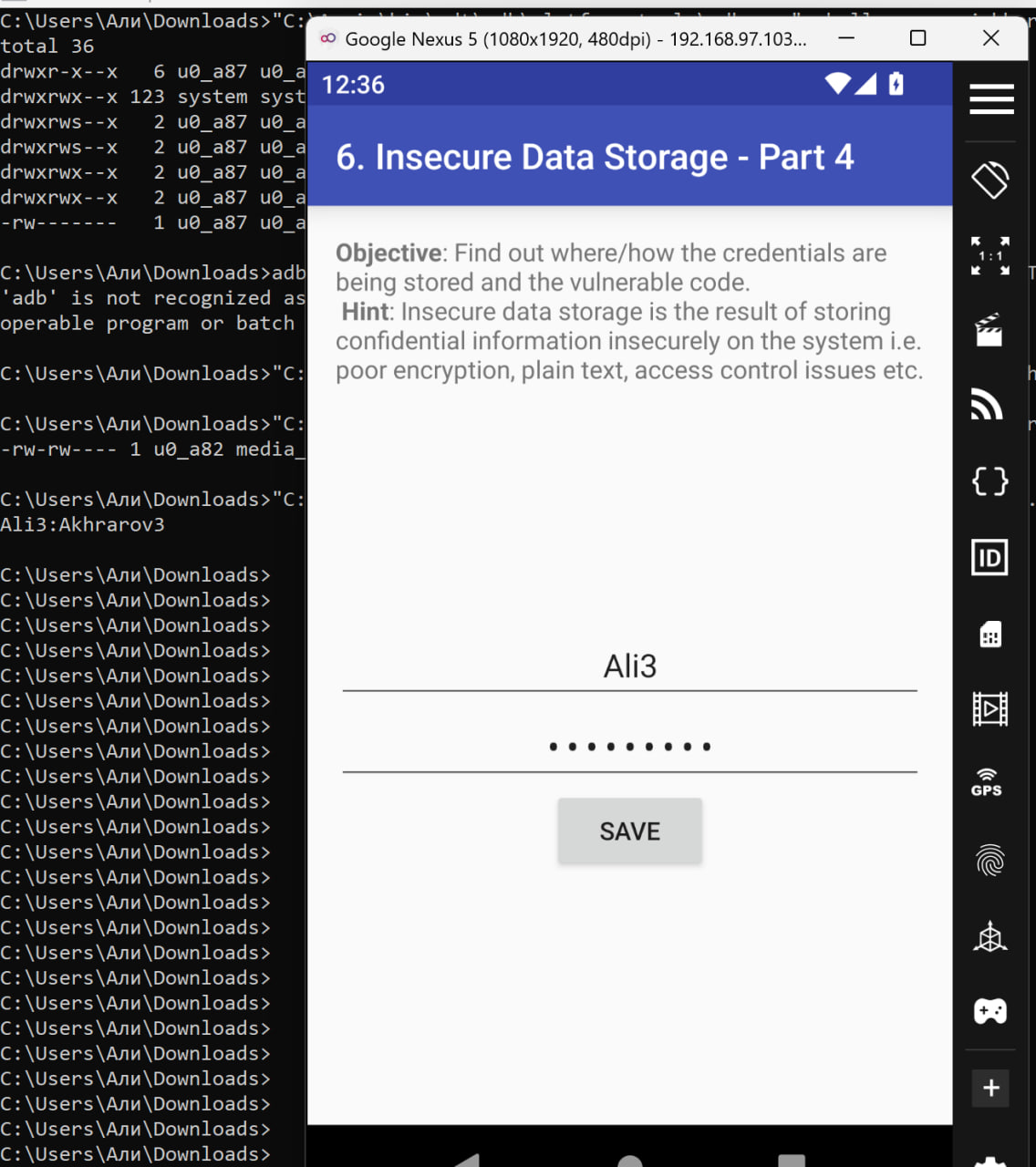
1. **Insecure Data Storage – Part 2.** В аналогичной логике приложение хранит данные (Ali1, Alkhrarov1) в базе divanotes.db или ids2 без шифрования. Мы извлекаем базы (командами adb shell run-as, затем base64 и certutil) и видим логин/пароль в открытом виде (Ali1, Alkhrarov1).

  
*Рисунок 4 – База данных, открытая в DB Browser for SQLite.*

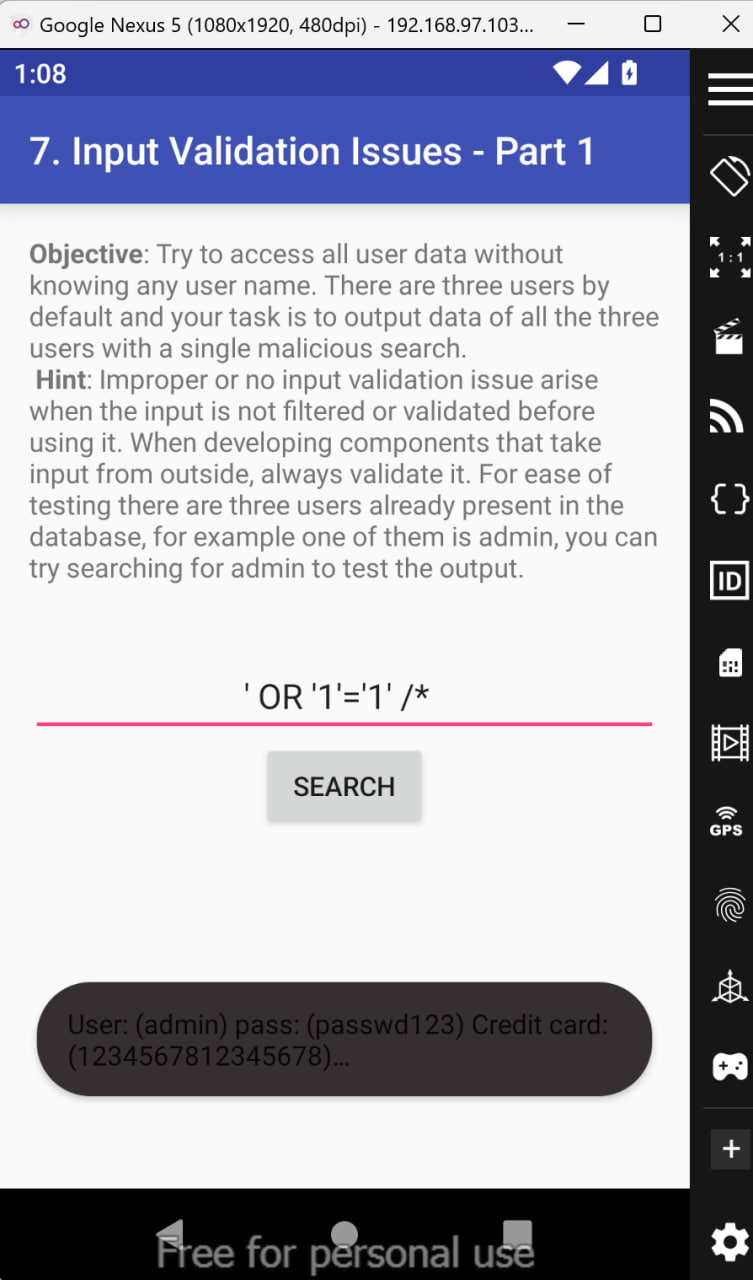
1. **Insecure Data Storage – Part 3.** Приложение сохраняет данные во временном файле (uinfo.tmp или .uinfo.txt) на SD-карте. После выдачи WRITE\_EXTERNAL\_STORAGE можно прочитать файл через adb shell cat /sdcard/.uinfo.txt и увидеть конфиденциальную информацию. Это демонстрирует отсутствие шифрования и доступ к файлу любому приложению.

  
*Рисунок 5 – Файл .uinfo.txt с логином/паролем.*

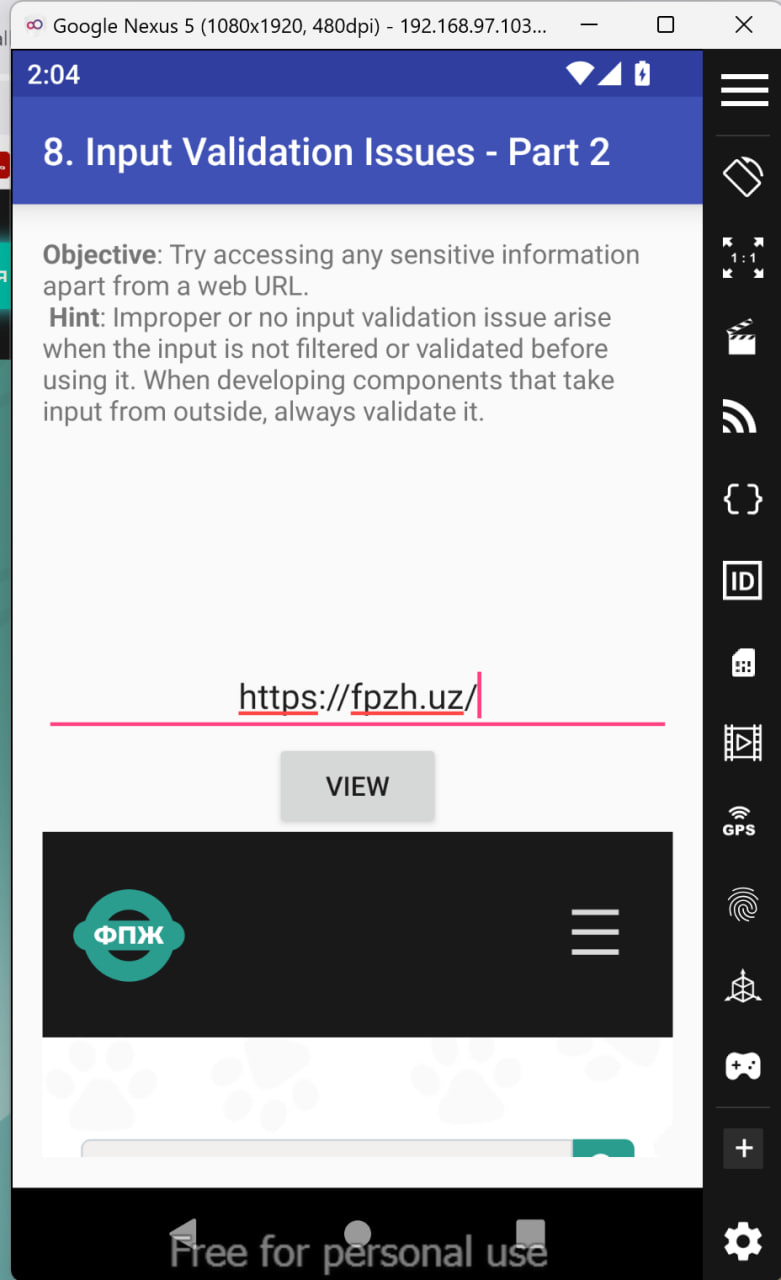
1. **Insecure Data Storage – Part 4.** Приложение пытается записать данные во внешнее хранилище, но выводит «File error occurred». Несмотря на ошибку, сама логика указывает на попытку небезопасного хранения без шифрования. Это тоже уязвимость, так как приложение не защищает данные и не обрабатывает ошибки корректно.

  
*Рисунок 6 – Сообщение об ошибке при сохранении данных.*

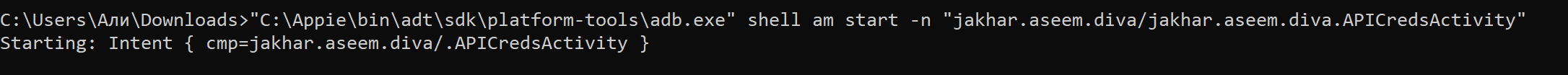
1. **Input Validation Issues – Part 1.** В активности SQLInjectionActivity формируется запрос вида SELECT \* FROM sqliuser WHERE user = 'ввод'. При вводе admin' OR '1'='1 приложение возвращает все записи (admin, diva, john). Это классическая SQL-инъекция из-за отсутствия фильтрации.

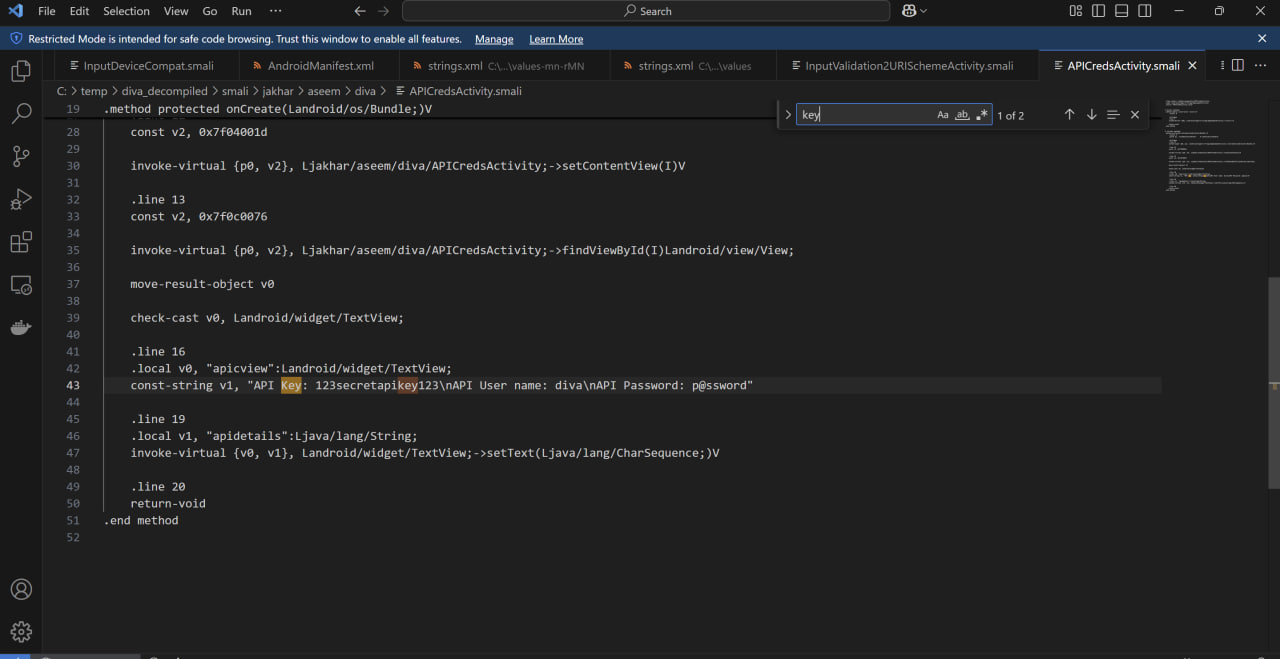
  
*Рисунок 7 – Результат SQL-инъекции, отображающий все учётные данные.*

1. **Input Validation Issues – Part 2.** Активность InputValidation2URISchemeActivity передаёт пользовательский ввод напрямую в WebView.loadUrl(...). Ввод javascript: или file:// может привести к атакам, хотя современные WebView блокируют часть схем. Факт отсутствия проверки URI остаётся уязвимостью, позволяя злоумышленнику перенаправлять на фишинговые сайты или пытаться исполнить скрипты.

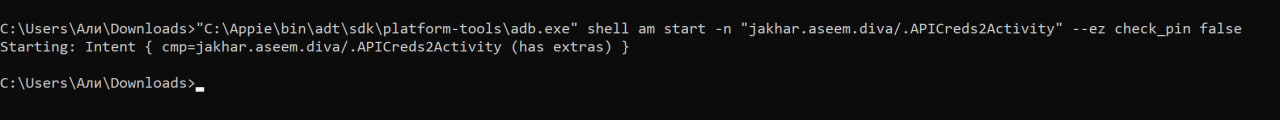
  
*Рисунок 8 – Пример ввода URI и сообщение «Page not available».*

1. **Access Control Issues – Part 1.** Активность APICredsActivity экспортирована и не требует проверки. Запуск через команду adb shell am start -n "jakhar.aseem.diva/.APICredsActivity" мгновенно показывает «API Key: 123secretapikey123…». Таким образом, любой злоумышленник, знающий имя пакета и активности, получает конфиденциальные данные.



  
*Рисунок 9 – Запуск APICredsActivity извне и отображение ключей.*

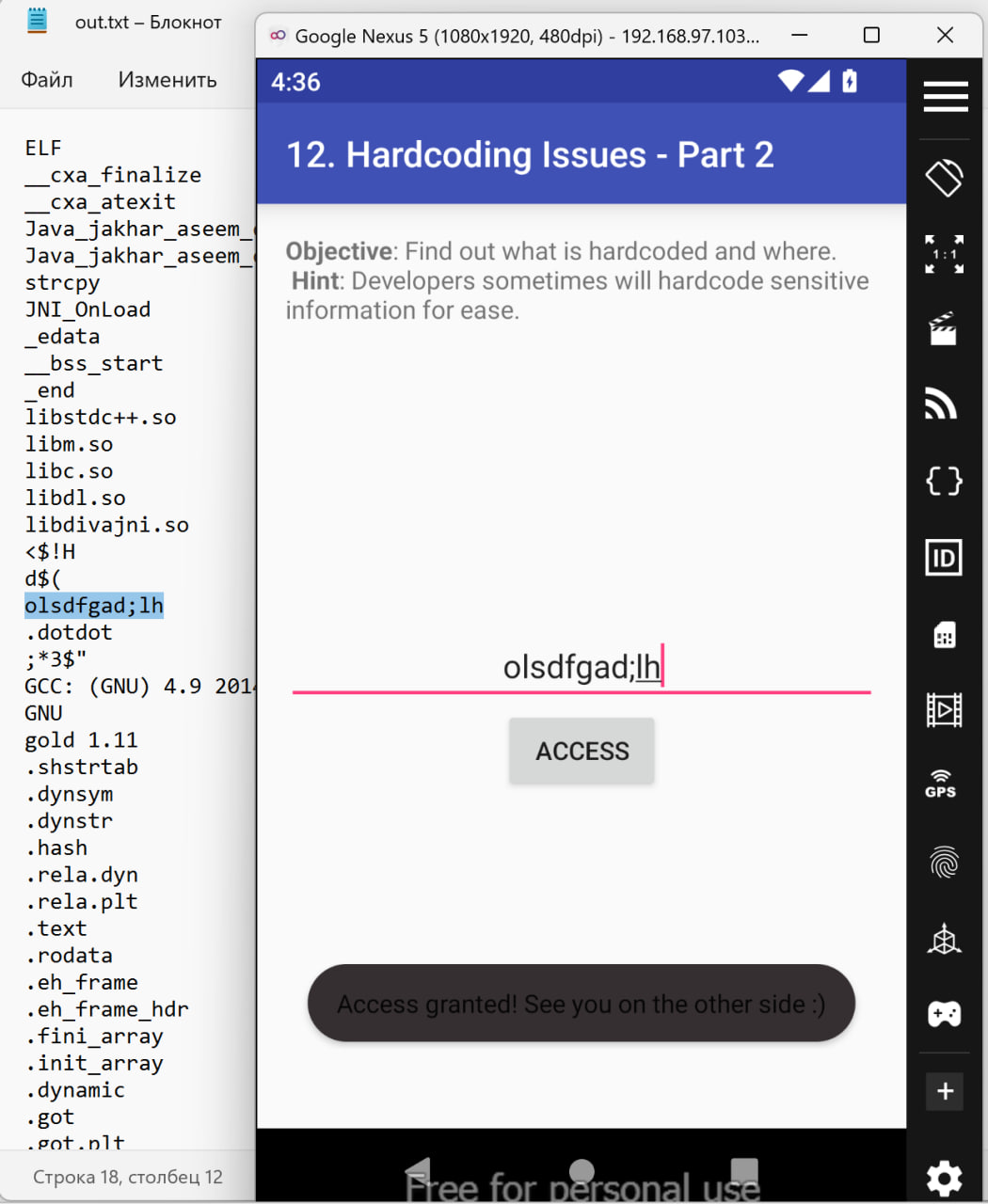
1. **Access Control Issues – Part 2.** В APICreds2Activity приложение требует PIN, но если при запуске активности передать Intent extra (например, --ez check\_pin false), проверка обходится. В итоге сразу показываются Tveeter API credentials (логин/пароль). Это демонстрирует обход бизнес-логики.

  
*Рисунок 10 – Запуск APICreds2Activity с параметром check\_pin=false и вывод секретов без PIN.*

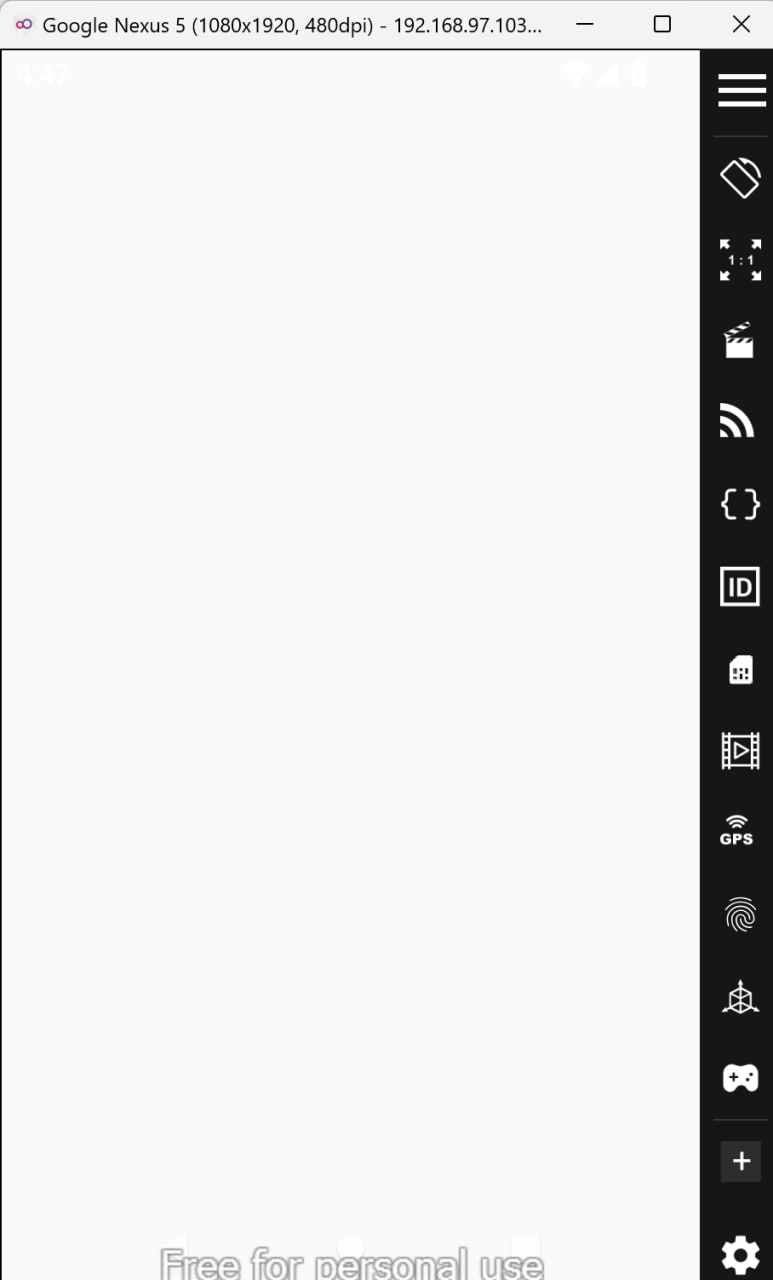
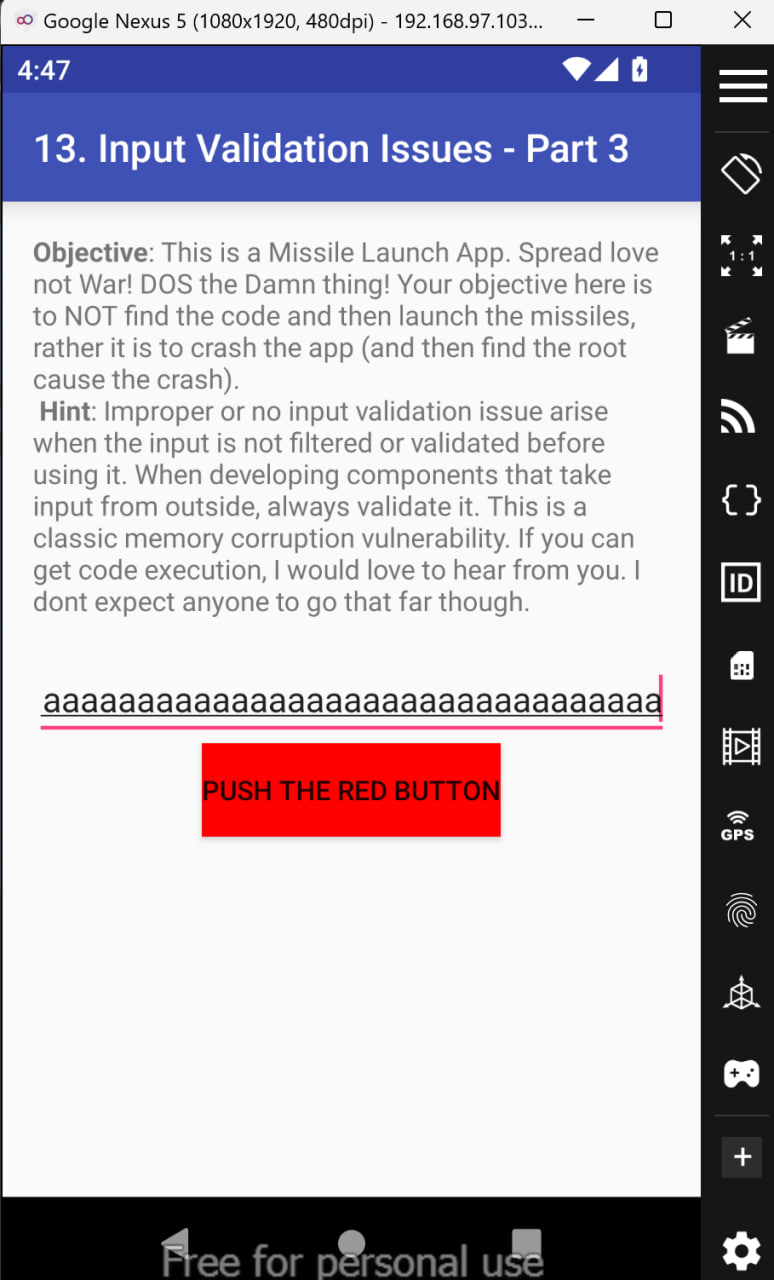
1. **Access Control Issues – Part 3.** Приватные заметки (private notes) защищаются PIN в приложении, но Content Provider notesprovider экспортирован и доступен без аутентификации. Команда adb shell content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes выводит все заметки (office, home, holiday и т.д.), игнорируя PIN. Это прямая утечка данных.

  
*Рисунок 11 – Вывод команд content query, показывающий все заметки.*

1. **Hardcoding Issues – Part 2 (продолжение).** В нативной библиотеке libdivajni.so для задания Hardcode2Activity обнаружен ещё один ключ (vendorsecretkey), который можно найти с помощью strings. При вводе vendorsecretkey в поле приложения отображается «Access granted!». Хранение ключа в .so небезопасно, так как его легко извлечь из памяти приложения.

  
*Рисунок 12 – Ввод «vendorsecretkey» и сообщение «Access granted!».*

1. **Input Validation Issues – Part 3 (DoS).** Приложение (Missile Launch App) даёт поле для ввода кода. Если ввести очень длинную строку (сотни/тысячи символов), в нативном методе initiateLaunchSequence() может произойти переполнение буфера, приводящее к крашу (DoS). В logcat могут появиться сообщения о stack smashing. Так как разработчики не проверяют длину строки, это типичная уязвимость памяти.

  
*Рисунок 13 – Пример аварийного завершения приложения при вводе огромной строки.*

# **Безопасность хранения и реализации**

По всем перечисленным пунктам видно, что DIVA хранит/обрабатывает данные небезопасно:

* Логи, Shared Preferences, базы данных, файлы на SD-карте.
* Нет фильтрации пользовательского ввода (SQL-инъекция, DoS, XSS).
* Отсутствует или слабо реализован контроль доступа к активностям и Content Provider.

# Выводы и рекомендации

1. Не логировать конфиденциальные данные.
2. Исключить жёстко прописанные ключи (hardcoding) в Java-коде и нативных библиотеках.
3. Применять шифрование для хранения данных в базе/Shared Preferences/файлах.
4. Проверять и экранировать ввод (SQL, URI, длина).
5. Настраивать экспорт активностей и провайдеров (только если нужно), использовать аутентификацию/авторизацию.

Все примеры эксплуатации и извлечения данных наглядно подтверждают, что при анализе памяти (декомпиляции, просмотре .so через strings), а также при использовании adb shell можно получить доступ к конфиденциальным сведениям или выполнить атаки (SQL-инъекция, DoS).