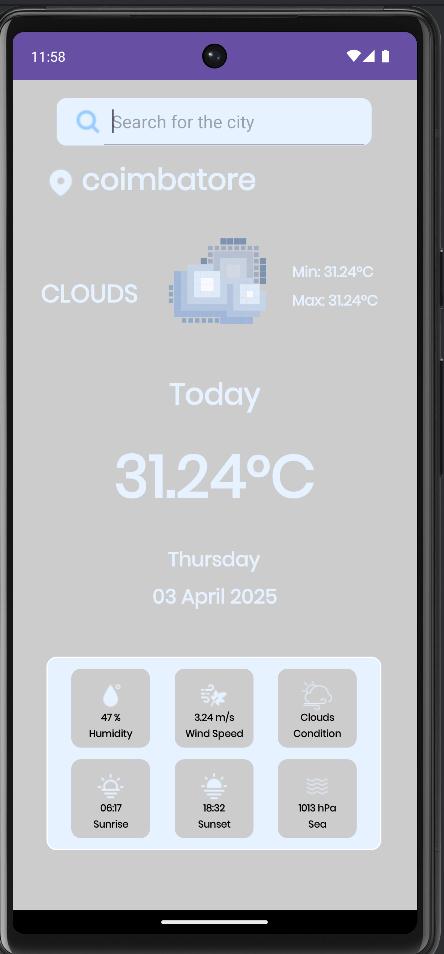
Mobile Security Package - Building a vulnerable Weather Forecast App

**Abstract:** This document provides an overview and security analysis of a weather forecast application built in Kotlin. The application allows users to enter a state name to retrieve weather forecast details via a backend API. However, the app contains six critical vulnerabilities that expose sensitive data, allow unauthorized access, and permit backend API abuse. This document presents a detailed walkthrough of each vulnerability, demonstrating how an attacker can exploit them, along with mitigation strategies to secure the application.

Link for APK:

Team Members:

* 21PC15 - Harshad Krishna BS
* 21PC16 - Harsith S
* 21PC27 - Sahil Jamadar SK

**APP MAIN PAGE: **

**Walkthrough of Vulnerabilities:**

### **1) Hidden Function in Code Base (Decompilation and UI Modification)**



**Description:** The APK contains a hidden login button that is not visible by default. The attacker must decompile the APK, modify the UI elements, and recompile it to reveal the login functionality.

**Exploitation Steps:**

Decompile the APK using tools like jadx or apktool:  
  
 jadx -d output\_folder weatherapp.apk

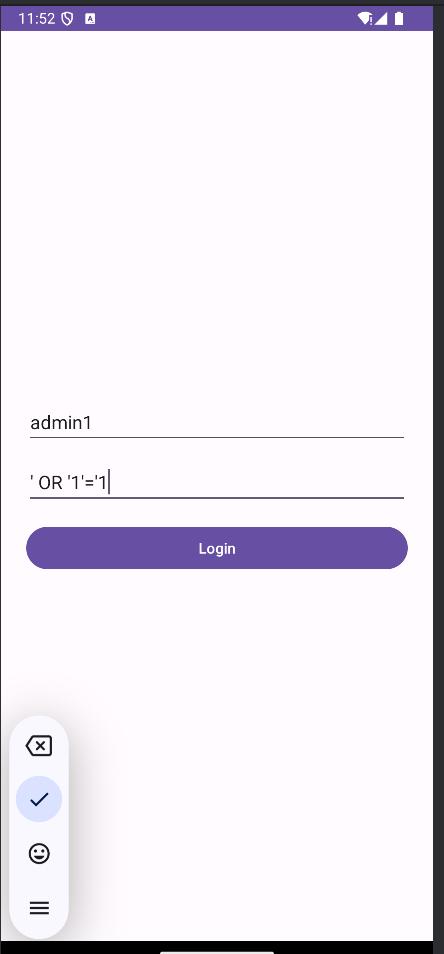
1. Locate the UI-related XML and Kotlin files in the decompiled code.
2. Modify the relevant XML to make the login button visible.
3. Recompile the APK and install it to access the login page.

**Mitigation:**

* Remove unused UI components instead of hiding them.
* Implement runtime validation to prevent UI modifications.

### **2) SQL Injection in Login Page**

**Description:** The login page does not have a sign-up option, making brute-force attempts viable. The default username is admin1, and the backend is vulnerable to SQLite injection.



**Exploitation Steps:**

1. Enter admin1 as the username.
2. Use the following SQL injection payload in the password field:  
     
    **' OR 1=1 --**
3. This bypasses authentication, granting access.

**Mitigation:**

* Use parameterized queries.
* Implement rate limiting and CAPTCHA for authentication.

### **3) Encrypted Flag Exposure via Logcat**

**Description:** When querying a state, an encrypted flag is logged via Logcat. The encryption uses AES and Base64 encoding.



**Exploitation Steps:**

1. Run the application and monitor Logcat:  
     
    adb logcat | grep "flag"
2. Extract the Base64-encoded flag.

**Decode using:**

1. import base64

print(base64.b64decode("ENCODED\_FLAG\_HERE").decode())

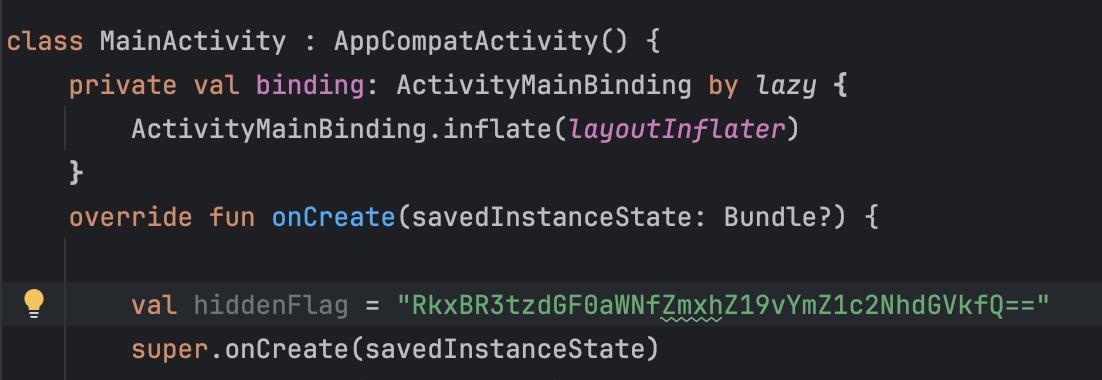
1. If further decryption is required, analyze the AES key in the source code.

**Mitigation:**

* Do not log sensitive data.
* Use environment variables or secure storage for encryption keys.

### **4) Static Analysis of MainActivity.kt for a Hidden Flag**

**Description:** The MainActivity.kt file contains a Base64-encoded flag stored in plaintext.



**Exploitation Steps:**

1. Decompile the APK and locate MainActivity.kt.
2. Identify the Base64-encoded flag.

Decode it using:

1. import base64

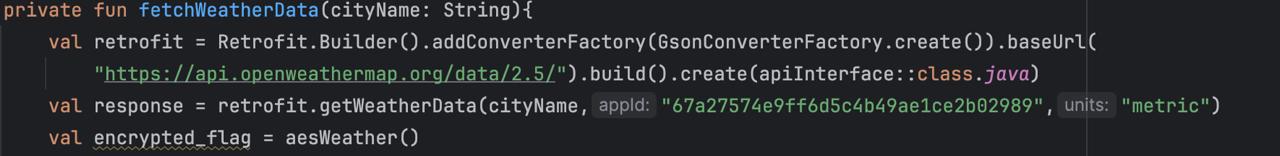
print(base64.b64decode("ENCODED\_FLAG\_HERE").decode())

**Mitigation:**

* Store sensitive data in a secure backend instead of embedding it in the app.

### **5) Exposed Backend API Key**

**Description:** The APK contains the backend API key in plaintext, allowing attackers to misuse it.



**Exploitation Steps:**

1. Extract the API key from the decompiled code.

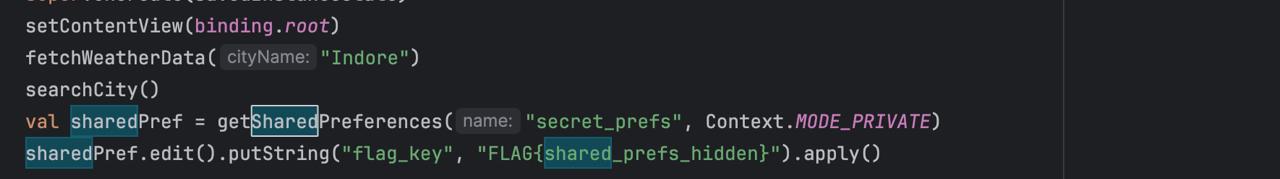
Use it in API calls to retrieve unauthorized data.  
  
 curl -H "Authorization: Bearer API\_KEY\_HERE" https://api.weatherapp.com/data

**Mitigation:**

* Use environment variables or secure storage for API keys.
* Implement rate limiting and request validation on the backend.

### **6) Vulnerability in SharedPreferences**

**Description:** Sensitive user information is stored insecurely in SharedPreferences, making it accessible to an attacker.



**Exploitation Steps:**

Extract stored data from SharedPreferences:

1. adb shell

cd /data/data/com.weatherapp/shared\_prefs

cat preferences.xm

1. Retrieve stored credentials or tokens.

**Mitigation:**

* Use encrypted SharedPreferences (EncryptedSharedPreferences).
* Store sensitive data in secure storage solutions like Android Keystore.

**Conclusion:** This document highlights the security flaws present in the weather forecast app and provides a step-by-step walkthrough for their exploitation. To secure the application, developers must follow secure coding practices, implement strong authentication mechanisms, and avoid storing sensitive data insecurely.

—-------------------------------------------------------------------------------------------------------------------------