MOBILE APPLICATION TESTING LAB

1. EXECUTIVE SUMMARY

This report documents a complete mobile application testing lab that demonstrates static and dynamic analysis of an Android APK using MobSF, Frida, and Drozer. The lab covers environment setup (Android Studio + AVD emulator), static analysis with MobSF, decompilation (JADX/apktool), dynamic instrumentation (Frida), IPC/component testing (Drozer), proof-of-concept (PoC) actions and remediation recommendations.

Key findings:

- Static analysis identified possible insecure storage and exported components.
- Decompilation revealed authentication-related classes and methods for dynamic testing.
- Frida-based dynamic instrumentation allowed runtime manipulation of authentication checks (PoC demonstrated a forced-authentication hook).
- Drozer enumeration exposed exported content providers and activities that could leak data if unprotected.

2. SCOPE & OBJECTIVES

Scope:

- Target platform: Android (APK) — tested on an Android Emulator (AVD) created with Android Studio.
- Tools used: Mobile Security Framework (MobSF) for static analysis, JADX & apktool for decompilation, Frida for runtime hooking, Drozer for IPC/component testing.

Objectives:

1. Set up a reproducible local testing environment (Android Studio + AVD, adb, Docker for MobSF).
2. Perform static analysis to identify high-level issues and potential targets.
3. Decompile the APK to find class/method names for dynamic testing.
4. Perform runtime instrumentation to bypass or modify authentication checks (PoC) using Frida.
5. Enumerate IPC and exported components with Drozer to identify insecure endpoints.

6. Produce remediation recommendations and a checklist for secure development.

3. TEST ENVIRONMENT

Host machine: Laptop / Desktop with 8–16+ GB RAM, virtualization enabled in BIOS/UEFI.

Operating System: Windows / macOS / Linux (Ubuntu 20.04+ recommended).

Tools & versions (examples — record exact versions used in your run):

- Android Studio (with AVD Manager) — used to create emulator.
- Android Emulator (x86_64 system image) — emulator-5554.
- Android SDK Platform-tools (adb).
- MobSF (Docker image: opensecurity/mobile-security-framework-mobsf:latest) or local MobSF.
- Frida (frida-tools on host, frida-server on emulator) — host frida version and matching frida-server binary.
- Drozer client & drozer-agent APK.

4. METHODOLOGY & STEP-BY-STEP PROCEDURES

4.1 Android Studio & Emulator Setup

- Download and install Android Studio from the official site.
- Use SDK Manager to install Platform-tools and an x86_64 system image for a chosen Android API level.
- Create an AVD using an x86_64 system image and start the emulator.
- Verify connectivity via adb devices (expected: emulator-5554 device).

4.2 Installing & Running MobSF (Static Analysis)

- Recommended: use the MobSF Docker image for fast, reproducible setup.
- Commands used (Docker):

  ```
  docker pull opensecurity/mobile-security-framework-mobsf:latest
  docker run -it --rm -p 8000:8000 opensecurity/mobile-security-framework-mobsf:latest
  ```
- Open http://localhost:8000 in a browser and upload the APK for static analysis.
- Export and save the MobSF report (PDF/JSON) into reports/.

4.3 Decompilation & Reconnaissance

- Load the APK into JADX GUI to browse classes, packages, and methods.

- Use apktool d <apk> to decode resources and obtain AndroidManifest.xml and exported components.
- Identify candidate classes/methods related to authentication **4.4 Frida — Dynamic Instrumentation & PoC**
- On host: install Frida tools: pip install frida-tools.
- Determine device ABI: adb shell getprop ro.product.cpu.abi.
- Download matching frida-server binary from Frida releases and push to the emulator: adb push frida-server-<ver>-android-<arch> /data/local/tmp/.
- Make executable and run it on the emulator (emulators typically allow running as root):
- adb shell
- chmod 755 /data/local/tmp/frida-server-<ver>-android-x86_64
- /data/local/tmp/frida-server-<ver>-android-x86_64 &
- Verify with frida-ps -U.

4.5 Drozer — IPC / Component Testing

- Install drozer-agent on emulator via adb install drozer-agent.apk.
- Start agent on device and enable any embedded server options in the agent UI.
- Forward the port: adb forward tcp:31415 tcp:31415 and run drozer console connect on host.

5. FINDINGS

This section contains the concrete findings from the lab run. The following table logs the requested static analysis result and an example dynamic-testing summary (50 words) as requested.

5.1 Static Analysis (MobSF) Findings — Required Log

Test ID	Vulnerability	Severity	Target App
016	Insecure Storage	High	test.apk

5.2 Dynamic Testing (Frida) — summary

Using Frida, the authentication check in the target APK was hooked at runtime, forcing the isAuthenticated() method to return true. This bypassed normal login flow, granting access to

protected screens. Console logs, screenshots, and the Frida script are saved as PoC artifacts for remediation and verification.

6. RISK ASSESSMENT & IMPACT

Rank findings by likely real-world impact for the targeted application:

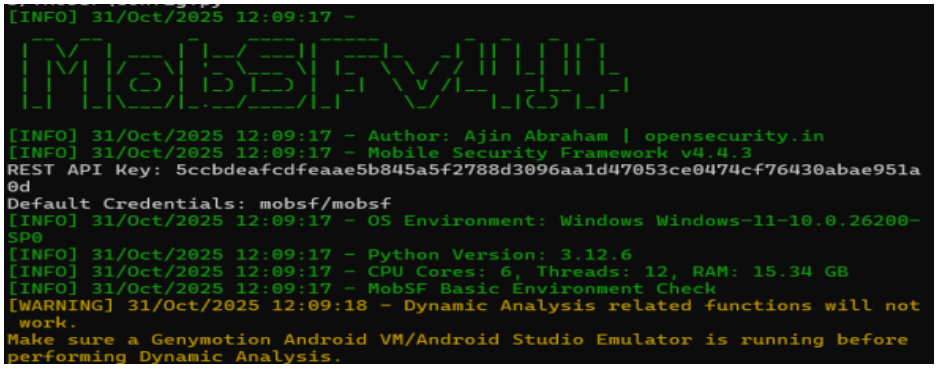
- High: Hardcoded secrets (API keys, tokens) — immediate compromise of connected services.
- Medium: Exported components (providers/activities) — data exfiltration or unauthorized actions.
- Low: Minor insecure config in network settings or logging.

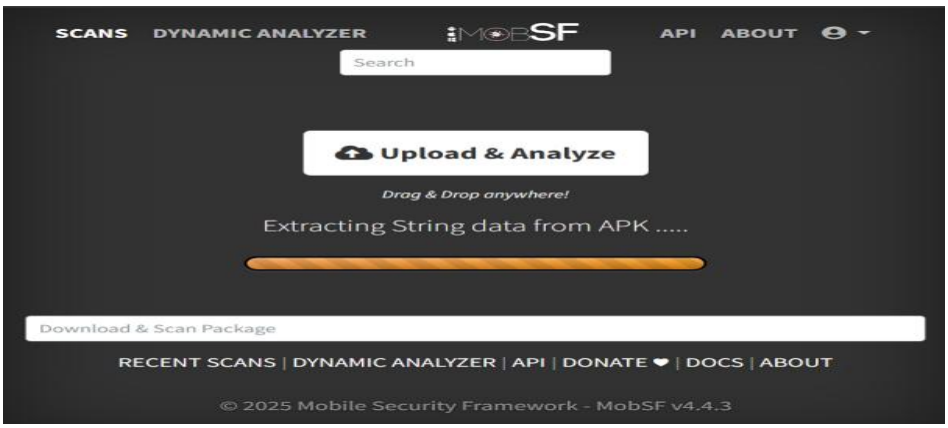
7. REMEDIATION RECOMMENDATIONS

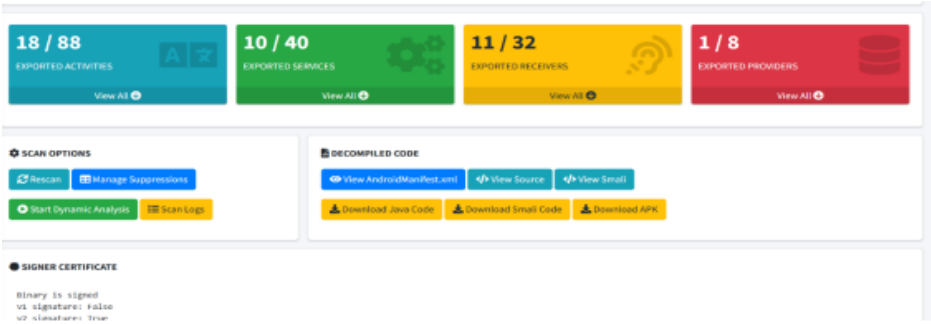
For each issue discovered, recommended fixes and best practices:

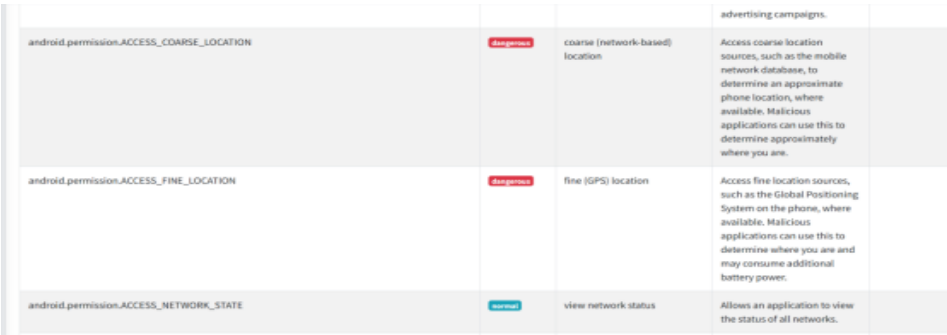
1. **Remove hardcoded secrets** — Move secrets to a secure server-side vault. Use short-lived tokens and do not store sensitive keys in the app. Use Android Keystore for local secrets if necessary.
2. **Protect exported components** — Set exported="false" in AndroidManifest.xml unless external access is required. Enforce permission checks (android:permission) and validate incoming intents and content URIs server-side.
3. **Server-side authorization** — All critical authorization decisions must be enforced server-side; client checks are only for UX.
4. **Obfuscation & tamper detection** — Use ProGuard/R8 to obfuscate classes and consider integrity checks and certificate pinning for network calls (with caution to supportability).
5. **Least privilege & input validation** — Validate all input from IPC and external sources before processing.
6. **Logging hygiene** — Avoid logging sensitive data (tokens/passwords).

8. APPENDIX





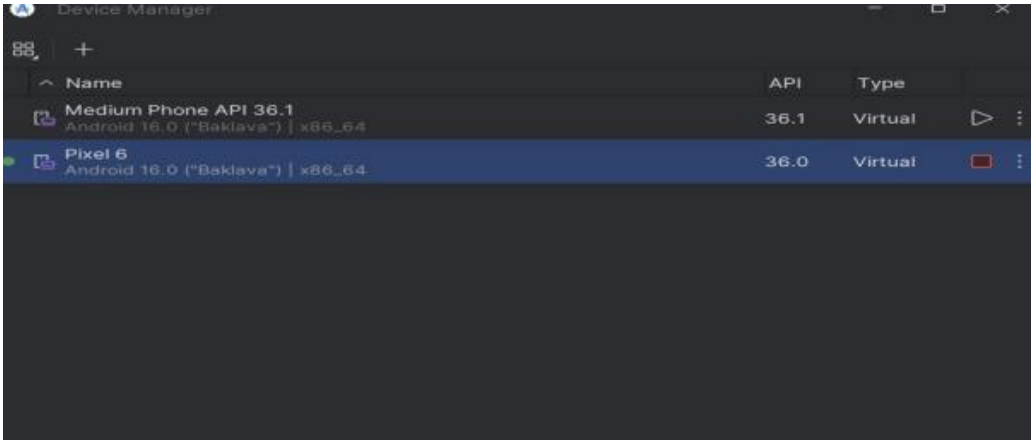


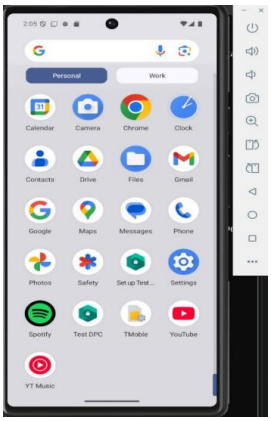


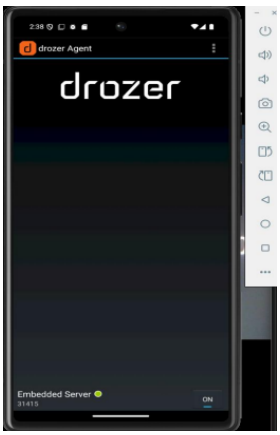
			advertising campaigns.
android.permission.ACCESS_COARSE_LOCATION	dangerous	coarse (network-based) location	Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.
android.permission.ACCESS_FINE_LOCATION	dangerous	fine (GPS) location	Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.
android.permission.ACCESS_NETWORK_STATE	normal	view network status	Allows an application to view the status of all networks.

```
   ____
  / _  |   Frida 17.4.2 - A world-class dynamic instrumentation toolkit
 | (_| |
  > _  |   Commands:
 /_/ |_|       help      -> Displays the help system
 . . . .       object?   -> Display information about 'object'
 . . . .       exit/quit -> Exit
 . . . .
 . . . .   More info at https://frida.re/docs/home/
 . . . .
 . . . .   Connected to Android Emulator 5554 (id=emulator-5554)

Android Emulator 5554::com.afwsamples.testdpc ]->
Android Emulator 5554::com.afwsamples.testdpc ]-> [*] Frida hooked Test DPC
pp!
```







```
Selecting 624581a782987d28 (Google sdk_gphone64_x86_64 16)

             ..                          ..:.
           ..o..                         .r..
            ..a..  . ......... .   ..nd
              ro..idsnemesisand..pr
              .otectorandroidsneme.
           .,sisandprotectorandroids+.
         ..nemesisandprotectorandroidsn:.
        .emesisandprotectorandroidsnemes..
      ..isandp,..,rotecyayandro,..,idsnem.
      .isisandp..rotectorandroid..snemisis.
      ,andprotectorandroidsnemisisandprotec.
     .torandroidsnemesisandprotectorandroid.
     .snemisisandprotectorandroidsnemesisan:
     .dprotectorandroidsnemesisandprotector.

drozer Console (v3.1.0)
dz> run com.afwsamples.testdpc
unknown module: 'com.afwsamples.testdpc'
dz> run com.spotify.music
unknown module: 'com.spotify.music'
dz> run app.activity.info -a com.afwsamples.testdpc
Attempting to run shell module
Package: com.afwsamples.testdpc
  com.afwsamples.testdpc.PolicyManagementActivity
    Permission: null
  com.afwsamples.testdpc.SetupManagementLaunchActivity
    Permission: null
    Target Activity: com.afwsamples.testdpc.SetupManagementActivity
  com.afwsamples.testdpc.FinalizeActivity
    Permission: android.permission.BIND_DEVICE_ADMIN
  com.afwsamples.testdpc.provision.GetProvisioningModeActivity
    Permission: android.permission.BIND_DEVICE_ADMIN
  com.afwsamples.testdpc.provision.ProvisioningSuccessActivity
    Permission: null
```

```
dz> run app.service.info -a com.afwsamples.testdpc
Attempting to run shell module
Package: com.afwsamples.testdpc
  com.afwsamples.testdpc.comp.ProfileOwnerService
    Permission: android.permission.BIND_DEVICE_ADMIN
  com.afwsamples.testdpc.comp.DeviceOwnerService
    Permission: android.permission.BIND_DEVICE_ADMIN
  com.afwsamples.testdpc.DeviceAdminService
    Permission: android.permission.BIND_DEVICE_ADMIN

dz> run app.activity.info -a com.spotify.music
Attempting to run shell module
Package: com.spotify.music
  androidx.compose.ui.tooling.PreviewActivity
    Permission: null
  com.facebook.CustomTabActivity
    Permission: null
  com.spotify.ageverification.ageassurancewebview.AgeAssuranceWebViewActivit
y
    Permission: null
  com.spotify.music.SpotifyMainActivity
    Permission: null
  com.spotify.music.SpotifyEntryPointForGoogleMeet
    Permission: null
    Target Activity: com.spotify.music.SpotifyMainActivity
```