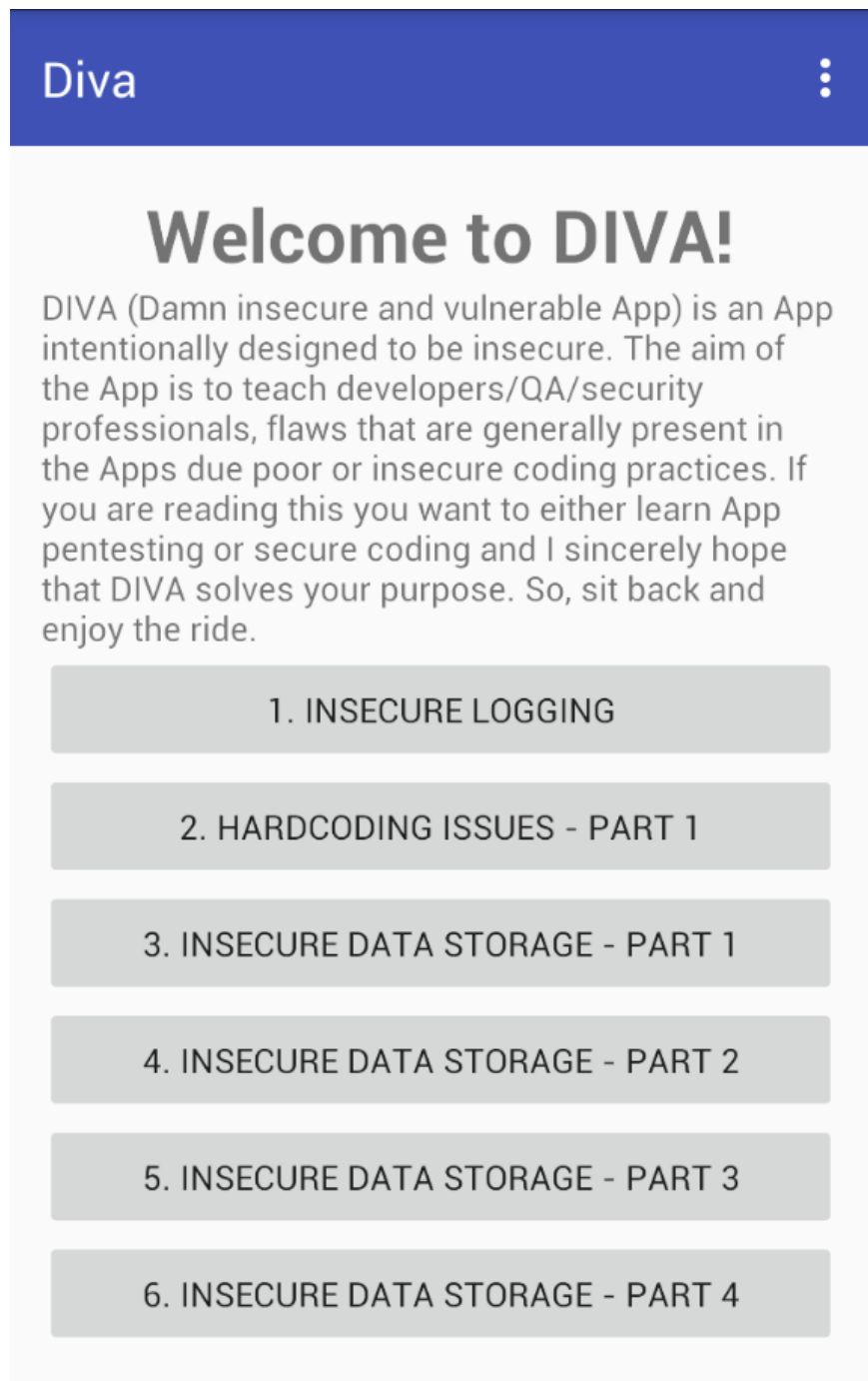


# DIVA android Solution

By Praful Mardhekar / sudiXO17

---



# Insecure Logging

## Description:

- The application logs sensitive information, specifically credit card details, during transaction processing. This sensitive data can be accessed through the logcat, potentially exposing users' financial information.

## Impact:

- Exposing sensitive information in logs can lead to unauthorized access and misuse of users' financial data. Attackers could exploit this vulnerability to conduct fraudulent transactions or identity theft, resulting in significant financial loss for users and reputational damage for the organization.

## Steps to Reproduce:

1. Launch the DIVA application on the virtual device.
2. Initiate a transaction using a valid credit card number (e.g., 123456789).
3. Observe the logcat output using the command:
4. `adb logcat | grep 123456789`
5. Note the sensitive credit card number logged in the output.

## PoC (Proof of Concept):

1. The log entry captured indicates that the application logs credit card numbers in plaintext:

```
$ adb logcat | grep 123456789
10-07 06:35:32.452 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:35:45.603 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:35:46.941 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:35:47.413 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:36:38.372 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:36:39.301 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:36:39.761 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:36:40.181 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:36:40.645 2482 2482 E diva-log: Error while processing transaction with credit card: 123456789
10-07 06:41:48.661 3194 3194 E diva-log: Error while processing transaction with credit card: 123456789
```

- 2.
3. This demonstrates that sensitive information can be accessed through logcat, illustrating the risk of data exposure.

## Remediation:

- Sanitizing sensitive data: Avoid logging sensitive information such as credit card numbers, personal identification numbers, or other sensitive user data.
- Using log levels appropriately: Use log levels to restrict sensitive information to only critical errors or warnings that do not disclose sensitive details.
- Employing data masking: When logging necessary information, mask sensitive parts (e.g., logging only the last four digits of credit card numbers).

## CWE (Common Weakness Enumeration):

- CWE-532: Information Exposure Through Log Files - This weakness occurs when an application logs sensitive information in a way that allows unauthorized users to access it.

# Hardcoding Issues – Part 1

## Description:

Hardcoding sensitive information such as API keys, usernames, passwords, or encryption keys directly into the source code of an application is a common security flaw. When hardcoded, this information can be easily extracted by attackers using reverse engineering techniques. This is particularly dangerous in mobile applications where the APK can be decompiled, and hardcoded credentials can be exposed.

## Impact:

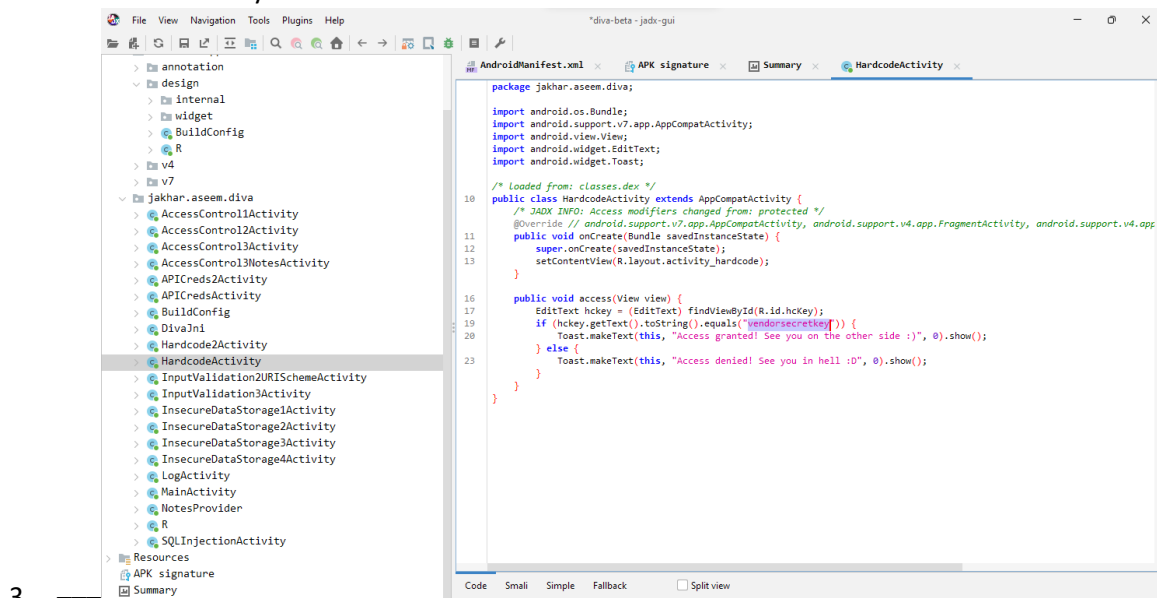
1. Access backend systems and APIs.
2. Bypass authentication mechanisms.
3. Perform unauthorized actions like retrieving sensitive user data.
4. Compromise the integrity of the entire application and its ecosystem.

## Steps to Reproduce:

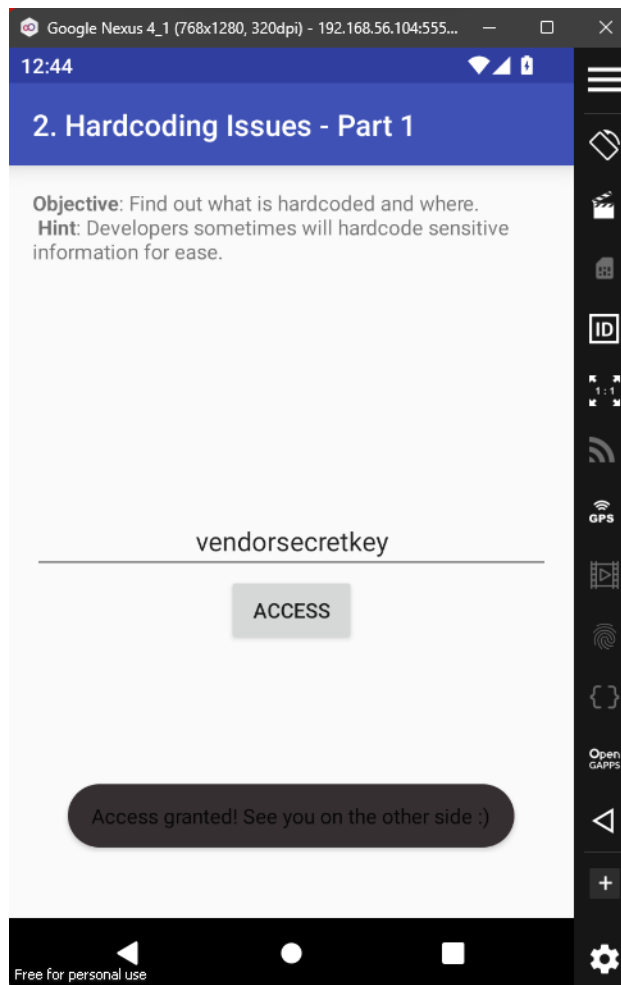
1. Decompile the APK: Use a decompiling tool such as apktool or Jadx to reverse engineer the APK file.
2. Analyze the Decompiled Code: Search for hardcoded strings like API keys, tokens, passwords, or other sensitive credentials in the source code, especially in the configuration or connection handling classes.
3. Extract the Hardcoded Information: Identify and extract sensitive information from the decompiled source files.

## PoC (Proof of Concept):

1. Use JADX (gui version) to decompile the APK.
2. To see what happens in the corresponding section go to Source code > jakhar.aseem.diva. > HardCodeActivity



3. \_\_\_\_\_
4. You can infer password in code: vendorsecretkey



5.

## Remediation:

- Remove Hardcoding: Do not hardcode sensitive information in the source code.
- Use Secure Storage: Store credentials securely using the Keystore API for Android or any secure storage method provided by the platform.
- Environment Variables: Use environment variables or encrypted configuration files that are securely accessed at runtime.
- Obfuscation: Use ProGuard or another obfuscation tool to make reverse engineering more difficult, although this is not a replacement for secure storage.

## CWE (Common Weakness Enumeration):

- CWE-798: Use of Hard-coded Credentials - This Common Weakness Enumeration (CWE) identifies the flaw of using hardcoded credentials, leading to a significant security vulnerability when the code is compromised.

# Hardcoding Issues – Part 2

## Description:

Hardcoding sensitive information, such as vendor codes or API secrets, directly into an application's codebase remains a significant security flaw. This practice exposes critical information when an attacker decompiles the APK, as the embedded data can be easily extracted using reverse engineering techniques. In the case of DIVA, the divajni native library contains hardcoded secrets that can be leveraged for unauthorized access to the application's functionalities.

## Impact:

Attackers can exploit hardcoded vendor keys to access backend systems and APIs.

Bypass security mechanisms, leading to unauthorized actions.

Retrieve sensitive user information, potentially causing data breaches.

Compromise the application's integrity and trustworthiness, affecting both users and the organization's reputation.

## Steps to Reproduce:

Extract the APK of the DIVA application.

Decompile the APK using tools like JADX.

Analyze the decompiled source code, native library.

Identify the hardcoded vendor key in the code.

Use the key to bypass authentication or access restricted features in the app.

## PoC (Proof of Concept):

1. Extract the APK file: unzip diva-beta.apk

```
(sudixo@vbox)-[~/Downloads]
$ ls
AndroidManifest.xml  classes.dex  lib  resources.arsc
META-INF            diva-android  methods.json
apktool_2.10.0.jar  diva-beta.apk  res
```

3. Search through the source files or native libraries (libdivajni) to find hardcoded credentials.

```
(sudixo@vbox)~[~/Downloads]
$ cd lib

(sudixo@vbox)~[~/Downloads/lib]
$ ls *
arm64-v8a:
libdivajni.so

armeabi:
libdivajni.so

armeabi-v7a:
libdivajni.so

mips:
libdivajni.so

mips64:
libdivajni.so

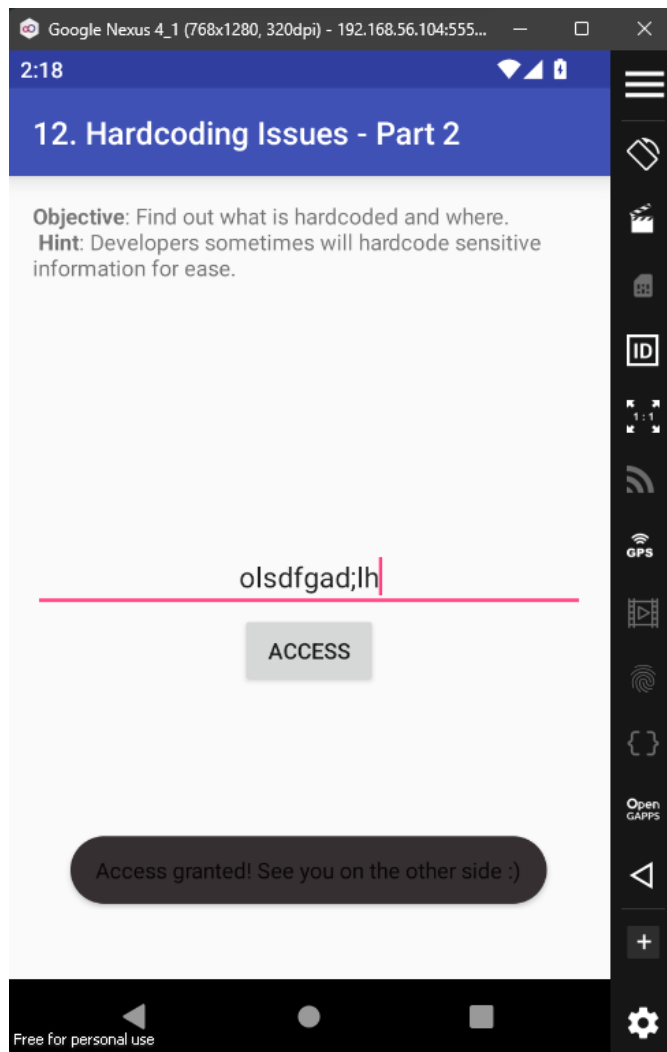
x86:
libdivajni.so

x86_64:
libdivajni.so
```

- 4.
5. open any of the library and analyze it

```
(sudixo@vbox)~[~/Downloads/lib/arm64-v8a]
$ strings libdivajni.so
__cxa_finalize
__cxa_atexit
Java_jakhar_aseem_diva_DivaJni_access
strncmp
Java_jakhar_aseem_diva_DivaJni_initiateLaunchSequence
strcpy
JNI_OnLoad
libstdc++.so
libm.so
libc.so
libdl.so
__edata
__bss_start
__bss_start__
__bss_end__
__end__
__end
libdivajni.so
olsdfgad;lh
.dotdot
GCC: (GNU) 4.9 20140827 (prerelease)
.shstrtab
.hash
.dynsym
.dynstr
.rela.dyn
.rela.plt
.text
.rodata
.eh_frame_hdr
.eh_frame
.init_array
```

- 6.
7. You will get the correct string which can give us access : olsdfgad;lh



8.

## Remediation:

- Eliminate Hardcoding: Sensitive information should never be hardcoded. Instead, adopt secure coding practices to manage sensitive data.
- Implement Secure Storage: Use Android's Keystore system or other secure storage mechanisms for storing credentials safely.
- Use Environment Variables: Access sensitive data through environment variables or encrypted configuration files that can be securely accessed during runtime.
- Apply Obfuscation: Employ tools like ProGuard to obfuscate the code, making reverse engineering more challenging, although this should not be the sole security measure.

## CWE (Common Weakness Enumeration):

- CWE-798: Use of Hard-coded Credentials - This CWE addresses the vulnerabilities that arise from using hardcoded credentials, emphasizing the significant security risks when such information is exposed through the application's codebase.

# Insecure Data Storage – Part 1

## Description:

- The application stores sensitive user information, such as usernames and passwords, in an unencrypted XML file within the application's internal storage. This sensitive data is easily accessible by any attacker with root access or physical access to the device. Storing sensitive information without encryption leaves it vulnerable to unauthorized access and compromises user privacy.

## Impact:

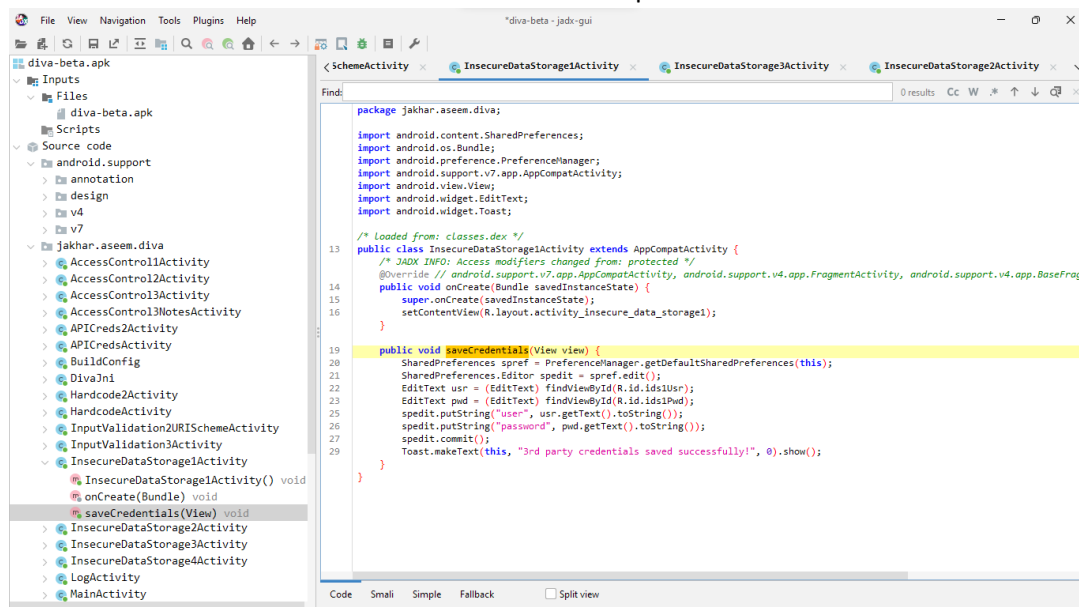
- An attacker with access to the device's storage (through rooting, malicious apps, or physical access) can retrieve sensitive user credentials.
- This can lead to account compromise, unauthorized access to backend systems, or misuse of user data.
- Users' privacy is at risk, as credentials stored in plaintext can be easily stolen and used for malicious purposes.

## Steps to Reproduce:

1. Launch the application and enter a username and password and confirm the credentials are saved.
2. Access the Device via ADB Shell: `adb shell`
3. Navigate to the App's Data Directory: `cd /data/data/application-name/`
4. Inside the app's internal storage, navigate to the location of the XML file where the credentials are saved. Open or list the content of the file, and you will see the username and password stored in plaintext.

## PoC (Proof of Concept):

1. As not stated where the credentials are stored at its possible that its stored at xml file



```
package jakhar.aseem.diva;

import android.content.SharedPreferences;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

/* Loaded from: classes.dex */
public class InsecureDataStorage1Activity extends AppCompatActivity {
    /* JADX INFO: Access modifiers changed from: protected */
    @Override // android.support.v7.app.AppCompatActivity, android.support.v4.app.FragmentActivity, android.support.v4.app.BaseFragmentActivity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_insecure_data_storage1);
    }

    public void saveCredentials(View view) {
        SharedPreferences spref = PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor spedit = spref.edit();
        EditText usr = (EditText) findViewById(R.id.ids1Usr);
        EditText pwd = (EditText) findViewById(R.id.ids1Pwd);
        spedit.putString("user", usr.getText().toString());
        spedit.putString("password", pwd.getText().toString());
        spedit.commit();
        Toast.makeText(this, "3rd party credentials saved successfully!", 0).show();
    }
}
```

2. Use the following ADB commands:
  - `adb shell`
  - `su`
  - `cd /data/data/jakhar.aseem.diva/shared_prefs/`
  - `cat jakhar.aseem.diva_preferences.xml`



3. The file will display the saved username and password in plaintext, confirming the vulnerability.

```
1|:/data/data/jakhar.aseem.diva/shared_prefs # cat jakhar.aseem.diva_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="password">foo</string>
  <string name="user">fool</string>
</map>
:/data/data/jakhar.aseem.diva/shared_prefs # |
```

## Remediation:

- **Encrypt Sensitive Data:** Store all sensitive information, including usernames and passwords, in encrypted form using the Android Keystore system.
- **Use Secure Storage Options:** Avoid saving sensitive data in shared preferences or internal storage without encryption.
- **Follow Best Practices:** Implement secure coding practices to ensure that sensitive information is protected at all times.
- **Perform Regular Security Audits:** Periodically check the app for vulnerabilities related to insecure storage and update the security mechanisms.

## CWE (Common Weakness Enumeration):

- **CWE-312: Cleartext Storage of Sensitive Information.** This CWE identifies the risk of storing sensitive information in cleartext, leading to unauthorized access if the storage is compromised

# Insecure Data Storage – Part 2

## Description:

- The application stores sensitive user information, such as usernames and passwords, in an unencrypted SQLite database file within the application's internal storage. This sensitive data is easily accessible by any attacker with root access or physical access to the device. Storing sensitive information without encryption leaves it vulnerable to unauthorized access and compromises user privacy

## Impact:

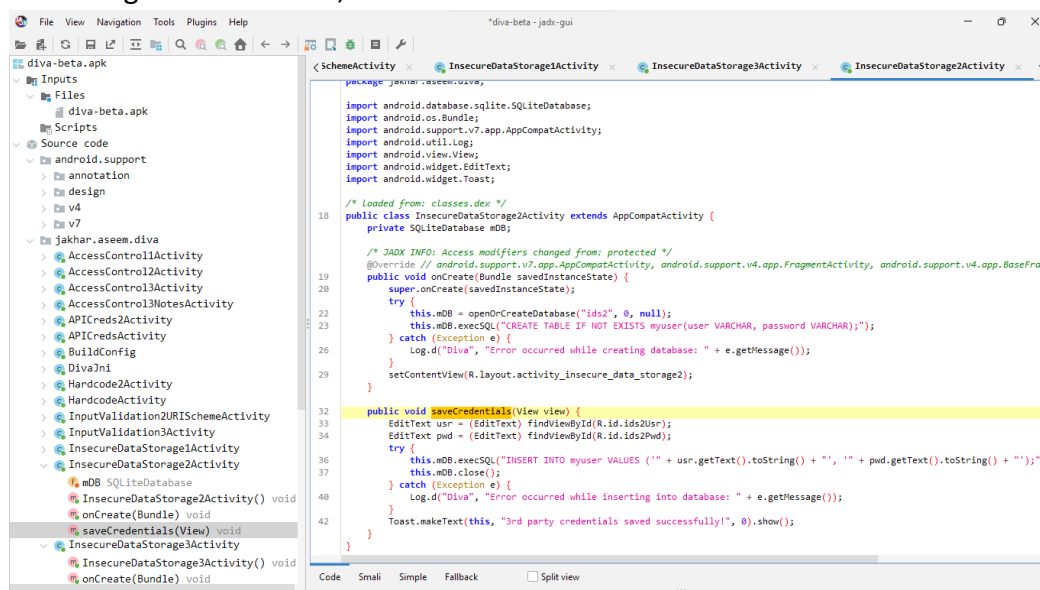
- Unauthorized Access: If an attacker gains access to the database file, they can easily read the stored credentials.
- Account Compromise: Compromised credentials could lead to unauthorized access to user accounts, causing potential data loss or leakage of sensitive information.
- Reputational Damage: The exposure of user data could lead to loss of trust and reputation for the organization, resulting in financial and legal consequences.

## Steps to Reproduce:

1. Launch the application and enter a username and password and confirm the credentials are saved.
2. Access the Database Directory with :
  - adb shell "ls /data/data/application.name/databases"
3. Pull the Database File i.e. Retrieve the database file to your local machine with:
  - adb pull "/data/data/application.name/databases/dbfolder"
4. Open and Analyze the Database

## Proof of Concept (PoC):

1. According to source code, we can see that credentials are saved at database



2. db directory: /data/data/jakhar.aseem.diva/databases
3. Use the following ADB commands:
  - a. adb shell "ls /data/data/jakhar.aseem.diva/databases"
  - b. adb pull "/data/data/jakhar.aseem.diva/databases/ids2"
4. Now ids2 file is pulled from an Android device (located at /data/data/jakhar.aseem.diva/databases/)
5. To read this database file use following commands

- sqlite3 ids2
- .tables
- SELECT \* FROM myuser;

```
sqlite> SELECT * FROM myuser;  
root|root
```

6.

## Remediation:

- **Implement Strong Encryption:** Use strong encryption algorithms (e.g., AES-256) to encrypt sensitive data before storing it in the database.
- **Use Secure Storage Solutions:** Leverage the Android Keystore system to manage cryptographic keys securely, which prevents unauthorized access to sensitive information.
- **Data Access Control:** Implement strict access controls to limit which parts of the application can access sensitive data.
- **Conduct Regular Security Audits:** Regularly review and audit the application's storage practices to identify and remediate potential vulnerabilities.
- **User Education:** Educate users about safe practices for handling their credentials and securing their accounts.

## Common Weakness Enumeration (CWE):

- **CWE-922: Insecure Storage of Sensitive Information** - This classification refers to the inadequate protection of sensitive data, leading to potential exposure.

# Insecure Data Storage – Part 3

## Description:

- In this scenario, the application creates a temporary file to store sensitive credentials, such as usernames and passwords, in plain text. This file is stored within the device's internal storage without encryption, leaving it vulnerable to unauthorized access. Storing sensitive data in temporary files, especially without encryption, exposes it to potential theft if an attacker gains access to the file system.

## Impact:

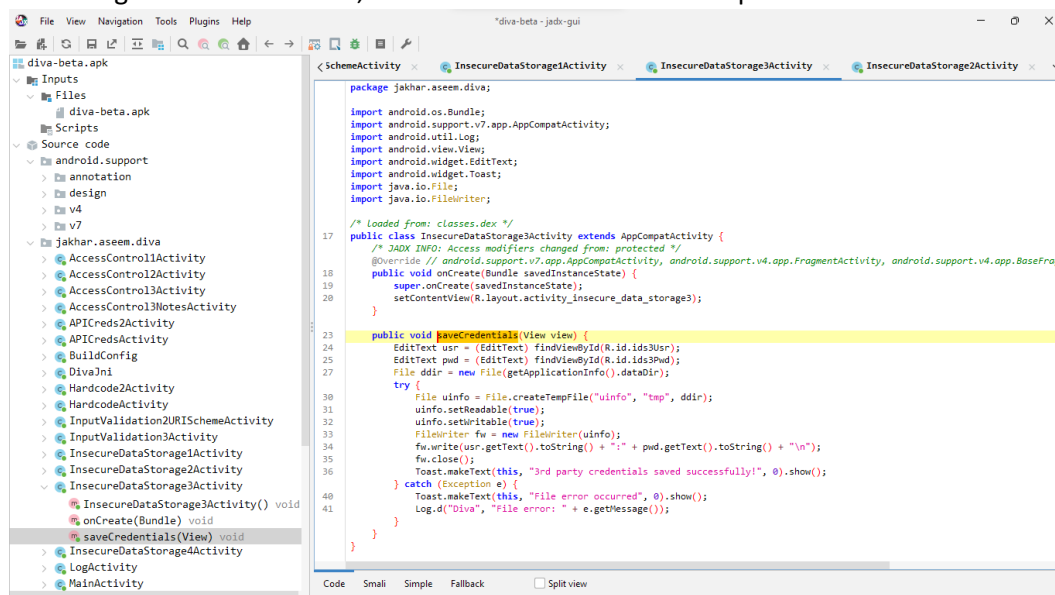
- Sensitive Data Exposure: Storing credentials in plain text means any attacker with file system access can read this data without any additional steps.
- Increased Attack Surface: If temporary files are not properly deleted after use, they could provide attackers with sensitive information long after the application's session ends.
- Reputation and Trust Loss: The exposure of user credentials could lead to security breaches, tarnishing the organization's reputation.

## Steps to Reproduce:

1. Launch the application and enter a username and password and confirm the credentials are saved.
2. Open the file system of the device and locate the temporary file in the application's directory.
  - adb shell ls /data/data/[application-package]/files/tmp/
3. Open the temporary file to reveal the credentials stored in plain text.

## Proof of Concept (PoC):

1. According to the source code, credentials are stored at a temp file



2. Temp file directory: `:/data/data/jakhar.aseem.diva/`
3. Use the following ADB commands:
  - adb pull /data/data/jakhar.aseem.diva/
  - cat uinfo2313633763112709918tmp

```
:/data/data/jakhar.aseem.diva # ls
cache          shared_prefs      uinfo2479473814745132580tmp
code_cache     uinfo1412247794022919423tmp uinfo8175160741345017432tmp
databases      uinfo2313633763112709918tmp
:/data/data/jakhar.aseem.diva # cat uinfo2313633763112709918tmp
foo:foo
```

4.

## Remediation:

- Avoid Temporary File Storage for Sensitive Data: Never store sensitive information, such as usernames and passwords, in temporary files.
- Encrypt Sensitive Data: If temporary storage is necessary, encrypt the data using strong encryption algorithms like AES before writing it to the file system.
- Secure Storage Locations: Store sensitive data using secure storage APIs like Android's SharedPreferences with encryption or the Keystore.
- File Deletion: Ensure that temporary files containing sensitive information are securely deleted after use.
- Periodic Audits: Perform regular security audits to identify and resolve insecure storage practices.

## CWE:

- CWE-312: Cleartext Storage of Sensitive Information - This refers to the storage of sensitive data in a format that is not protected from unauthorized access (e.g., plain text).

# Insecure Data Storage – Part 4

## Description:

- The application stores user credentials in an unprotected file on the external storage (SD card) without encryption. The file is named .uinfo.txt, and even though it is marked as hidden (with the dot prefix), it is still easily accessible. Storing sensitive information like this in a publicly accessible location allows any other application or attacker with access to the SD card to read this data without any special permissions.

## Impact:

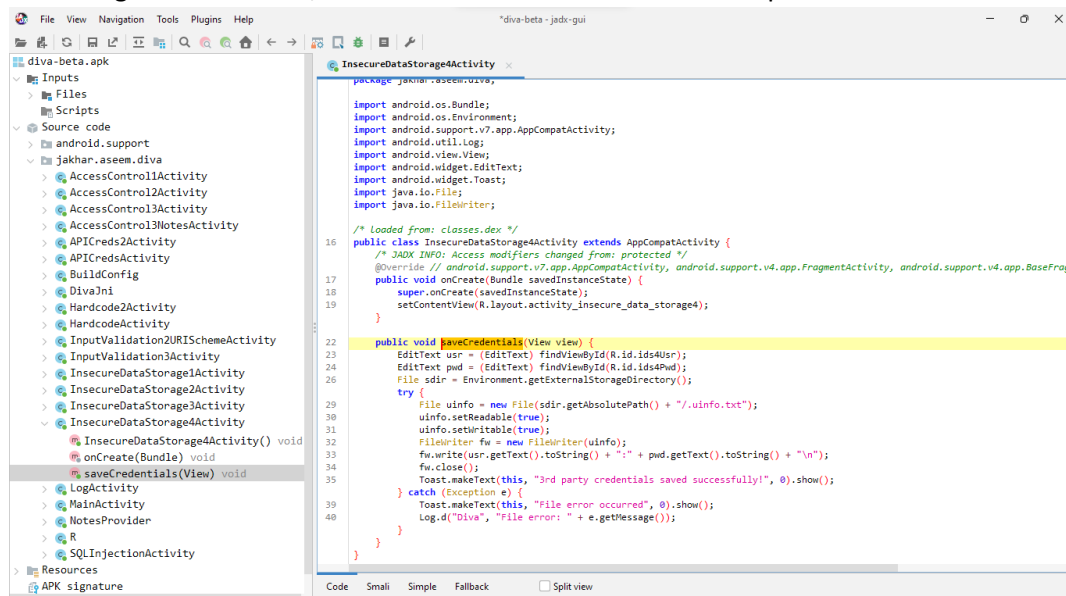
- Any malicious app or attacker with access to the SD card can retrieve sensitive user information like usernames and passwords.
- There is no encryption or protection applied to the file, making it trivial for attackers to access this data.
- Exposing credentials in such a manner can lead to unauthorized access to user accounts, identity theft, and further exploitation of backend systems that rely on these credentials.

## Steps to Reproduce:

1. Open the app and try to save a username and password. You may face an error saying that the file cannot be created due to insufficient permissions.
2. Go to the app's settings, open App Info, then navigate to Permissions and enable Storage permission for the app.
3. Return to the app and save the credentials again. This time the credentials will be saved successfully.
4. Check the Stored File, You will find the a hidden file .uinfo.txt containing the stored username and password in plaintext.

## PoC (Proof of Concept):

1. According to source code, credentials are stored at .uinfo.txt in plain text



2. Accessing the Stored File using following commands:

- adb shell
- cd /sdcard
- ls -la
- cat .uinfo.txt

3. The credentials will be displayed in plaintext, demonstrating the vulnerability.

```
vbox86p:/sdcard # cat .uinfo.txt  
root:root  
vbox86p:/sdcard # |
```

## Remediation:

- Do Not Store Sensitive Information on External Storage: Avoid storing any sensitive data, like credentials, on external storage (SD card), which is accessible to other apps and users.
- Use Internal Storage or Encrypted Storage: Store credentials in internal storage, which is private to the app, or use encrypted external storage if necessary.
- Encrypt the Data: If storing data externally is absolutely necessary, apply encryption to protect the data.
- Follow Android Security Best Practices: Ensure that the app follows secure data storage guidelines provided by Android, including using the Keystore system for credentials and sensitive information.

## CWE (Common Weakness Enumeration):

- CWE-922: Insecure Storage of Sensitive Information. This CWE identifies the risk of storing sensitive information in an insecure manner, allowing unauthorized access.

# Input Validation Issues – Part 1

## Description:

- The application is vulnerable to SQL Injection due to improper input validation when processing user inputs. SQL Injection (SQLi) occurs when an attacker can manipulate the SQL queries executed by the application by inserting or altering parts of the query via user inputs. In this case, the app is dynamically constructing SQL queries based on user input without sanitizing it properly, which leads to potential SQL Injection attacks.

## Impact:

- Database Compromise: An attacker can retrieve, modify, or delete sensitive information from the database.
- Information Disclosure: Sensitive user data (like usernames, passwords, or secret data) can be exposed.
- Data Integrity and Application Behavior: An attacker can manipulate the database and cause the application to behave unpredictably.
- Privilege Escalation: It could also lead to privilege escalation where an attacker gains unauthorized access to admin-level functionality or sensitive data.

## Steps to Reproduce:

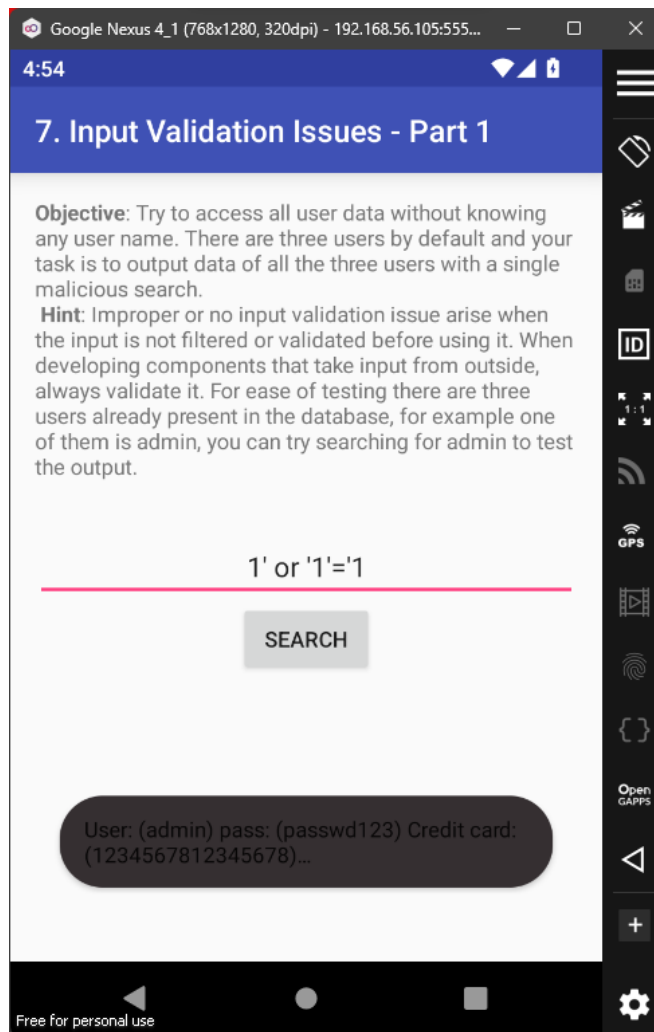
1. Open the app and go to the section where user credentials or data is processed.
2. Start by entering a single quote (') as input.
3. Use ADB to monitor the log with:
  - adb logcat | grep SQL
4. You should see an SQL error indicating improper handling of input.
5. Enter two or three quotes to see if the application throws additional SQL errors, confirming the input is processed directly in a SQL query.
6. Enter a query like 1' OR '1' = '1 to execute a "true" condition, allowing you to extract user data from the database. After injecting a successful query, the app will return data from the database, displaying the secret data associated with the users.

## PoC (Proof of Concept):

1. After inputting ' OR '1'='1, the logcat will display the SQL query being executed, showing how the injected condition retrieves data from the database.

```
PraFul@DESKTOP-ppm17 MINGW64 ~
$ adb logcat | grep sql
10-08 03:56:44.566 142 142 I keystore2: keystore2_main: Setting up sqlite lo
gging for keystore2
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eCursor.finalize(SQLiteCursor.java:282)
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eCursor.finalize(SQLiteCursor.java:291)
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eCursor.<init>(SQLiteCursor.java:98)
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eDirectCursorDriver.query(SQLiteDirectCursorDriver.java:52)
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eDatabase.rawQueryWithFactory(SQLiteDatabase.java:1712)
10-08 03:57:10.941 1602 1632 D StrictMode: at android.database.sqlite.SQLite
eQueryBuilder.query(SQLiteQueryBuilder.java:592)
10-08 04:39:30.180 2506 2506 E SQLiteLog: (1) unrecognized token: "'" in "SE
LECT * FROM sqliuser WHERE user = '"
10-08 04:39:30.181 2506 2506 D Diva-sqli: Error occurred while searching in da
tabase: unrecognized token: "'" (code 1 SQLITE_ERROR): , while compiling: SELE
CT * FROM sqliuser WHERE user = '"
10-08 04:40:09.551 2506 2506 E SQLiteLog: (1) unrecognized token: "'" in "SELE
CT * FROM sqliuser WHERE user = '1' or '1'=1"
10-08 04:40:09.551 2506 2506 D Diva-sqli: Error occurred while searching in da
tabase: unrecognized token: "'" (code 1 SQLITE_ERROR): , while compiling: SELECT
* FROM sqliuser WHERE user = '1' or '1'=1"
10-08 04:40:11.642 2506 2506 E SQLiteLog: (1) unrecognized token: "'" in "SELE
CT * FROM sqliuser WHERE user = '1' or '1'=1"
10-08 04:40:11.643 2506 2506 D Diva-sqli: Error occurred while searching in da
tabase: unrecognized token: "'" (code 1 SQLITE_ERROR): , while compiling: SELECT
* FROM sqliuser WHERE user = '1' or '1'=1"
```





- 2.
3. This confirms the SQL query is vulnerable and returns unauthorized data from the database.

## Remediation:

- Input Validation: Always sanitize user inputs by using prepared statements (parameterized queries) to prevent SQL Injection.
- Use ORM (Object Relational Mapping): Use ORM frameworks that abstract SQL queries and minimize the risk of SQL Injection by design.
- Input Escaping: If dynamic queries are necessary, ensure user input is properly escaped before constructing SQL queries.
- Error Handling: Avoid leaking detailed SQL error messages in logs, as these can provide attackers with information on how to exploit the application.

## CWE (Common Weakness Enumeration):

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). This CWE identifies vulnerabilities where user input is used to manipulate the structure of SQL queries inappropriately, leading to data leaks and compromise.

# Input Validation Issues – Part 2

## Description:

- The application allows a user to input URLs that are directly passed to the webview component without proper validation or sanitization. This leads to a path traversal vulnerability where malicious users can access sensitive internal device files by inputting file paths (e.g., to the shared preferences XML file) instead of valid web URLs. This issue is a consequence of failing to restrict input to valid web addresses and not preventing access to local system files.

## Impact:

- Data Exposure: Attackers can view confidential files, including those that store user credentials, application preferences, and sensitive configuration data.
- Unauthorized Access: An attacker can access any file the application has permissions to read, including potentially sensitive files stored on the device (such as in the /data/data/ or /sdcard directories).
- Security Bypass: This could lead to unauthorized access to data that should be restricted to specific functions or users within the application.

## Steps to Reproduce:

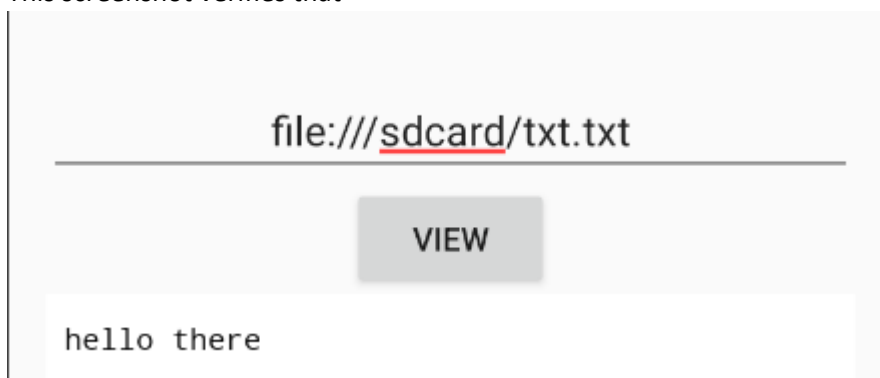
1. Open the part of the application where URLs are processed and loaded within a webview.
2. Instead of entering a web URL, input a local file URL that points to sensitive files such as:
  - file:///data/data/jakhar.aseem.diva/shared\_prefs/jakhar.aseem.diva\_preferences.xml
3. Upon submitting the input, the application will load the content of the internal XML file within the webview, displaying sensitive information (e.g., credentials).

## PoC (Proof of Concept):

1. This is text file we create to verify that application can also view stored files

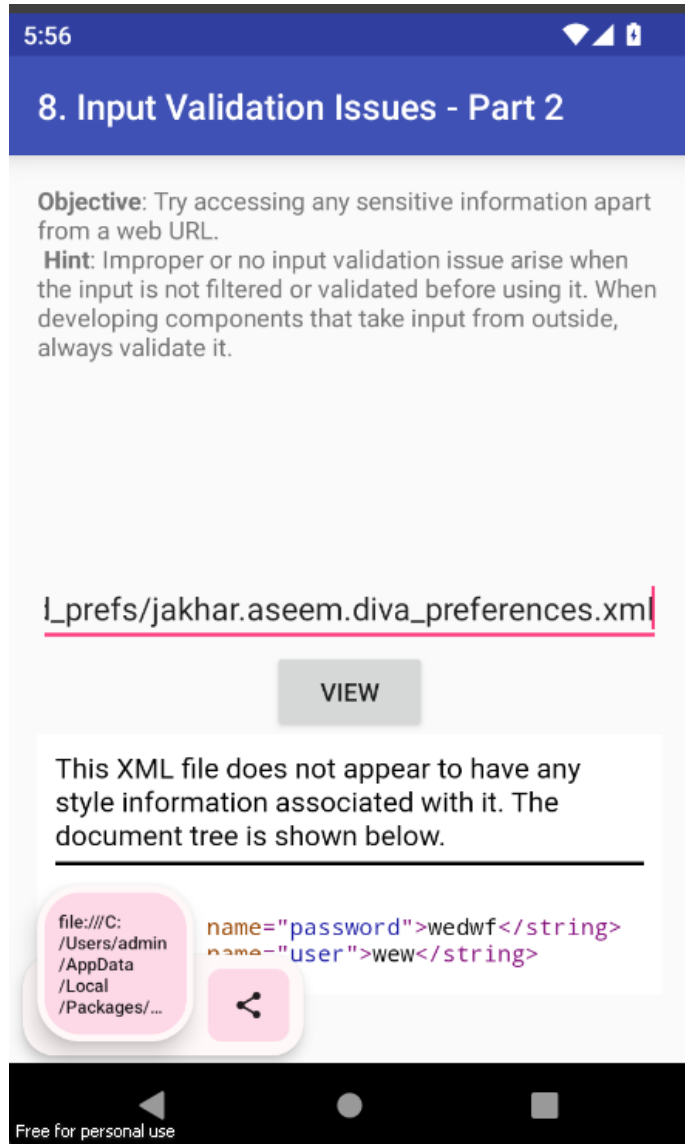
```
vbox86p:/sdcard # ls
Alarms  Audiobooks  Documents  Movies  Notifications  Podcasts  Ringtones
Android DCIM      Download   Music   Pictures      Recordings
vbox86p:/sdcard # echo "hello there" > txt.txt
vbox86p:/sdcard # ls
Alarms  Audiobooks  Documents  Movies  Notifications  Podcasts  Ringtones
Android DCIM      Download   Music   Pictures      Recordings  txt.txt
```

2. This screenshot verifies that



3. Input Local File URL to:
  - file:///data/data/jakhar.aseem.diva/shared\_prefs/jakhar.aseem.diva\_preferences.xml

4. After entering this URL into the application's input field, the XML file storing Sensitive data will be visible to the user without any restrictions.



- 5.

## Remediation:

- Input Validation: The application should strictly validate that the input URLs conform to expected web URLs (e.g., using regex) and reject any attempts to access local file paths.
- Restrict File Access: Ensure that the application does not have permissions to access system files unnecessarily, especially sensitive directories like /data/data/ and /sdcard/.
- Sanitize Inputs: Properly sanitize and escape user inputs before processing them within the webview.
- Use Security Permissions: Implement strict permission checks and ensure that sensitive file access is limited and properly guarded against unauthorized access.

## CWE (Common Weakness Enumeration):

- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'). This CWE identifies issues where attackers can bypass security measures by manipulating input to gain unauthorized access to files outside the expected directories.

# Input Validation Issues – Part 3

## Description:

- This vulnerability arises from improper input validation when handling user input in the application. The app fails to sanitize and limit the type and amount of input characters effectively. By inputting large quantities of random characters or strings, including numbers, symbols, or letters, an attacker can cause the app to crash, leading to a Denial of Service (DoS) condition.

## Impact:

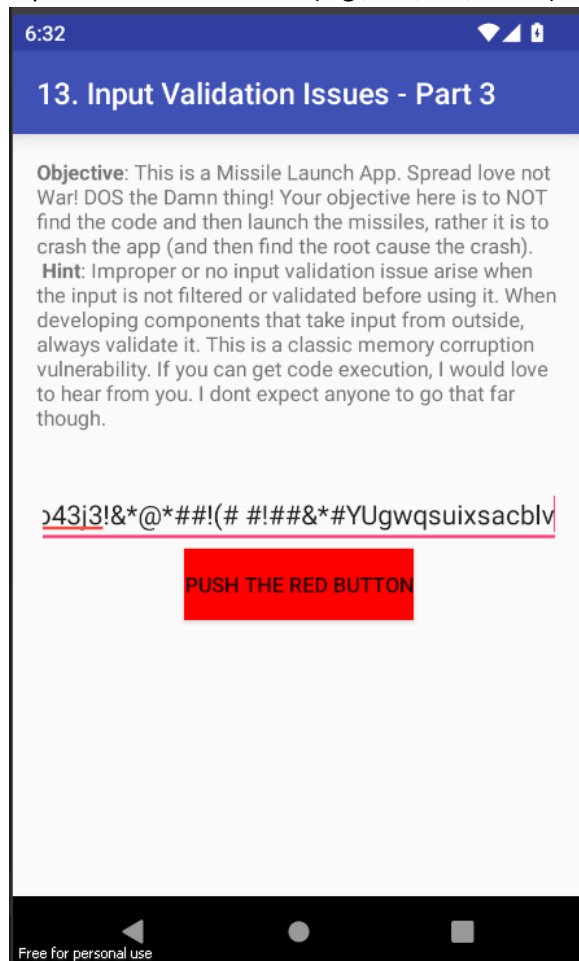
- Denial of Service: Attackers can crash the application by flooding it with random characters, making it inaccessible to legitimate users.
- Loss of Availability: If exploited, this vulnerability can make the application unavailable to users for an extended time, reducing its reliability.

## Steps to Reproduce:

1. Open the Application.
2. Start entering random characters (letters, numbers, symbols) into any input field.
3. Continuously add random characters in bulk or rapid succession.
4. After a sufficient number of inputs, the application will crash, causing a denial of service.

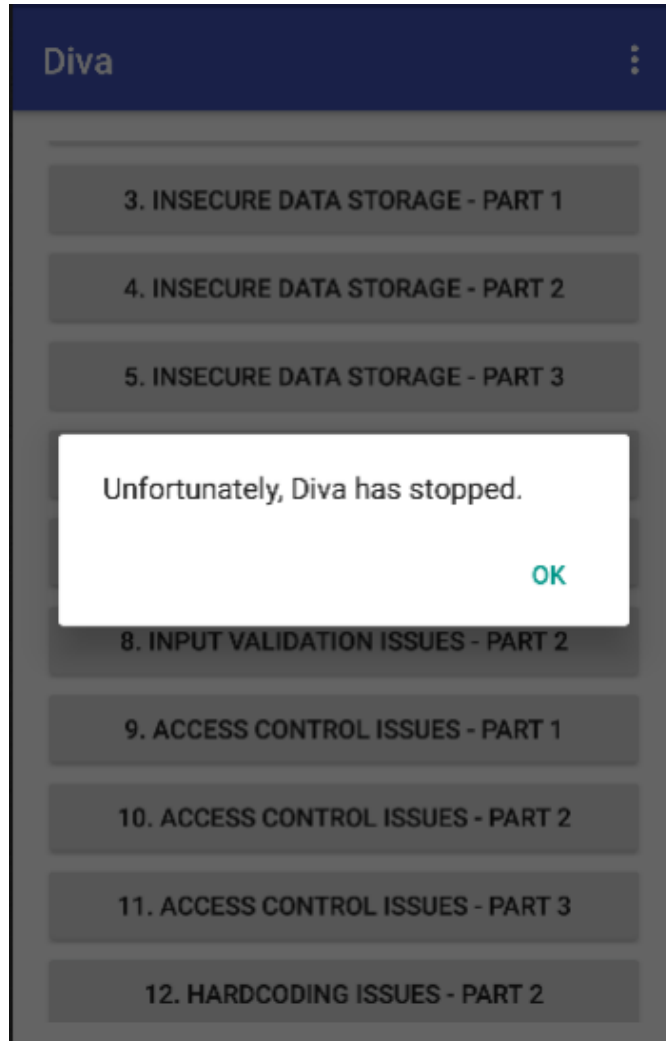
## PoC (Proof of Concept):

1. Launch the app and go to any input field.
2. Input random characters (e.g., "a", "#", "123") repeatedly.



3.

4. Continue to increase the frequency and volume of character inputs.
5. After a certain threshold, observe the crash, which triggers the denial of service.



6.

## Remediation:

- Enforce Input Validation: Implement strict input validation to limit the types and amount of characters allowed in input fields.
- Input Length Limit: Set maximum input limits to prevent large character inputs that could overwhelm the application.
- Graceful Error Handling: Handle erroneous or excessive inputs with appropriate error messages instead of allowing the app to crash.
- Resource Limiting: Restrict the resources allocated to user input to prevent denial of service from heavy input loads.

## CWE (Common Weakness Enumeration):

- CWE-400: Uncontrolled Resource Consumption. This vulnerability describes how excessive input data (e.g., long strings of characters) can overwhelm an application, leading to crashes or denial of service.

# Access Control Issues – Part 1

## Description:

- This vulnerability arises due to insecure activity exposure in the application. The application exposes an activity, jakhar.aseem.diva.action.VIEW\_CREDS, that allows attackers to bypass the normal user interface protections and directly access sensitive API credentials. By launching this hidden activity, an attacker can retrieve the API credentials without needing to press the “VIEW API CREDENTIALS” button in the app. This issue is a result of the app not properly restricting access to sensitive activities.

## Impact:

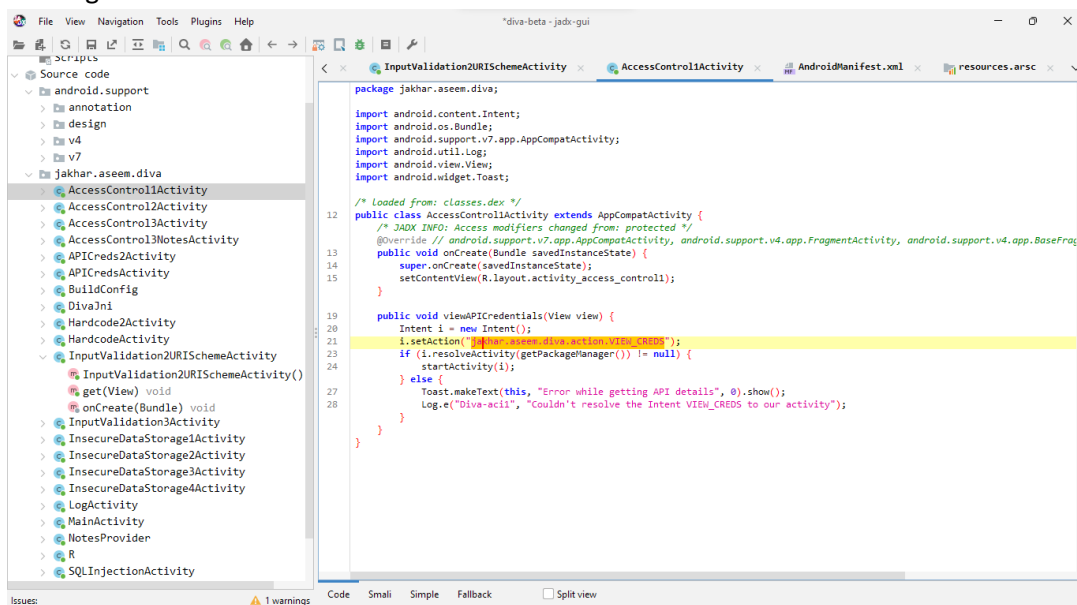
- **API Key Exposure:** Attackers can gain access to the API credentials, which may allow them to interact with the app’s backend services.
- **Bypass Authentication:** Attackers can bypass authentication and authorization mechanisms, leading to unauthorized access to sensitive data and application functionalities.
- **Potential Backend Manipulation:** With access to API keys, attackers could manipulate backend systems or impersonate the application, potentially causing further security breaches.

## Steps to Reproduce:

1. Use JADX-GUI to decompile the APK and inspect the source code.
2. Locate the activity named jakhar.aseem.diva.action.VIEW\_CREDS responsible for viewing API credentials.
3. Open a terminal or PowerShell and run the following command to access the Android shell:
  - adb shell
4. Launch the vulnerable activity and bypass the user interface protections:
  - am start -a jakhar.aseem.diva.action.VIEW\_CREDS
5. The API credentials will be displayed without requiring interaction with the app’s UI.

## PoC (Proof of Concept):

1. According to source code action named ” jakhar.aseem.diva.action.VIEW\_CREDS” responsible for viewing API credentials.

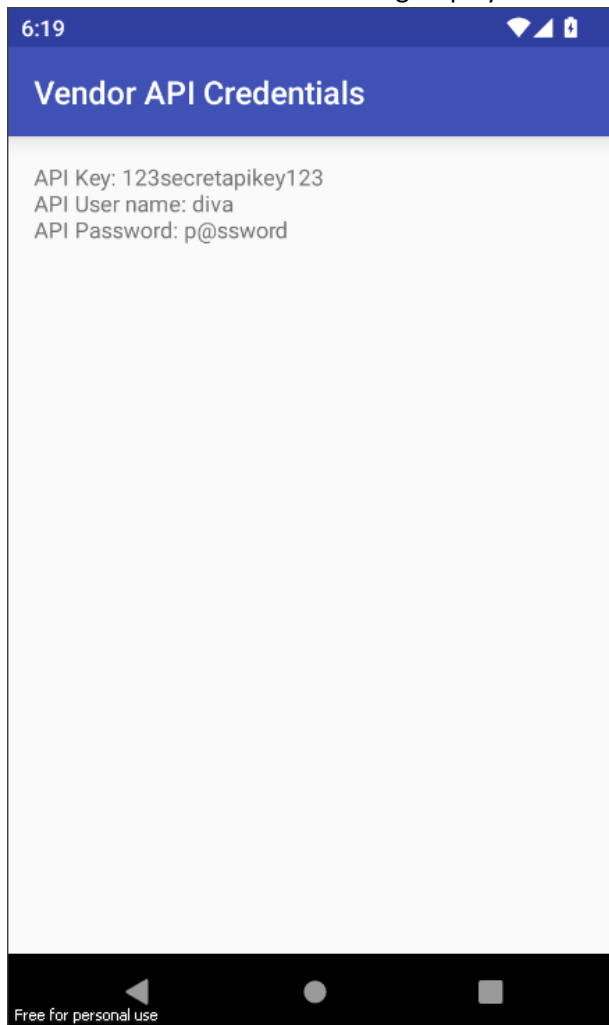


2. Run adb shell.

3. Use the command: `am start -a jakhar.aseem.diva.action.VIEW_CREDS` to launch the activity.

```
vbox86p:/ # am start -a jakhar.aseem.diva.action.VIEW_CREDS
Starting: Intent { act=jakhar.aseem.diva.action.VIEW_CREDS }
vbox86p:/ # |
```

4. Observe the API credentials being displayed without any further authentication requirements.



- 5.

## Remediation:

- **Restrict Activity Access:** Protect sensitive activities by requiring proper permissions or authentication checks before they are started. Activities that expose sensitive data should not be freely accessible via an intent.
- **Use Explicit Permissions:** Ensure that sensitive actions can only be triggered by trusted components within the application.
- **Apply Authorization Logic:** Implement strict access control measures to verify that the user has proper authorization before allowing access to sensitive data or functions.

## CWE (Common Weakness Enumeration):

- **CWE-285: Improper Authorization.** This CWE applies when applications fail to properly verify whether a user is allowed to perform a specific action, such as accessing sensitive activities like viewing API credentials.

# Access Control Issues – Part 2

## Description:

- The application fails to properly validate the user when accessing the API credentials activity. Using reverse engineering tools like Jadx, attackers can identify the activity responsible for viewing sensitive information (API credentials). By bypassing the PIN validation process, an unauthorized user can directly access and view these credentials, which should be protected.

## Impact:

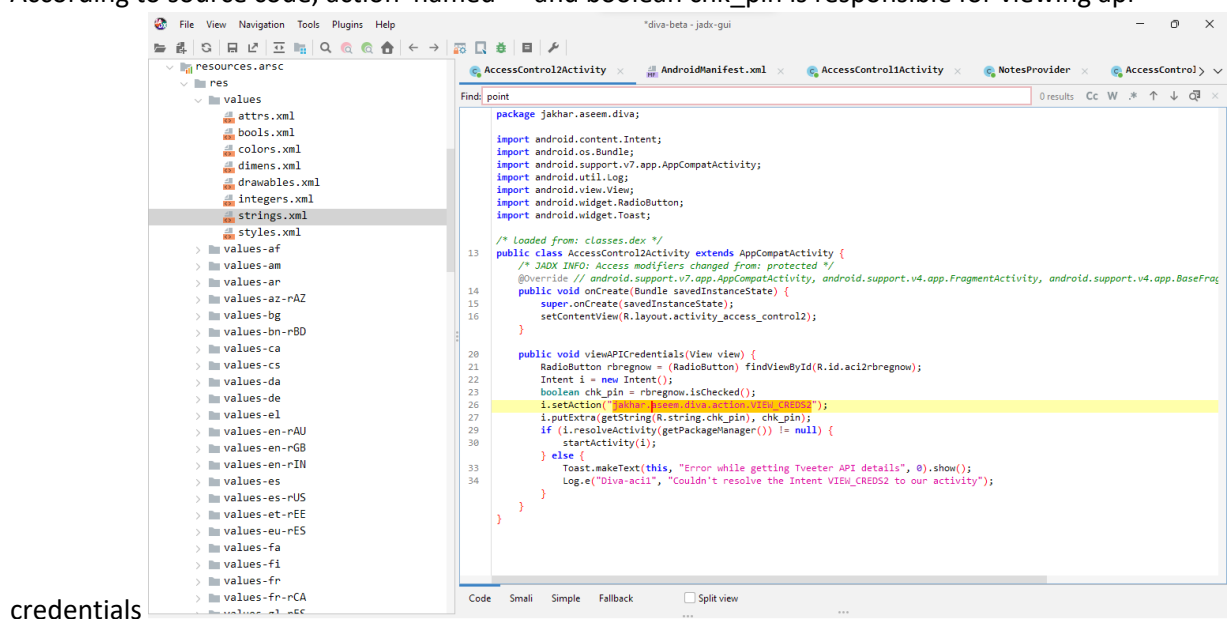
- View sensitive API credentials without proper authorization.
- Use these credentials to perform actions on behalf of the legitimate user, compromising the integrity of the application.
- Leak sensitive information that can lead to broader security issues such as unauthorized access to backend services.

## Steps to Reproduce:

1. Use Jadx-gui or a similar reverse engineering tool to inspect the source code of the DIVA application.
2. Find the activity responsible for checking the Tveeter PIN, likely stored within APICreds2Activity.
3. Start the Activity Using ADB:
  - adb shell
4. Start the API credentials activity, bypassing the PIN check with the following command:
  - am start -n jakhar.aseem.diva/.APICreds2Activity -a jakhar.aseem.diva.action.VIEW\_CREDS2 --ez check\_pin false
5. After executing the command, observe the result, where the API credentials will be displayed on the screen.

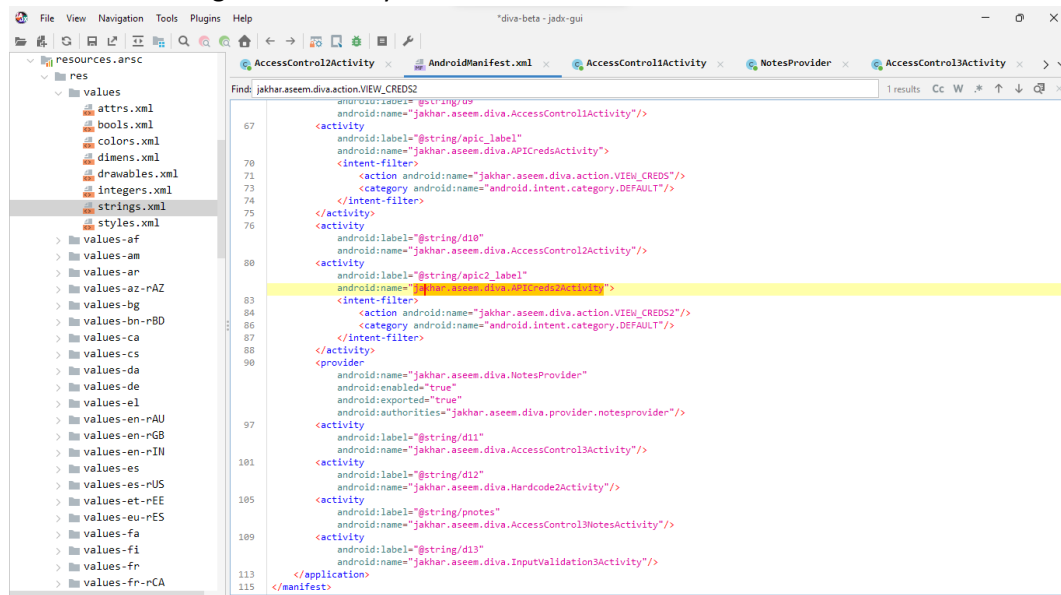
## PoC (Proof of Concept):

1. According to source code, action named "" and boolean chk\_pin is responsible for viewing api

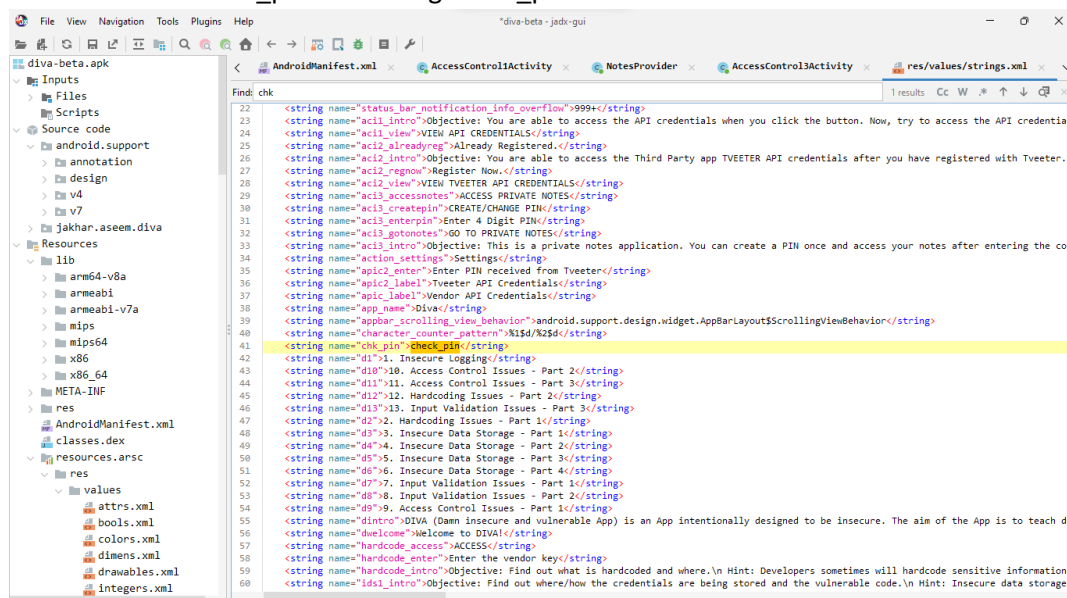




## 2. This action belongs from activity named ""



## 3. Boolean variable chk\_pin uses string check\_pin



## 4. Execute the following command to bypass the PIN:

- adb shell am start -n jakhar.aseem.diva/.APICreds2Activity -a jakhar.aseem.diva.action.VIEW\_CREDS2 -ez check\_pin false

```
vbox86p:/ # am start -a jakhar.aseem.diva.action/.VIEW_CREDS2 -n jakhar.aseem.diva/.APICreds2Activity --ez check_pin false
Starting: Intent { act=jakhar.aseem.diva.action/.VIEW_CREDS2 cmp=jakhar.aseem.diva/.APICreds2Activity (has extras) }
vbox86p:/ #
```

6. This command starts the activity and bypasses the need for a PIN validation, leading to the exposure of Tveeter API credentials.



7.

## Remediation:

- Implement Proper Access Control: Ensure that sensitive activities such as API credential views require proper authorization, not just through an intent flag but through a secure validation mechanism.
- Use Secure Authentication Mechanisms: Implement secure PIN checks that cannot be bypassed with ADB commands, such as server-side validation.
- Code Obfuscation: Use ProGuard or another obfuscation tool to make it harder to identify and exploit sensitive activities in the source code.

## CWE (Common Weakness Enumeration):

- CWE-285: Improper Authorization - This weakness occurs when an application does not perform necessary checks to ensure that a user is allowed to perform an action or access a resource, leading to unauthorized access.

# Access Control Issues – Part 3

## Description

- This vulnerability stems from insufficient access control mechanisms within the DIVA application, specifically in handling sensitive user data stored in shared preferences. The application allows unauthorized access to private notes and API credentials without requiring the correct user PIN. This oversight enables attackers to retrieve sensitive information directly from the application's data storage, undermining the security and confidentiality intended for user data.

## Impact

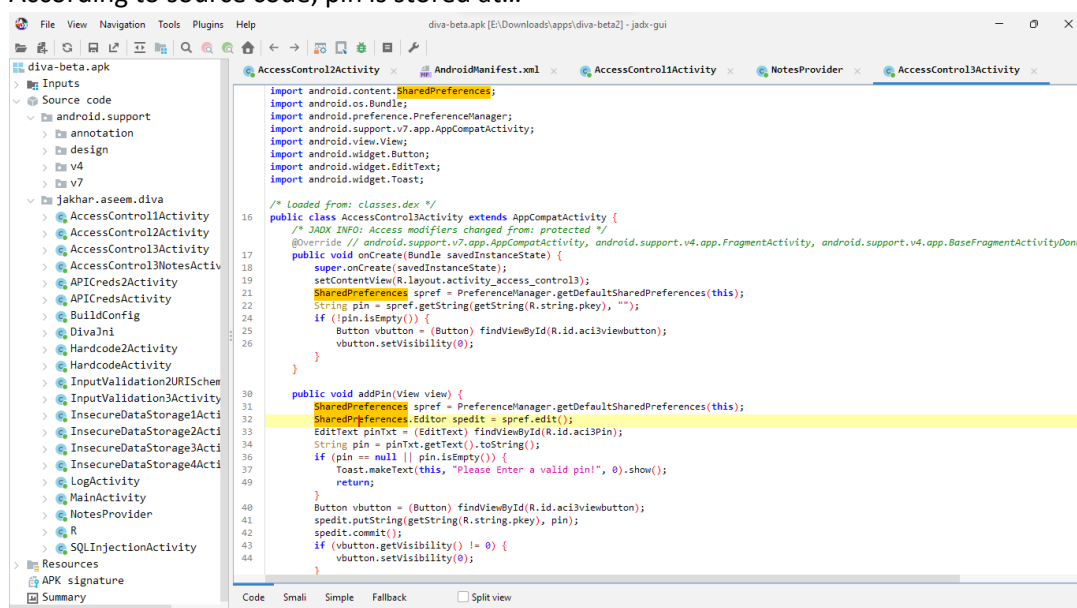
- Data Exposure: Unauthorized access to sensitive information, including private notes and API credentials, which could lead to data leaks and privacy violations.
- User Privacy Compromise: The ability for an attacker to view personal notes can result in significant privacy breaches, especially if these notes contain sensitive or personal information.
- Integrity Risks: Attackers might modify sensitive data, leading to further unauthorized changes that could disrupt application functionality or user experience.
- Reputation Damage: Such vulnerabilities can erode user trust in the application, potentially leading to decreased usage, legal consequences, and reputational harm.

## Steps to Reproduce

1. Use Jadx-gui to examine the DIVA application's code and locate the section where the PIN is stored and validated.
2. Open a command prompt or PowerShell and execute the command to obtain a shell on the Android device: `adb shell`
3. To bypass PIN authentication and retrieve private notes, execute the following content provider query:
  - `content query --uri content://jakhar.aseem.diva.provider`

## PoC (Proof of Concept)

1. According to source code, pin is stored at...

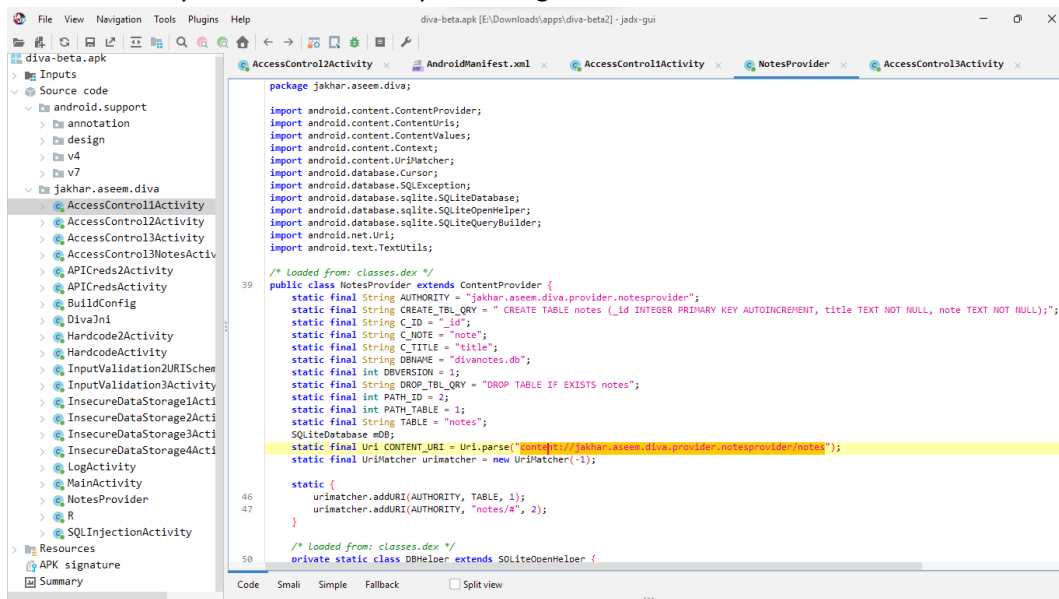


2. So we can access pin by accessing xml file by using commands
  - `adb shell`
  - `cd /data/data/jakhar.aseem.diva/shared_prefs`

- cat jakhar.aseem.diva\_preferences.xml

```
vbox86p:/data/data/jakhar.aseem.diva/shared_prefs # cat jakhar.aseem.diva_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="notespin">0987</string>
</map>
```

- 3.
4. We can directly access the notes by accessing this location



5. Use this command to access this content
  - adb shell
  - content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes

```
E:\Downloads\apps\diva-beta2>adb shell
vbox86p:/ # content query --uri content://jakhar.aseem.diva.provider.notesprovider/notes
Row: 0 _id=5, title=Exercise, note=Alternate days running
Row: 1 _id=4, title=Expense, note=Spent too much on home theater
Row: 2 _id=6, title=Weekend, note=b3333333333333r
Row: 3 _id=3, title=holiday, note=Either Goa or Amsterdam
Row: 4 _id=2, title=home, note=Buy toys for baby, Order dinner
Row: 5 _id=1, title=office, note=10 Meetings. 5 Calls. Lunch with CEO
vbox86p:/ #
```

- 6.

## Remediation:

- Strengthen Access Controls: Implement strict authentication measures before allowing access to sensitive data. Ensure that user-defined credentials (like PINs) are effectively validated.
- Encrypt Sensitive Data: Store sensitive information securely by encrypting data in shared preferences to prevent unauthorized access.
- Limit Data Access: Ensure that sensitive data access is restricted to authorized users only, utilizing robust access control mechanisms.
- Regular Security Audits: Conduct routine security assessments to identify and rectify potential access control vulnerabilities.
- User Education: Inform users about the importance of choosing strong PINs and the risks of weak credentials.

## CWE (Common Weakness Enumeration):

- CWE-284: Improper Access Control - This vulnerability is categorized under improper access control, which occurs when an application fails to adequately restrict access to sensitive resources. This oversight can result in unauthorized users performing actions or accessing data they should not have permission to, leading to significant security issues.