Project Report

14 oct. 2024

Ayush Kumar Thakur

Mentor.- ***Kirankumar S.*** *(CSI)*

CYBER SECURED INDIA (CSI)

Vulnerability Report for DIVA Android App Challenges

A Comprehensive Analysis of Vulnerabilities in the DIVA Android App Challenges

Ayush Kumar Thakur

14-october-2024

This report highlights vulnerabilities in the DIVA Android app challenges, covering insecure logging, hardcoded secrets, insecure data storage, input validation issues, and access control problems. Each issue is rated for severity, impact, steps to reproduce, PoC, remediation suggestions, and CWE identifiers. The report underscores the need for secure coding practices and thorough security testing in Android app development, offering valuable insights for developers and security professionals.

Let's Start 👉 👇

Challenge 1: Insecure Logging.

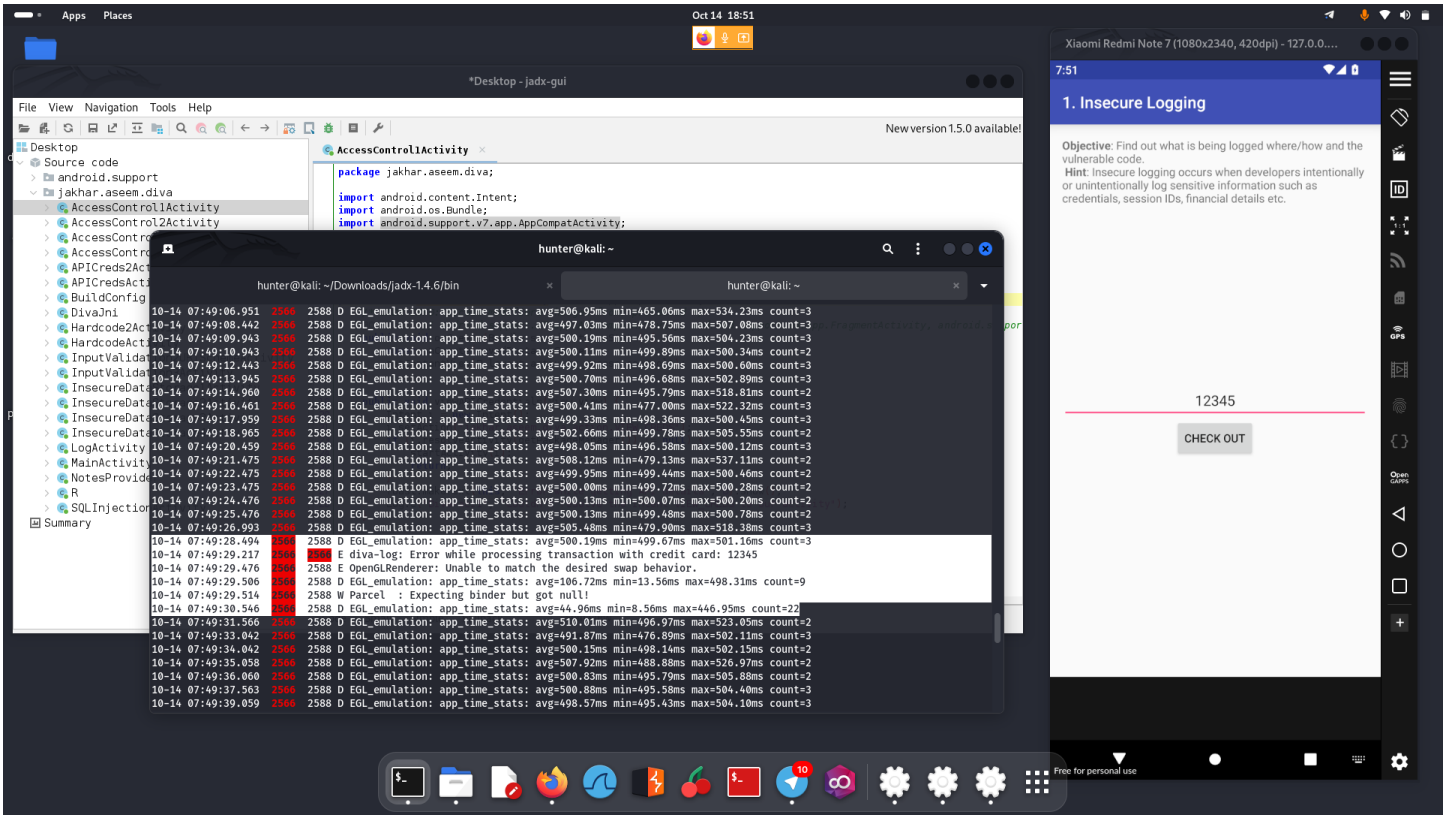
- **Title & Severity:**
    - **Title: Insecure Logging**
    - **Severity: High**

- **Description:** The application logs sensitive data, such as credit card information, without proper protection. This data is logged in plain text.

- **Impact:** The impact of this vulnerability is significant. It exposes users' credit card information to potential attackers and could lead to unauthorized access to sensitive data.

- **Steps to Reproduce:**
    1. Enter data into the application and trigger the "check out" action.
    2. Observe the logcat output for logged data.

- **PoC (Proof of Concept):**



E/diva-log(6067): Error while processing transaction with credit card: 4711

- **Remediation:**
    - Avoid logging sensitive data.
    - Implement proper access controls and encryption for log files.
    - Educate developers on secure logging practices.

- **CWE (Common Weakness Enumeration):**
  - CWE-532: Inclusion of Sensitive Information in Log Files

Challenge 2: Hardcoding Issues - Part 1

- **Title & Severity:**
  - **Title:** Hardcoding Issues - Part 1
  - **Severity:** Medium

- **Description:** The application contains a hardcoded secret key, "**vendorsecretkey**," which provides access to restricted areas.

- **Impact:** The impact is moderate. Attackers who discover this hardcoded key can gain unauthorized access to restricted parts of the application.

- **Steps to Reproduce:**

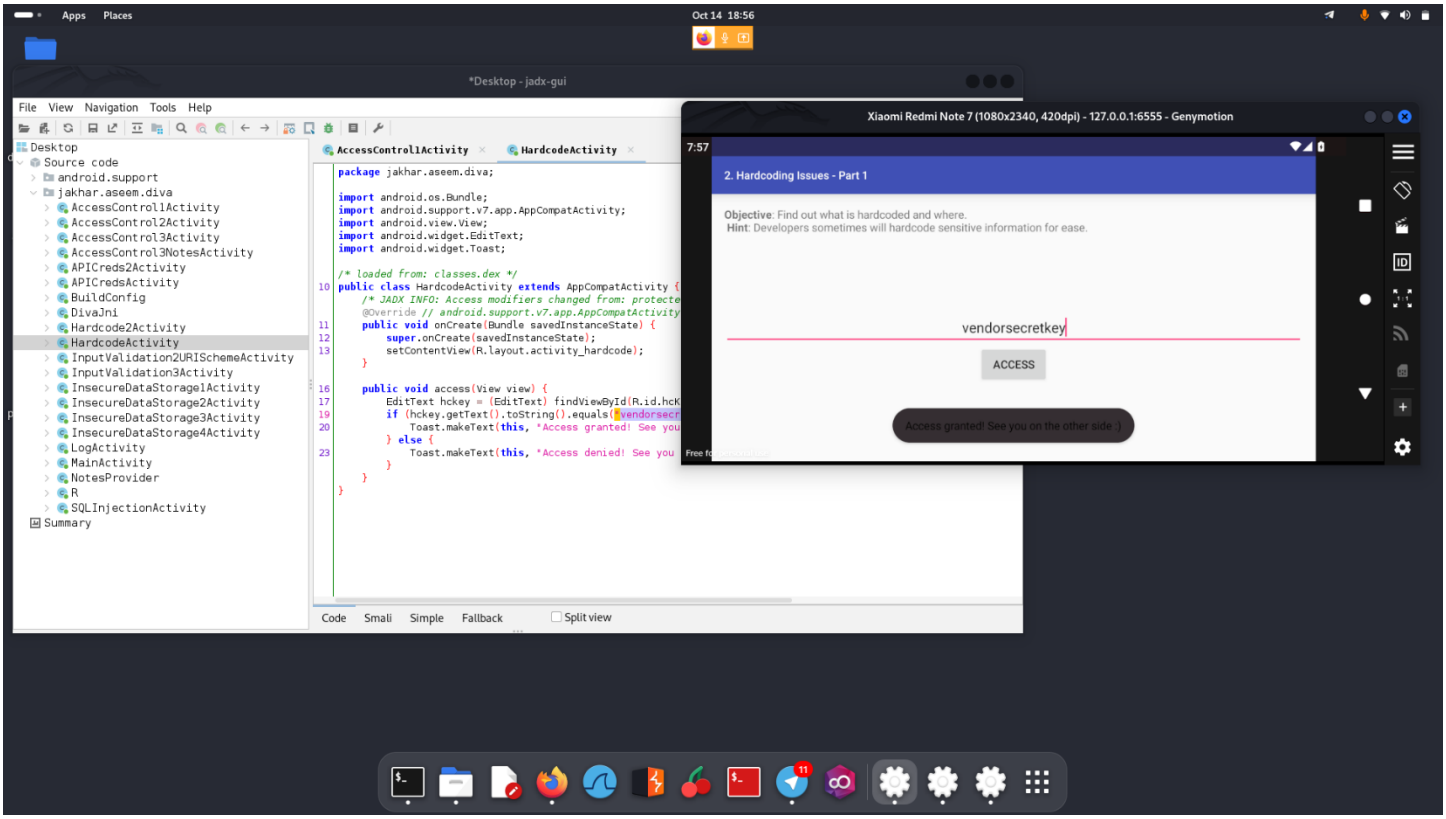
1. Examine the source code of the application.
2. Identify the presence of the "vendorsecretkey."
3. Use the key to access restricted areas.

- **PoC (Proof of Concept):**

```
if (((EditText)this.findViewById(2131492987)).getText().toString().equals("vendorsecretkey")) {

   Toast.makeText((Context)this, (CharSequence)"Access granted! See you on the other side :)",
(int)0).show();     return;

}
```



- **Remediation:**
    - Avoid hardcoding sensitive information like secret keys in the source code.
    - Use secure storage mechanisms or environment variables for such data.
    - Implement proper access controls.

- **CWE (Common Weakness Enumeration):**
    - CWE-798: Use of Hard-coded Credentials

Challenge 3: Insecure Data Storage - Part 1

- **Title & Severity:**
    - **Title:** Insecure Data Storage - Part 1
    - **Severity:** Medium

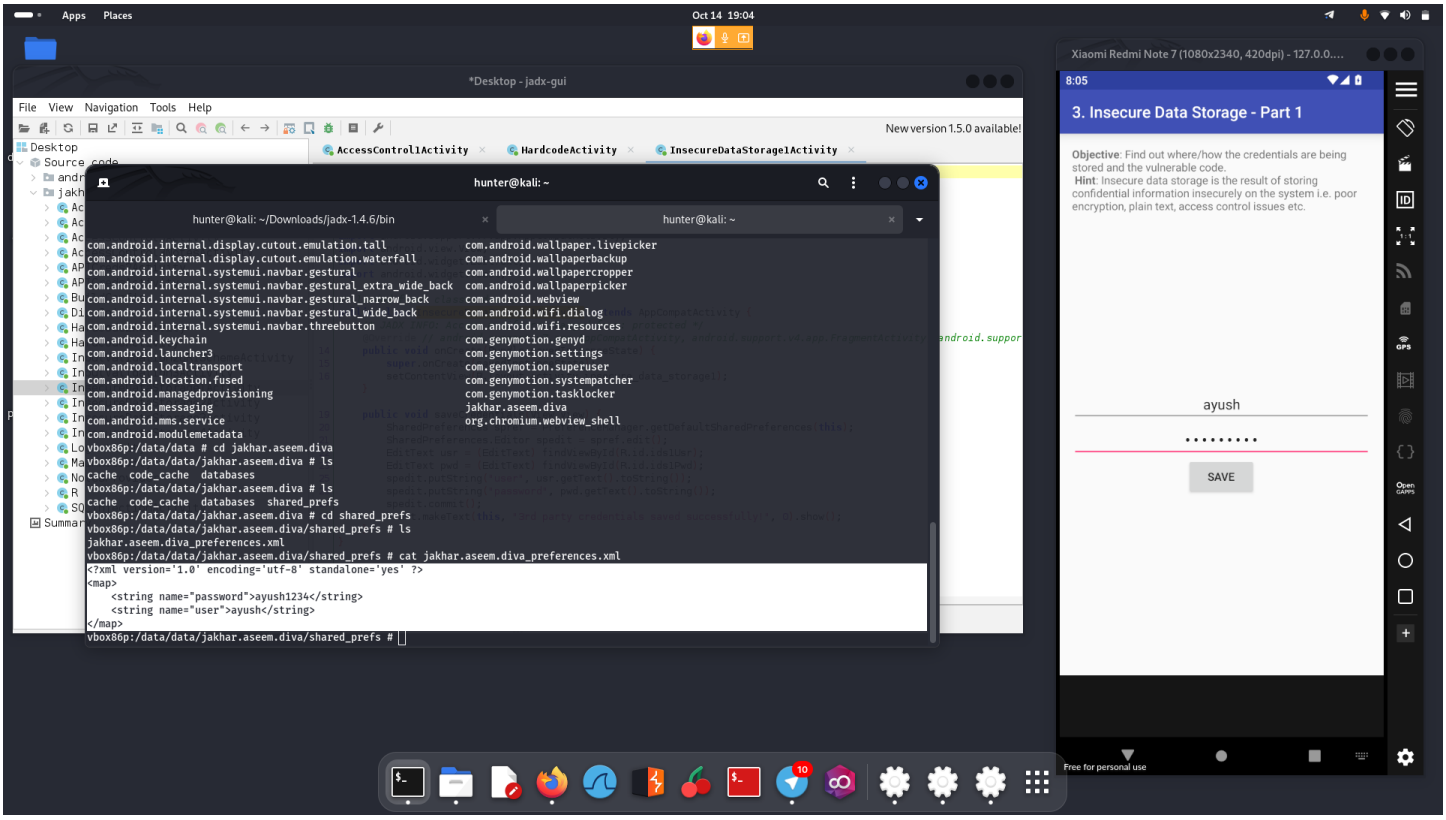
• **Description:** The application stores user credentials in plain text in the shared preferences folder.

• **Impact:** The impact is moderate. It exposes user credentials to potential attackers who have access to the device, risking unauthorized access.

• **Steps to Reproduce:**

1. Enter credentials into the application.
2. Review the source code and discover the use of shared preferences.
3. Use ADB shell to access the shared preferences folder.

• **PoC (Proof of Concept):**



• Using ADB shell to access shared preferences:

```
Bash adb shell cd

/data/data/jakhar.aseem.diva/shared_prefs
```

- **Remediation:**
    - Avoid storing sensitive data in plain text in shared preferences.
    - Implement encryption or hashing for stored credentials.
    - Educate developers on secure data storage practices.

- **CWE (Common Weakness Enumeration):**
    - CWE-312: Cleartext Storage of Sensitive Information

Challenge 4: Insecure Data Storage - Part 2

- **Title & Severity:**
    - **Title:** Insecure Data Storage - Part 2
    - **Severity:** Medium

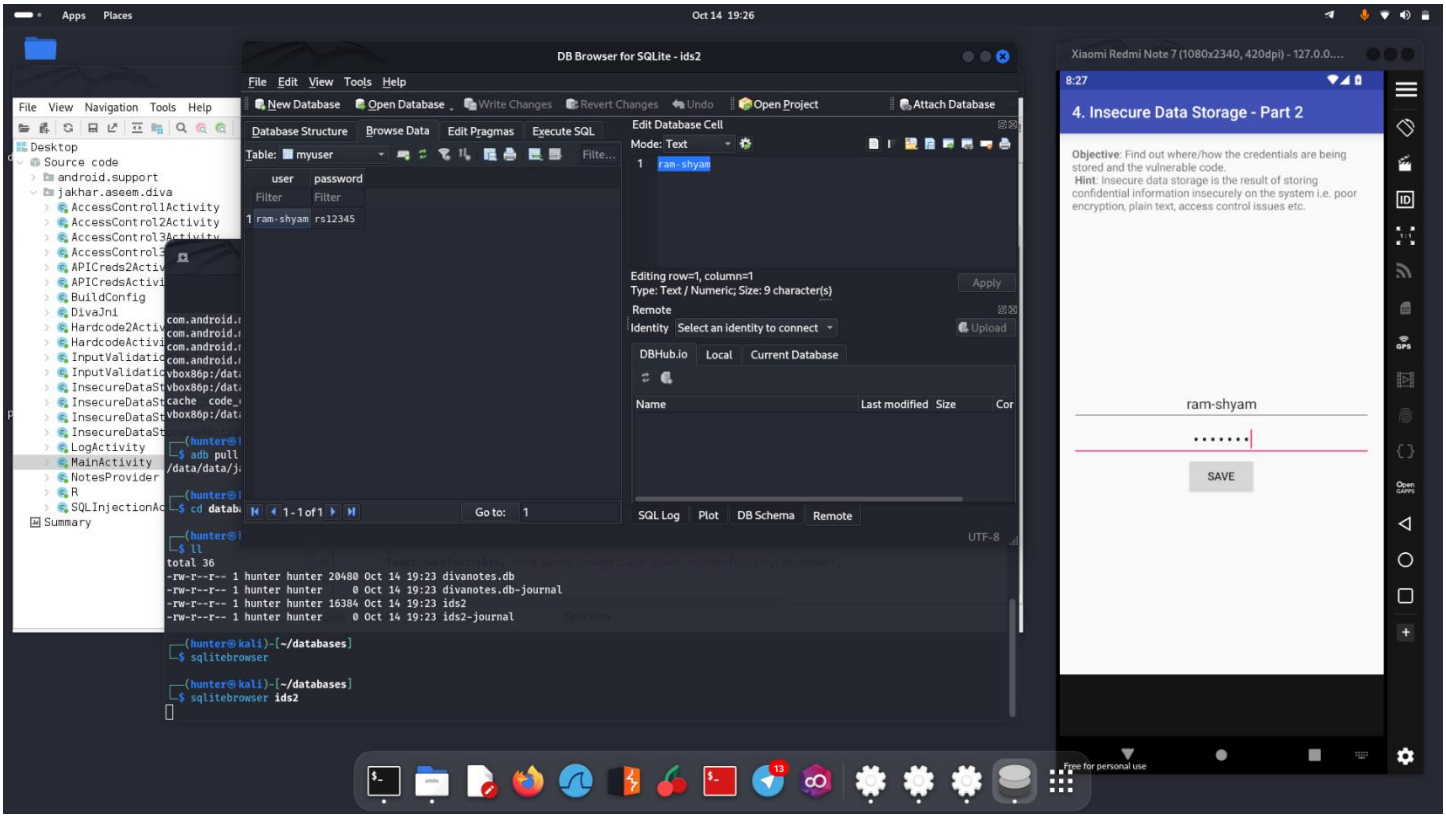
- **Description:** The application stores user credentials in a SQLite database without encryption.

- **Impact:** The impact is moderate. It exposes user credentials to potential attackers with access to the device, risking unauthorized access.

- **Steps to Reproduce:**
    1. Enter credentials into the application.
    2. Review the source code and discover the use of an SQLite database.
    3. Use ADB shell to access and extract the database file.

- **PoC (Proof of Concept):**
    - **Using ADB shell to pull the database**:

```
adb pull /data/data/jakhar.aseem.diva/databases/ids .
```



- **Remediation:**
    - Implement encryption or hashing for stored credentials in the database.
    - Avoid storing sensitive data in plain text in databases.
    - Educate developers on secure data storage practices.

- **CWE (Common Weakness Enumeration):**
    - CWE-313: Cleartext Storage in a File or on Disk

Challenge 5: Insecure Data Storage - Part 3

- **Title & Severity:**

- **Title:** Insecure Data Storage - Part 3
- **Severity:** Medium

- **Description:** The application stores user credentials in a temporary file without proper protection.

- **Impact:** The impact is moderate. It exposes user credentials to potential attackers with access to the device, risking unauthorized access.

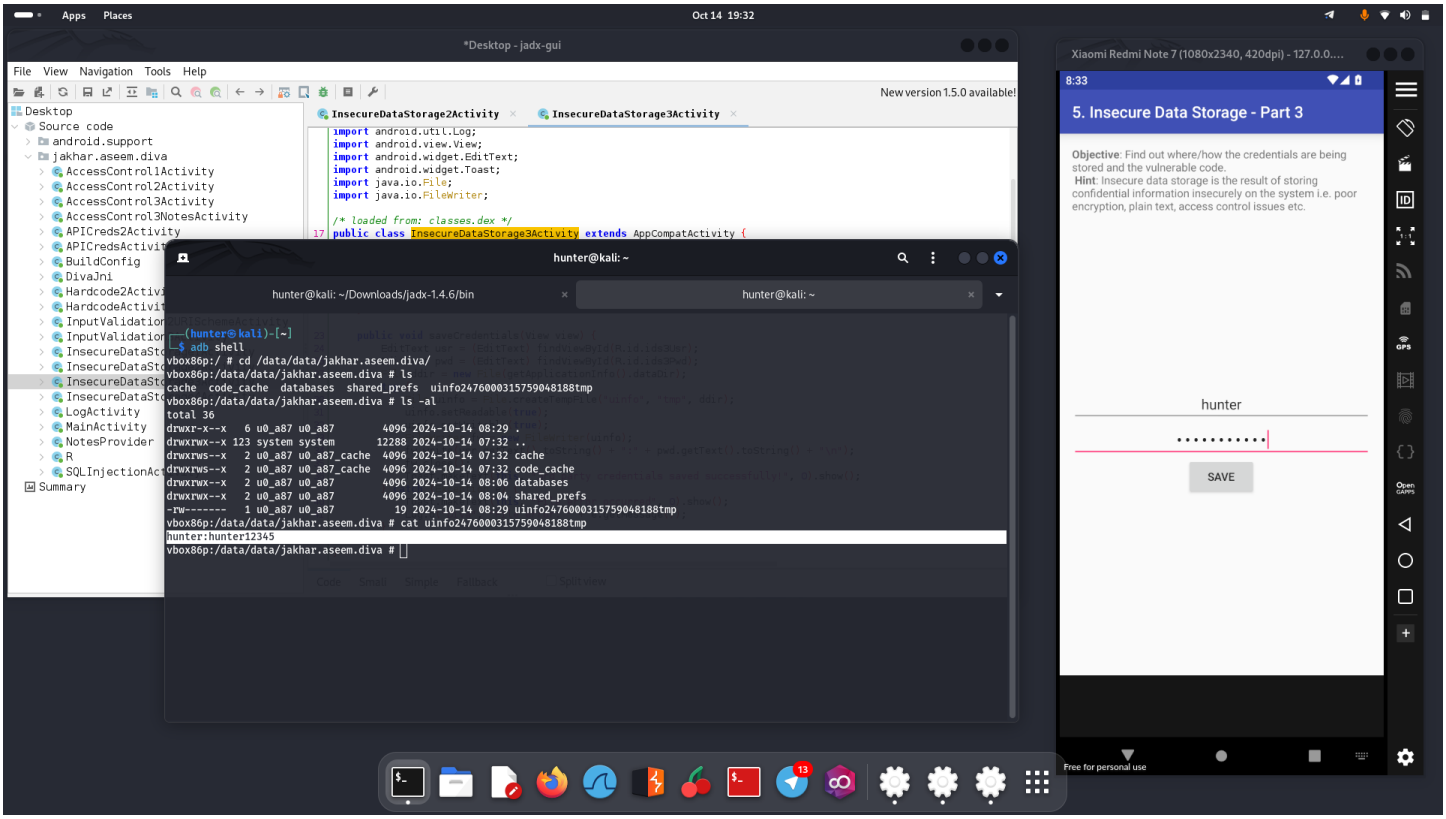
- **Steps to Reproduce:**

1. Enter credentials into the application.
2. Review the source code and discover the use of temporary file creation.
3. Use ADB shell to locate and access the temporary file.

- **PoC (Proof of Concept):**

  - **Using ADB shell to access the temporary file:**

bash adb shell cat /data/data/jakhar.aseem.diva/app_uinfo-838734269tmp



- **Remediation:**

- Avoid storing sensitive data in plain text in temporary files.
- Implement encryption or hashing for stored credentials.
- Educate developers on secure data storage practices.

- **CWE (Common Weakness Enumeration):**
  - CWE-314: Cleartext Storage of Sensitive Information in a File

Challenge 6: Insecure Data Storage - Part 4

- **Title & Severity:**
  - **Title:** Insecure Data Storage - Part 4
  - **Severity:** Medium

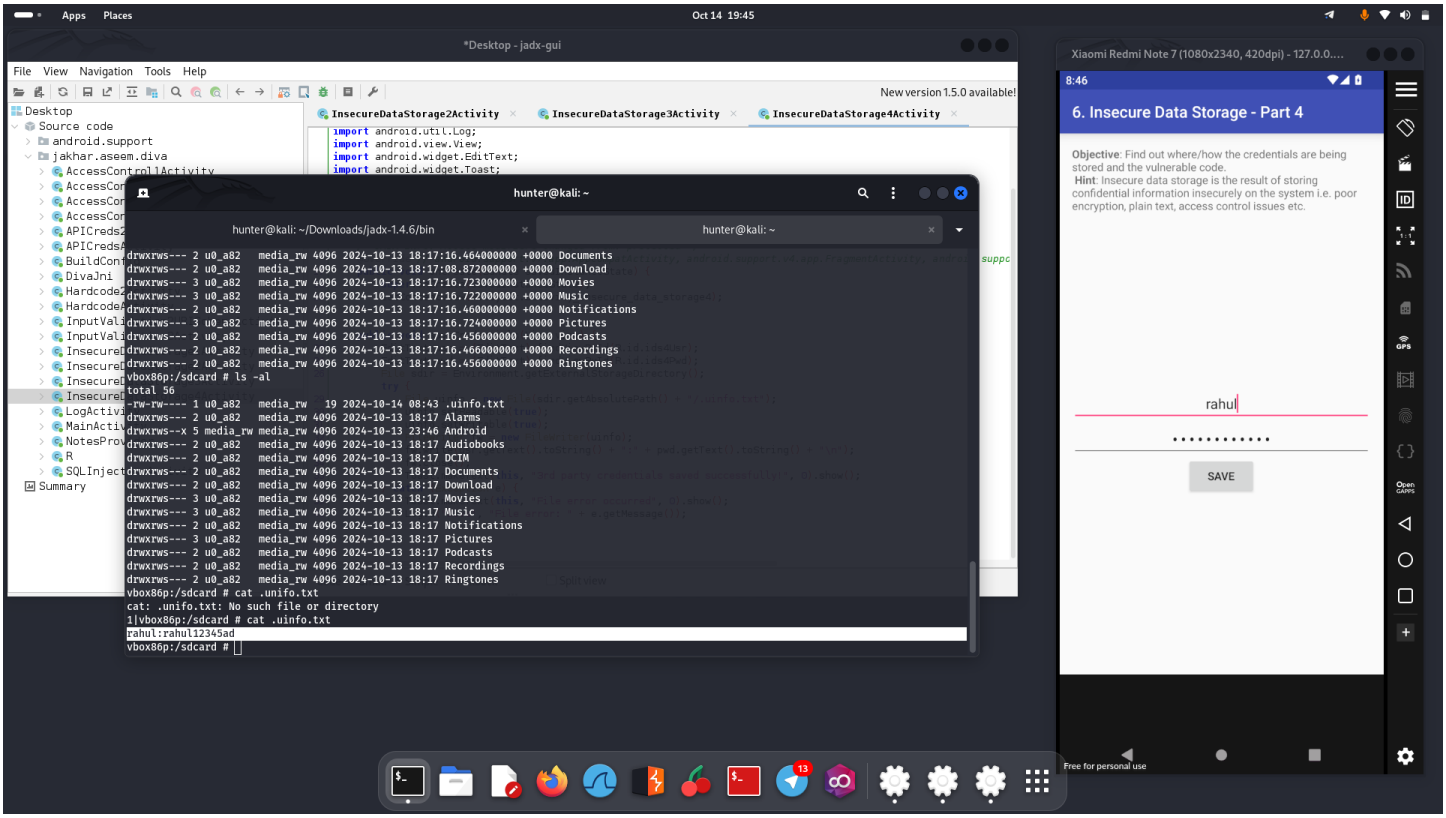
- **Description:** The application stores user credentials in a local file on the SD card without proper protection.

- **Impact:** The impact is moderate. It exposes user credentials to potential attackers with access to the device's SD card, risking unauthorized access.

- **Steps to Reproduce:**
  1. Enter credentials into the application.
  2. Review the source code and discover the use of an SD card file.
  3. Use ADB shell to locate and access the file on the SD card.

- **PoC (Proof of Concept):**
  - **Using ADB shell to access the file on the SD card:**

```
bash adb shell cat /sdcard/.uinfo.txt
```



- **Remediation:**
  - Avoid storing sensitive data in plain text on the SD card.
  - Implement encryption or hashing for stored credentials.
  - Educate developers on secure data storage practices.

- **CWE (Common Weakness Enumeration):**
  - CWE-313: Cleartext Storage in a File or on Disk

Challenge 7: Input Validation Issues - Part 1

- **Title & Severity:**
  - **Title:** Input Validation Issues - Part 1
  - **Severity:** Medium

• **Description:** The application does not properly validate user input when searching for users in the database, potentially exposing sensitive data.

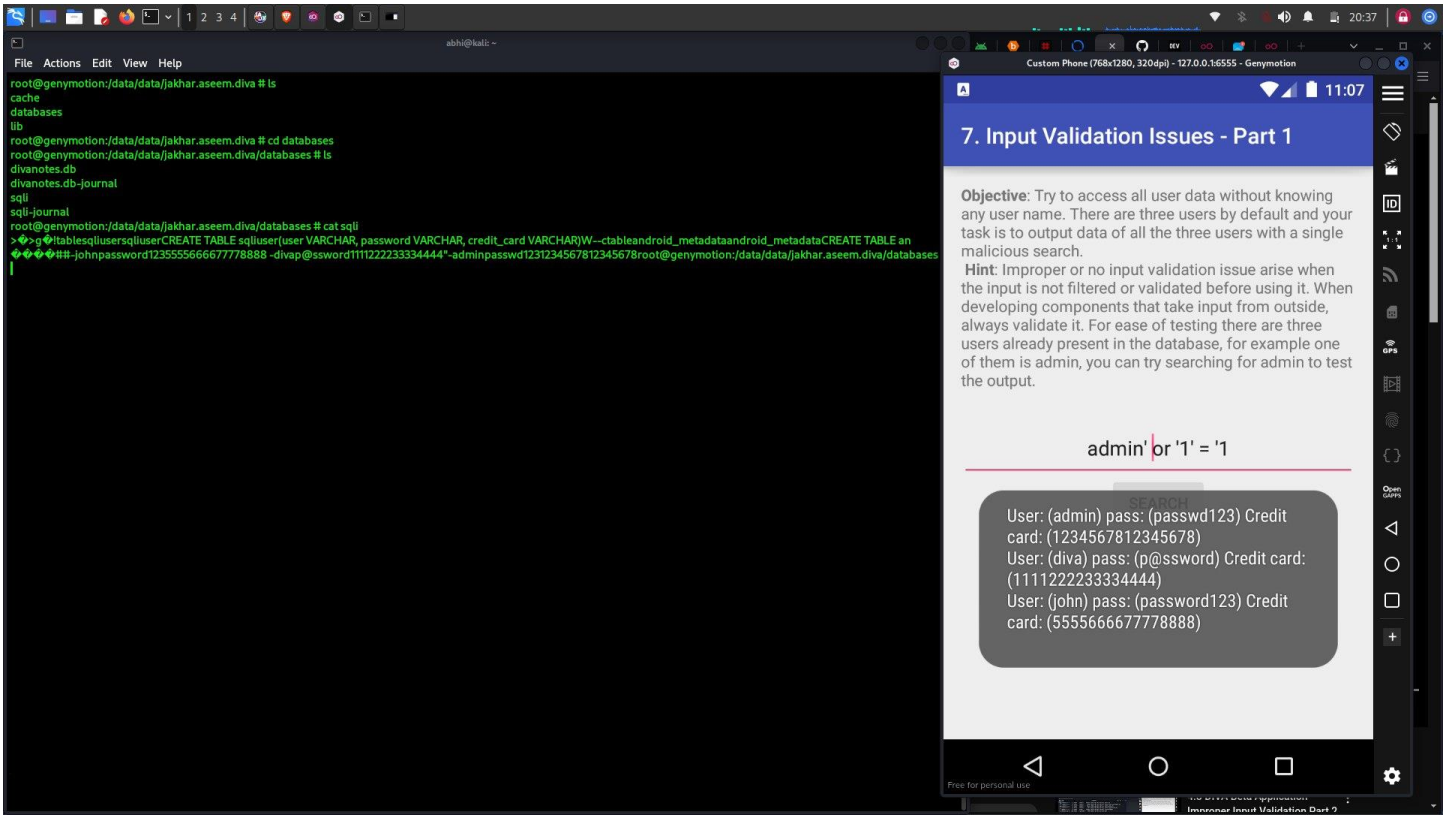
• **Impact:** The impact is moderate. An attacker can exploit this vulnerability to retrieve user data without proper authorization.

• Steps to Reproduce:

1. Enter a malicious input (e.g., a single quote) into the search field.
2. Observe the error message or the database query in logcat.

• PoC (Proof of Concept):

Using a single quote as input: 1' or '1' != '2



• Remediation:

- Implement proper input validation and sanitization to prevent SQL injection.
- Avoid executing user input as part of database queries.
- Educate developers on secure coding practices.

- **CWE (Common Weakness Enumeration):**
    - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Challenge 8: Input Validation Issues - Part 2

- **Title & Severity:**
    - **Title:** Input Validation Issues - Part 2
    - **Severity:** Medium

- **Description:** The application allows users to access sensitive information by entering a specific URL into the search field.

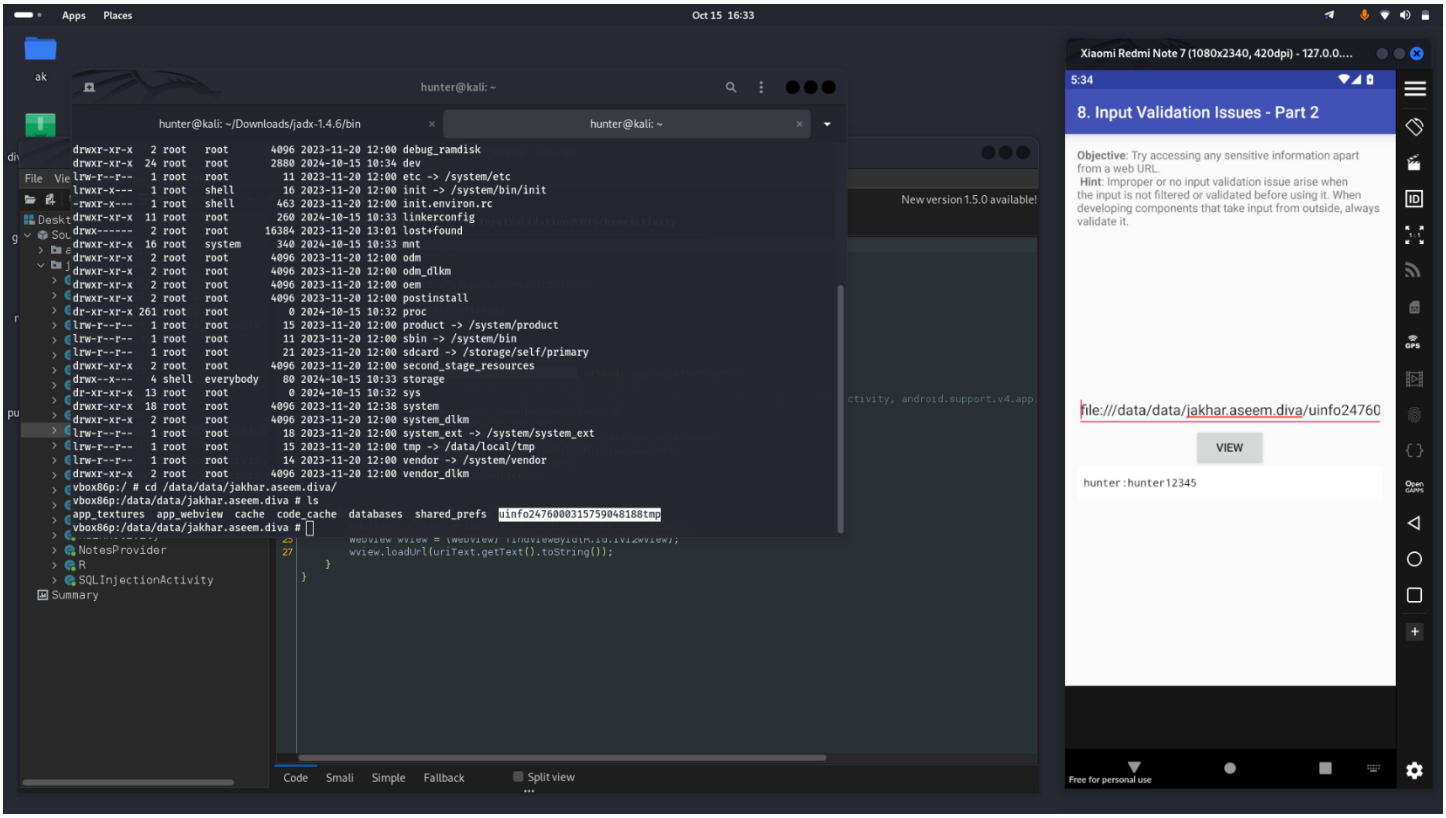
- **Impact:** The impact is moderate. It enables users to access potentially confidential data via URL manipulation.

- **Steps to Reproduce:**

1. Enter a specific URL **(e.g., "file:///sdcard/.uinfo.txt")** into the search field.
2. Observe the retrieval of sensitive information.

- **PoC (Proof of Concept):**

using the URL "file:///sdcard/.uinfo.txt" to access the file.



- **Remediation:**
  - Implement proper input validation to prevent unauthorized access to sensitive data via URL manipulation.
  - Educate developers on secure coding practices.

- **CWE (Common Weakness Enumeration):**
  - CWE-264: Permissions, Privileges, and Access Controls

Challenge 9: Access Control Issues - Part 1

- **Title & Severity:**
  - **Title:** Access Control Issues - Part 1
  - **Severity:** High

- **Description:** The application allows unauthorized access to a specific activity without proper authentication or access control.

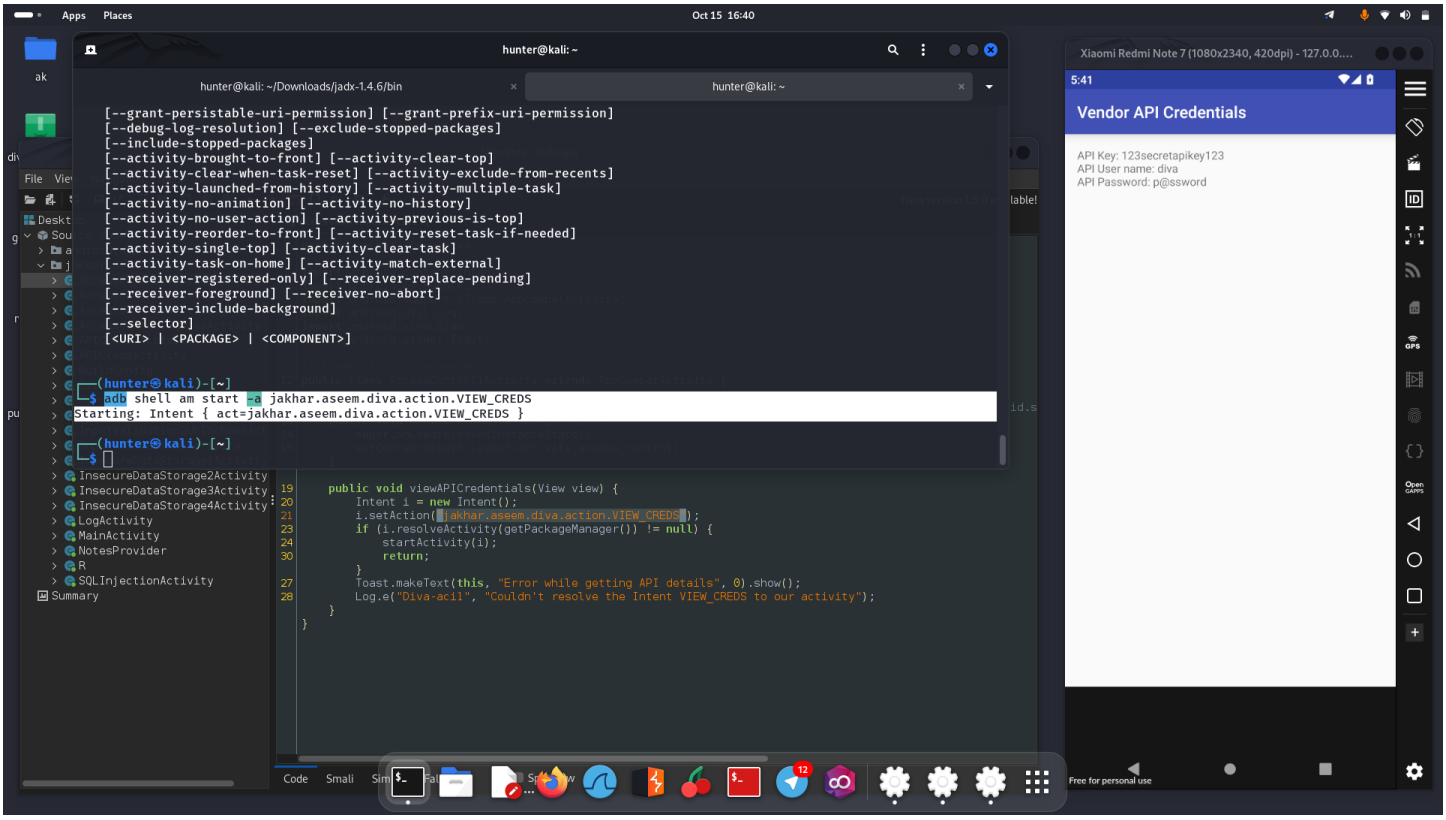
- **Impact:** The impact is high. Attackers can gain unauthorized access to a specific activity without navigating through the normal user interface.

- **Steps to Reproduce:**

1. Discover the activity name ("jakhar.aseem.diva.APICredsActivity").
2. Use ADB shell or other means to directly start the activity.

- **PoC (Proof of Concept):**

arduino run app.activity.start --component jakhar.aseem.diva jakhar.aseem.diva.APICredsActivity



- **Remediation:**
    - Implement proper access controls and authentication mechanisms for activities.
    - Ensure that sensitive functionalities are not accessible without proper authorization.
    - Educate developers on secure coding practices.

- **CWE (Common Weakness Enumeration):**
    - CWE-285: Improper Access Control

Challenge 10: Access Control Issues - Part 2

- **Title & Severity:**
    - **Title:** Access Control Issues - Part 2
    - **Severity:** High

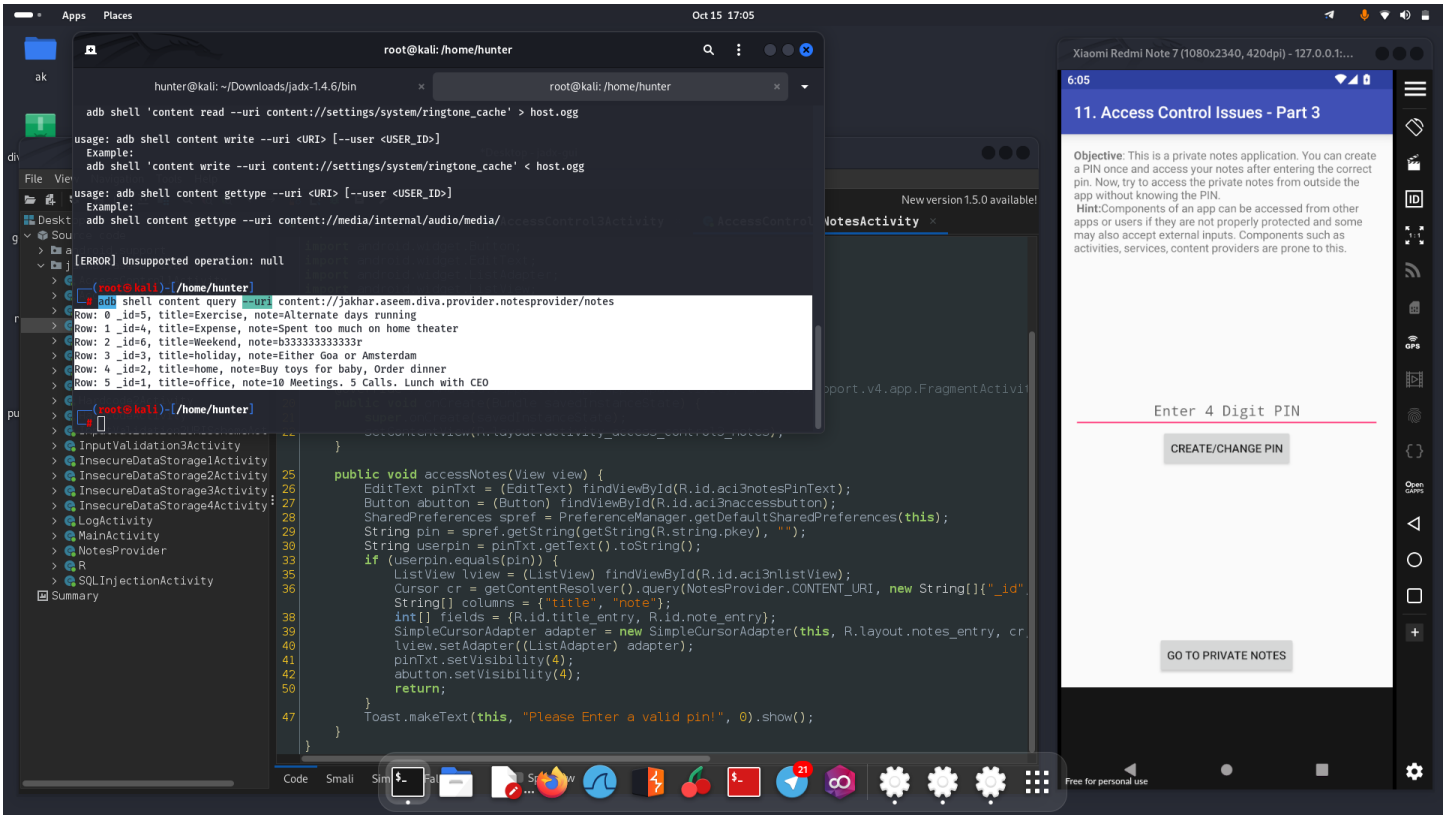
- **Description:** The application allows unauthorized access to another specific activity with a PIN-based protection without validating the PIN.

- **Impact:** The impact is high. Attackers can gain unauthorized access to a specific activity by bypassing the PIN protection.

- **Steps to Reproduce:**
    1. Discover the activity name **("jakhar.aseem.diva.APICreds2Activity").**
    2. Observe that the PIN protection is not properly validated.
    3. Use an intent to directly start the activity without a valid PIN.

- **PoC (Proof of Concept):**

arduino run app.activity.start --component jakhar.aseem.diva jakhar.aseem.diva.APICreds2Activity --extra boolean check_pin false



- **Remediation:**

  - Implement proper PIN validation for activities that require it.
  - Ensure that sensitive functionalities are protected by strong access controls.
  - Educate developers on secure coding practices.

- **CWE (Common Weakness Enumeration):**

  - **C**WE-306: Missing Authentication for Critical Function

Challenge 11: Access Control Issues - Part 3

- **Title & Severity:**
  - **Title:** Access Control Issues - Part 3
  - **Severity:** High

- **Description:** The application allows unauthorized access to a notes storage without requiring the PIN.

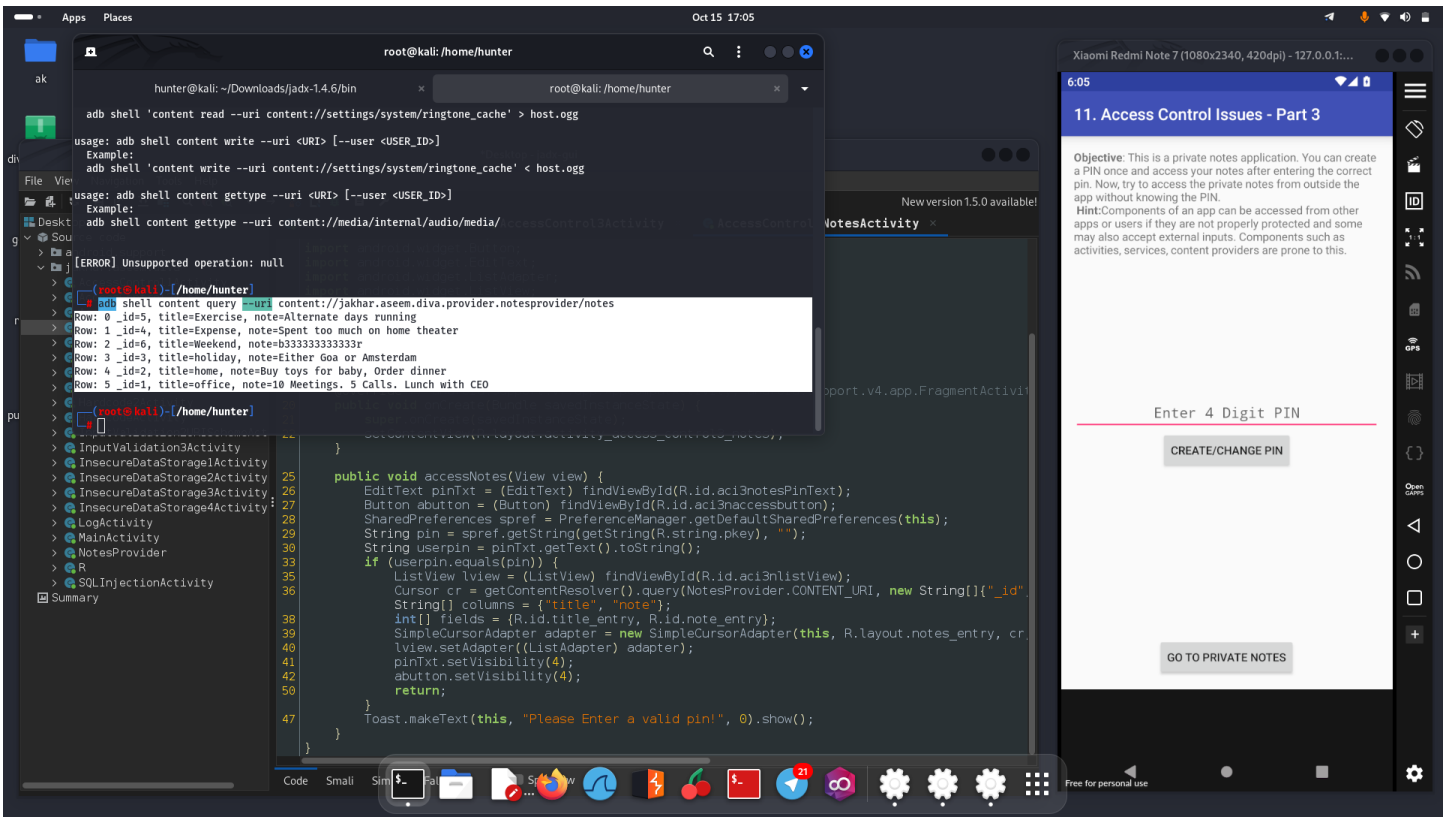
- **Impact:** The impact is high. Attackers can gain unauthorized access to sensitive notes without requiring the PIN.

- **Steps to Reproduce:**

1. Discover the exported content provider **("content://jakhar.aseem.diva.provider.notesprovider/notes/").**
2. Observe that the content provider does not enforce the PIN requirement.
3. Query the content provider to retrieve the notes.

- **PoC (Proof of Concept):**

arduino run app.provider.query content://jakhar.aseem.diva.provider.notesprovider/notes/ --projection "* FROM notes;--"



- **Remediation:**
    - Ensure that access to sensitive data is properly controlled and authenticated.
    - Implement proper access controls and validation for content providers.
    - Educate developers on secure coding practices

- **CWE (Common Weakness Enumeration):**
    - CWE-284: Improper Access Control

Challenge 12: Hardcoding Issues - Part 2

- **Title & Severity:**
    - **Title:** Hardcoding Issues - Part 2
    - **Severity:** Medium

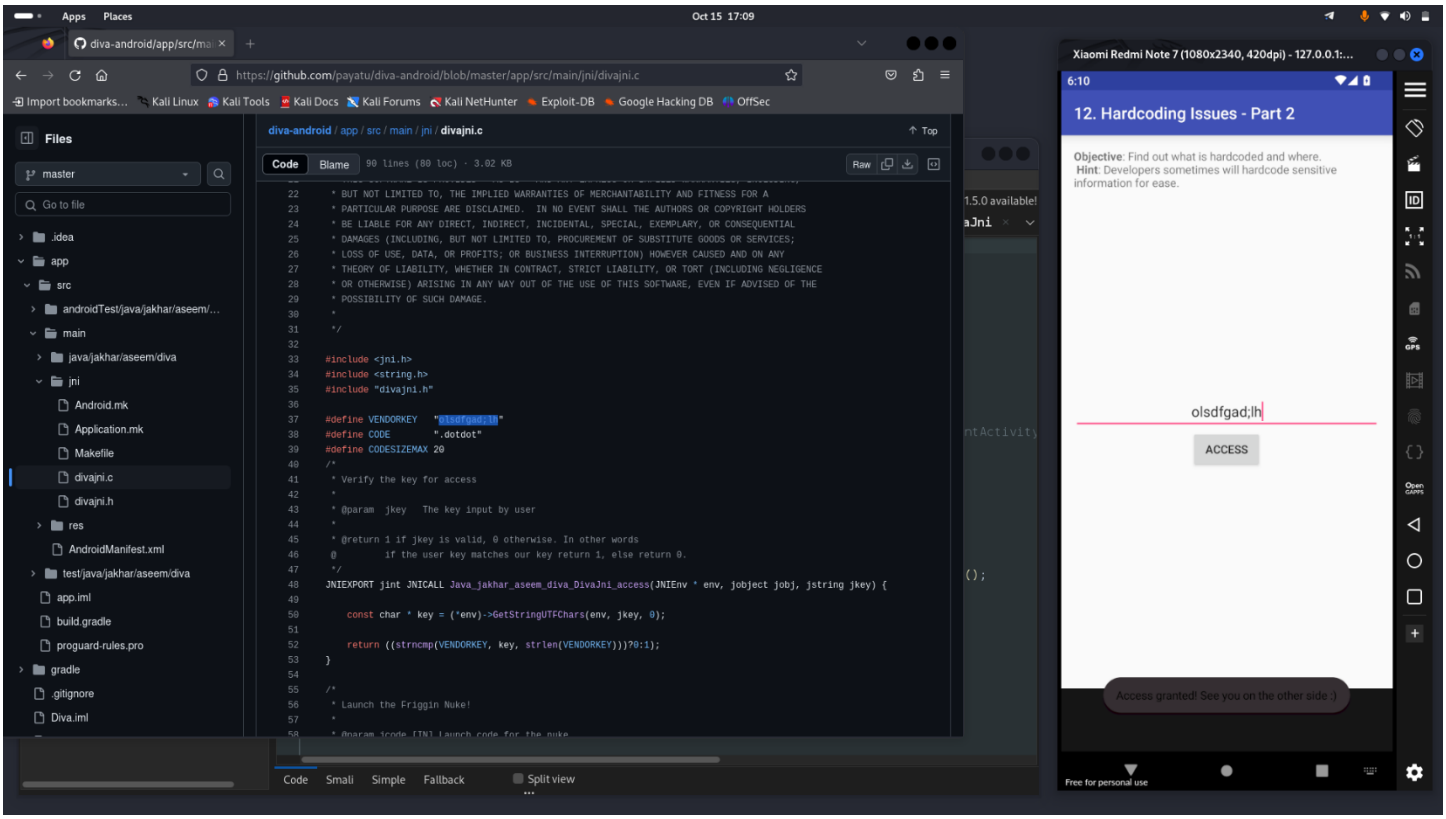
- **Description:** The application loads a native library that contains a suspicious string *"olsdfgad;lh."*

- **Impact:** The impact is moderate. Attackers who discover this hardcoded string may have access to sensitive information or functionality.

- **Steps to Reproduce:**
    1. Examine the native library used by the application.
    2. Discover the suspicious hardcoded string.

- **PoC (Proof of Concept):**

Use a strings command to examine the native library for the hardcoded string.



- **Remediation:**

- Avoid hardcoding sensitive information in native libraries.
- Use secure storage mechanisms or environment variables for such data.
- Implement proper access controls and authentication.

- **CWE (Common Weakness Enumeration):**

- CWE-798: Use of Hard-coded Credentials

Challenge 13: Input Validation Issues - Part 3

- **Title & Severity:**
  - **Title:** Input Validation Issues - Part 3
  - **Severity:** Medium

- **Description:** The application crashes when it receives a very large input string, indicating inadequate input validation.

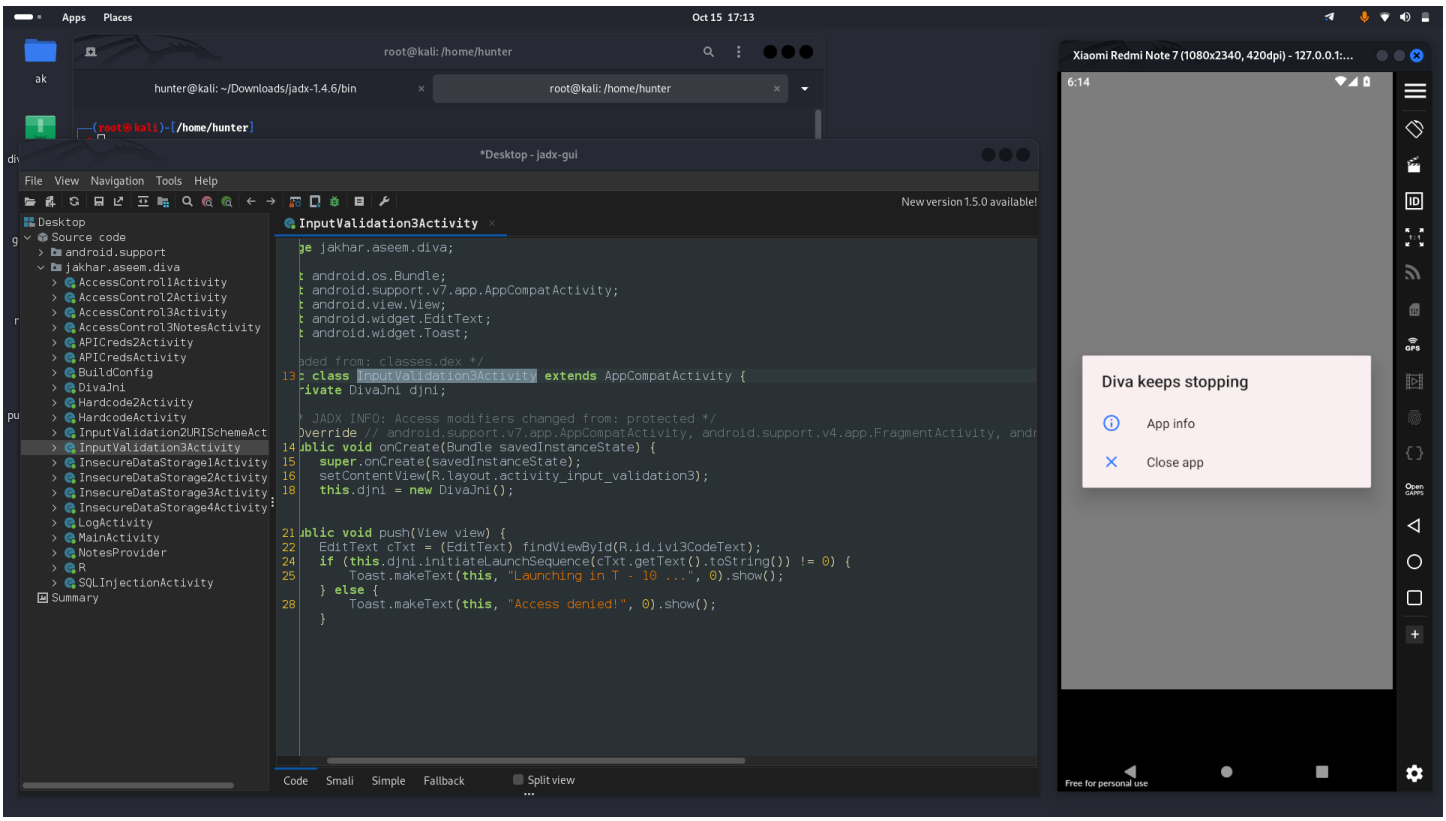
- **Impact:** The impact is moderate. Attackers can disrupt the application's functionality by sending oversized input strings.

- **Steps to Reproduce:**

1. Enter a very large input string (e.g., 30 times "A") into the application.

- **PoC (Proof of Concept):**

Using a very large input string to crash the application.



- **Remediation:**

    - Implement proper input validation and error handling to prevent application crashes.
    - Set input size limits to prevent buffer overflows.
    - Educate developers on secure coding practice.

- **CWE (Common Weakness Enumeration):**

    - CWE-190: Integer Overflow or Wraparound

This report provide an overview of each challenge, its impact, steps to reproduce, proof of concept, remediation recommendations, and the relevant Common Weakness Enumeration (CWE) identifiers.