



Mobile Application Testing Lab

Intern: RadhaSingh

Assessment date: 27/2/2026

Environment: Kali VM (192.168.75.128) attacking Metasploitable2 VM (192.168.75.129)

1. Executive Summary

This lab focused on identifying security weaknesses in an Android application using industry-standard mobile security tools. Static and dynamic analysis techniques were applied to detect insecure data storage, authentication weaknesses, and IPC misconfigurations. Critical findings reveal that sensitive information is stored insecurely and authentication logic can be bypassed at runtime, posing a high risk of data compromise and unauthorized access.

2. Tools and Methodology

Tool Purpose

MobSF Static APK analysis and security assessment

Frida Runtime instrumentation and function hooking

Drozer IPC, component, and permission testing

The assessment followed **OWASP Mobile Top 10** methodology, combining static, dynamic, and runtime analysis.

3. Static Analysis – MobSF

The screenshot shows the MobSF static analysis interface for the APK file 'diva-beta.apk'. The interface is divided into several sections:

- File Information:** Includes details like Name: diva-beta.apk, Size: 1.43MB, MD5, SHA1, SHA256, and a QR code.
- App Information:** Shows package name: jakhar.aseem.diva, main activity: Jakhar.aseem.diva.MainActivity, target SDK: 23, min SDK: 15, max SDK: 1, and other version details.
- Summary Metrics:** Displays counts for Activities (17), Services (0), Receivers (0), Providers (1), Exported Activities (2), Exported Services (0), Exported Receivers (0), and Exported Providers (1).
- Code Nature:** Lists native (True), dynamic (False), reflection (True), crypto (True), and obfuscation (False) status.
- Options:** Includes links to view Java, download Java code, view Smali, download Smali code, rescan, view AndroidManifest.xml, and start dynamic analysis.



The screenshot shows the MobSF interface for the 'diva-beta.apk' file. The left sidebar includes options like 'Information', 'Scan Options', 'Signer Certificate', 'Permissions', 'Binary Analysis', 'Android API', 'Browsable Activities', 'Security Analysis', 'Malware Analysis', 'Reconnaissance', 'Components', 'PDF Report', 'Print Report', and 'Start Dynamic Analysis'. The main content area has tabs for 'RECENT SCANS', 'STATIC ANALYZER', 'DYNAMIC ANALYZER', 'API DOCS', and 'ABOUT'. A search bar at the top right says 'Search MDS'. The 'APP SCORES' section shows an average CVSS of 6.1 and a security score of 55/100. The 'FILE INFORMATION' section lists the file name as 'diva-beta.apk', size as 1.43MB, MD5 as '82abb82193b3cfb1c737e3a786be363a', SHA1 as '27e849d9d7b86a3a3357fb3e980433a91d416801', SHA256 as '5cefc51fce9bd760b92ab2340477f4dda84b4ae0c5d04a8c9493e4fe34fab7c5', and SHA512 as '5cefc51fce9bd760b92ab2340477f4dda84b4ae0c5d04a8c9493e4fe34fab7c5'. The 'APP INFORMATION' section provides details about the app: App Name (Diva), Package Name (jakhar.aseem.diva), Main Activity (jakhar.aseem.diva.MainActivity), Target SDK (23), Min SDK (15), Max SDK (1), and Android Version Name (1.0). Below this, there are four cards: 'ACTIVITIES' (17), 'SERVICES' (0), 'RECEIVERS' (0), and 'PROVIDERS' (1). The 'SCAN OPTIONS' section includes 'Rescan' and 'Start Dynamic Analysis' buttons. The 'DECOMPILED CODE' section offers links to view AndroidManifest.xml, Java code, Smali code, and download APK.

3.1 Objective

Identify insecure coding practices and data storage flaws without executing the application.

3.2 Procedure

1. MobSF server was started on Kali Linux.
2. test.apk was uploaded through the MobSF web interface.
3. Static analysis was executed automatically.
4. Security findings were reviewed under **Code Analysis** and **Storage** sections.

3.3 Key Finding

Test ID Vulnerability Severity Target App

016 Insecure Storage High test.apk

3.4 Technical Analysis

MobSF detected that the application stores sensitive data (authentication tokens and user preferences) in **unencrypted SharedPreferences**. This data can be extracted on rooted devices or via ADB backup, enabling session hijacking and data leakage.

4. Dynamic Analysis – Frida Runtime Hooking

The screenshot shows a Windows PowerShell window titled 'Windows PowerShell' with the command 'frida -U -N b3nac.injuredandroid -l .\scripts-frida\bypass.js' entered. The output shows the Frida toolkit version (16.5.1), available commands (help, object?, exit/quit), and connection information ('Connected to Android Emulator 5554 (id=emulator-5554)'). A red box highlights the message '[Android Emulator 5554:b3nac.injuredandroid] -> Method a() hooked'. To the right, a red box contains the message 'Congrats you found the flag! :D'.



```
(kali㉿kali)-[~/Downloads]
$ frida-ps
PID  Name
1697 Thunar
10709 adb
1824 agent
1614 at-spi-bus-launcher
1632 at-spi2-registryd
1525 dbus-daemon
1621 dbus-daemon
1688 dconf-service
2121 firefox-esr
2226 firefox-esr
2323 firefox-esr
2401 firefox-esr
2487 firefox-esr
2505 firefox-esr
2508 firefox-esr
2561 firefox-esr
2786 firefox-esr
1526 gnome-keyring-daemon
1967 gvfs-afc-volume-monitor
2013 gvfs-goa-volume-monitor
2003 gvfs-gphoto2-volume-monitor
1937 gvfs-mtp-volume-monitor
1891 gvfs-udisks2-volume-monitor
1653 gvfsd
1659 gvfsd-fuse
```

4.1 Objective

Manipulate application logic at runtime to bypass authentication mechanisms.

4.2 Procedure

1. Frida server was deployed on the Android environment.
2. The target app was launched.
3. Java methods responsible for login validation were identified.
4. Frida hooks were injected to override authentication checks.

4.3 Result

Authentication checks were successfully bypassed by forcing the login function to always return a valid response, granting unauthorized access without valid credentials.

4.4 Dynamic Testing Summary

Using Frida, runtime function hooking was performed on the target Android application.

Authentication-related Java methods were intercepted and modified to bypass login validation. This allowed unauthorized access without credentials, demonstrating weak client-side authentication controls and highlighting the risks of relying solely on application-side security logic.



5. IPC and Component Testing – Drozer

```
Kali[Kali]:~$ drozer console connect --server 192.168.56.103
Selecting d278838d23b6b7b0 (innotek GmbH VirtualBox 8.1.0)

..:.
..o..
..a.. . ..nd
ro..idsnemesisand..pr
.otectionandnemisname.
.sisandprotectorandrods+.
..nemesisandprotectorandrodsn:.
.emesisandprotectorandrodnemis..
..isandp..,rotectorandro,..,idsnem.
.isisandp..rotectorandroid..snemisis.
,ndprotectionandrodnemesisandprotec.
.torandrodnemesisandprotectorandro.
.snmisisandprotectorandrodnemesisan.
.dprotectorandrodnemesisandprotector.

drozer Console (v2.4.4)
dz> run scanner.activity.browsable
Package: com.android.cts.priv.ctsshim
  Invocable URIs:
    Classes:
      .InstallPriority

Package: com.google.android.googlequicksearchbox
  Invocable URIs:
    assistant-handoff://complete
    ://assistant.google.com
    assistant-settings://
    com.google.android.apps.gsa.gdi://oauth2redirect (PATTERN_LITERAL)
    android-app://
    agsa://search/ (PATTERN_LITERAL)
    googlequicksearchbox://
    googleassistant://android/app/link/promo/tooltip (PATTERN_LITERAL)
    dynact://
  Classes:
    com.google.android.apps.gsa.assistant.handoff.BrowserReturnActivity
    com.google.android.apps.gsa.assistant.settings.hq.agentdirectory.AgentDirectoryActivity
    com.google.android.apps.gsa.assistant.settings.AssistantSettingsActivity
    net.openid.apauth.RedirectUriReceiverActivity
    com.google.android.search.calypso.AppIndexingActivity
    com.google.android.googlequicksearchbox.SearchActivity
    com.google.android.apps.gsa.staticplugins.opa.promo.UpgradePromoTooltipActivity
    com.google.android.apps.gsa.velour.DynamicActivityTrampoline
    com.google.android.apps.gsa.velvet.ui.VelvetIntentDispatcher

Package: com.android.vending
  Invocable URIs:
    http://
```

5.1 Objective

Evaluate exposed components and inter-process communication vulnerabilities.

5.2 Procedure

1. Drozer agent was installed on the test device.
2. Application package information was enumerated.
3. Exported activities, services, and broadcast receivers were analyzed.
4. Permission enforcement was validated.

5.3 Observations

Several components were found exported without adequate permission checks, enabling potential intent injection and unauthorized access to internal application functionality.

6. Findings Summary Table

Finding ID	Vulnerability	Severity	Impact
F016	Insecure Storage	High	Sensitive data extraction
F017	Authentication Bypass	High	Unauthorized access
F018	Exposed Components	Medium	Intent injection



7. Remediation Recommendations

1. Encrypt all sensitive data using Android Keystore APIs.
 2. Move authentication logic to server-side validation.
 3. Implement root and tamper detection.
 4. Restrict exported components using explicit permissions.
 5. Apply ProGuard/R8 to obfuscate sensitive logic.
-

9. Conclusion

The Mobile Application Testing Lab demonstrated how improper client-side security controls can be exploited using widely available tools. Insecure storage, weak authentication logic, and exposed components collectively increase the attack surface. Addressing these issues is essential to prevent data leakage and unauthorized access in real-world mobile applications.