P E N E T R A T I O N T E S T I N G

Android Application
Security Assessment Report

Client	**FinTrack Mobile Ltd.**
Target	Android App v4.1.2 (com.client.app)
Assessment Period	2 June 2026 – 13 June 2026
Report Date	22 June 2026
Assessment Type	Grey-box Android Application Penetration Test
Prepared By	**Devansh Gandhi**
Classification	**CONFIDENTIAL — Authorised Recipients Only**
Version	1.0 — Final

devansgandhi.com | info@devansgandhi.com

Table of Contents

1. Executive Summary

This report presents the findings of a grey-box Android application penetration test conducted against the FinTrack Mobile Android App (v4.1.2) between 2 June 2026 and 13 June 2026. Testing followed the OWASP Mobile Security Testing Guide (MSTG) v2.0 and OWASP Mobile Top 10 2024.

The assessment identified 9 findings across five severity levels. The most critical issues are the presence of hardcoded production API keys in the APK binary (accessible to anyone who downloads the app) and an exported Activity that allows any co-installed app to hijack the payment confirmation flow.

1.1 Findings Overview

CRITICAL	HIGH	MEDIUM	LOW	INFO	TOTAL
0	4	2	2	1	9
Findings	Findings	Findings	Findings	Finding	Findings

1.2 Severity Distribution

Severity	Distribution	#	%
Critical		0	0%
High		4	44%
Medium		2	22%
Low		2	22%
Info		1	11%

The overall mobile security posture is assessed as HIGH RISK. The hardcoded Razorpay production key (AND-002) and the unauthenticated exported PaymentConfirmActivity (AND-003) require immediate remediation before any further distribution of the application. Additionally, the combination of insecure local storage (AND-001) and absent certificate pinning (AND-004) means a device compromise or network interception leads to complete account takeover.

1.3 Key Recommendations

Priority order:

- Immediate (P0): Revoke and rotate hardcoded API keys (AND-002) and set android:exported="false" on PaymentConfirmActivity (AND-003).
- High (30 days): Implement EncryptedSharedPreferences + SQLCipher for local storage (AND-001) and add OkHttp certificate pinning (AND-004).
- Medium (60 days): Strip Logcat logging from release builds (AND-005) and replace MD5 with HMAC-SHA256 (AND-006).
- Low / Info (90 days): Disable ADB backup (AND-007), set debuggable false (AND-008), and add FLAG_SECURE to financial screens (AND-009).

2. Scope & Assessment Details

2.1 Engagement Scope

Item	Details
Target Application	FinTrack Mobile Android App v4.1.2 (com.fintrack.app)
APK Source	Provided by client (debug + release builds)
Android API Levels	minSdk: 21 (Android 5.0) | targetSdk: 34 (Android 14)
Test Type	Grey-box — APK + partial source code provided
Test Devices	Pixel 6 (Android 13) — rooted | Samsung Galaxy A52 (Android 12) — stock
Out of Scope	Backend API infrastructure, third-party SDKs (Razorpay, Firebase)
Environment	Staging backend; test accounts provisioned by client
Assessment Period	2 June 2026 – 13 June 2026 (10 business days)
Report Date	22 June 2026
Assessor	Devansh Gandhi | devansgandhi.com
Methodology	OWASP Mobile Security Testing Guide (MSTG) v2.0, OWASP Mobile Top 10 2024

2.2 Testing Methodology

Testing was conducted across the OWASP Mobile Top 10 2024 categories using both static and dynamic analysis:

- Static Analysis: APK decompilation (jadx, apktool), manifest review, hardcoded secret search, code pattern analysis.
- Dynamic Analysis: Runtime testing on rooted and stock devices, Frida instrumentation, Burp Suite traffic interception.
- Network Security: SSL/TLS configuration, certificate pinning, traffic interception on untrusted networks.
- Data Storage: SharedPreferences, SQLite, external storage, Logcat, backup extraction.
- Inter-App Communication: Intent handling, exported components, deep link validation.
- Cryptography: Algorithm selection, key storage, IV reuse, randomness quality.

3. Severity Rating Scale

Each finding is rated using CVSS v3.1. Mobile-specific attack vectors (AV:L for local device access, AV:P for physical access) are used where appropriate.

Severity	CVSS Range	SLA	Description
CRITICAL	9.0–10.0	Immediate	Remote exploitation, critical data/system impact. P0 incident.
HIGH	7.0–8.9	30 days	High exploitability with significant data or business impact. Escalate.
MEDIUM	4.0–6.9	60 days	Exploitable under specific conditions; moderate impact. Next sprint.
LOW	0.1–3.9	90 days	Limited exploitability requiring physical access or privilege. Quarterly cycle.
INFO	N/A	Best effort	No direct security impact; hardening recommendation.

4. Findings Summary

All 9 findings are summarised below. Full technical detail appears in Section 5.

ID	Title	Severity	CVSS	Location	Status
AND-001	Insecure Data Storage — Sensitive Data in Shared Preferences (Plaintext)	HIGH	7.1	com.client.app/shared_prefs/user_prefs.xml | SQLite: /data/data/com.client.app/databases/app.db	Open
AND-002	Hardcoded API Keys and Secrets in APK Binary	HIGH	8.2	com/client/app/network/ApiConfig.class | res/values/strings.xml	Open
AND-003	Exported Activity Without Permission Check — Deep Link Hijacking	HIGH	7.5	AndroidManifest.xml — com.client.app.ui.PaymentConfirmActivity	Open
AND-004	SSL/TLS Certificate Pinning Not Implemented	HIGH	7.4	com/client/app/network/RetrofitClient.java — OkHttpClient configuration	Open
AND-005	Sensitive Data Logged to Logcat in Production Build	MEDIUM	5.5	com/client/app/auth/LoginViewModel.java | com/client/app/network/ApiInterceptor.java	Open
AND-006	Weak Cryptography — MD5 Used for Local Cache Integrity Check	MEDIUM	5.9	com/client/app/cache/CacheManager.java	Open
AND-007	android:allowBackup="true" — ADB Backup Exposes App Data	LOW	3.7	AndroidManifest.xml — application element	Open
AND-008	Debuggable Flag Enabled in Release Build	LOW	3.1	AndroidManifest.xml — android:debuggable="true" | build.gradle	Open
AND-009	Screenshot Prevention Not Enforced on Sensitive Screens	INFO	N/A	com/client/app/ui/PaymentConfirmActivity.java | WalletFragment.java	Open

5. Detailed Findings

Each finding includes description, business impact, exploitation steps, evidence, and remediation guidance.

AND-001: Insecure Data Storage — Sensitive Data in Shared Preferences (Plaintext)

Severity	CVSS v3.1 Score
HIGH	**7.1 / 10.0**
Location	CVSS Vector
com.client.app/shared_prefs/user_prefs.xml | SQLite: /data/data/com.client.app/databases/app.db	AV:A/V:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Description

Authentication tokens, user PII (name, email, phone), and the session JWT are stored unencrypted in SharedPreferences XML and in the application SQLite database. Any application with READ_EXTERNAL_STORAGE permission or physical access to the device can read these values without root on Android < 10.

Business Impact

Session hijacking and full account takeover without network access. On a rooted or compromised device, a malicious application co-installed by the user can silently steal authentication tokens and user data. Constitutes a reportable data breach under DPDP if device is lost or compromised.

Steps to Reproduce

1. Install the APK on a rooted Android device or emulator.
2. Authenticate as a test user.
3. Pull the SharedPreferences file: adb pull /data/data/com.client.app/shared_prefs/user_prefs.xml
4. Open the file — auth_token, user_email, phone_number, and session_jwt are visible in plaintext.
5. Pull the SQLite DB: adb pull /data/data/com.client.app/databases/app.db
6. Open with DB Browser for SQLite — users table shows plaintext credentials and order history.

Evidence

```
Contents of user_prefs.xml (extracted via adb):
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<map>
  <string name="auth_token">eyJhbGciOiJIUzI1NiJ9.eyJpZCI6NDIxM...</string>
  <string name="user_email">alice@example.com</string>
  <string name="phone_number">+91-98765-43210</string>
  <string name="session_jwt">eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOi...</string>
  <string name="refresh_token">rt_4kLmN9pQx...</string>
</map>
```

Remediation

- Use Android Keystore + EncryptedSharedPreferences (Jetpack Security) for all sensitive values.
- Store session tokens in memory only; never persist JWTs to disk.
- Use SQLCipher to encrypt the SQLite database at rest.
- On logout, explicitly delete all stored credentials and clear the preferences file.
- Apply android:allowBackup="false" to prevent ADB backup extraction.

AND-002: Hardcoded API Keys and Secrets in APK Binary

Severity	CVSS v3.1 Score
HIGH	8.2 / 10.0
Location	**CVSS Vector**
com/client/app/network/ApiConfig.class | res/values/strings.xml	AV:A/V:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N

Description

Decompiling the APK with jadx reveals hardcoded production API keys for Firebase, Razorpay, and an internal backend service token embedded in the compiled Java classes and string resources. These keys are accessible to anyone who downloads the APK.

Business Impact

The exposed Razorpay key could be used to initiate payment operations, read transaction history, or generate payment links on behalf of the merchant. The Firebase key gives read/write access to the Firebase project if Firestore security rules are permissive. The backend service token bypasses API gateway authentication.

Steps to Reproduce

1. Download the APK: adb shell pm path com.client.app → adb pull <path>/base.apk
2. Decompile with jadx: jadx -d output/ base.apk
3. Search for secrets: grep -r "rzp_live\|AAAA\|Bearer" output/sources/
4. Locate com/client/app/network/ApiConfig.java — hardcoded keys visible.
5. Verify Razorpay key: curl -u rzp_live_KEY: https://api.razorpay.com/v1/payments

Evidence

```
Extracted from ApiConfig.java (decompiled via jadx):
public class ApiConfig {
    public static final String RAZORPAY_KEY   = "rzp_live_K9mXpQ3nWv1aZd";
    public static final String FIREBASE_KEY   =
"AIzaSyB8kLpX2mN4vQr7cW0fHsYjTuE9dRo3gIx";
    public static final String BACKEND_TOKEN  = "Bearer
svc_7fGhJ2kLmN9pQx4rStUv8wXyZaB3cD5e";
    public static final String BASE_URL       = "https://api.[TARGET].com";
}
```

Remediation

- Remove all hardcoded secrets from source code immediately; revoke and rotate all exposed keys.
- Use Android Keystore or remote config services (Firebase Remote Config, AWS Secrets Manager) to deliver secrets at runtime.
- For client-facing API keys (Razorpay, Firebase), use server-side proxying: the app calls your backend, which calls the payment provider — the payment key never reaches the client.
- Add pre-commit hooks (git-secrets, gitleaks) and CI/CD scanning (Semgrep, truffleHog) to prevent future leakage.
- Configure Razorpay key restrictions (domain/IP allowlist) as a defence-in-depth measure.

AND-003: Exported Activity Without Permission Check — Deep Link Hijacking

Severity	CVSS v3.1 Score
HIGH	7.5 / 10.0
Location `AndroidManifest.xml — com.client.app.ui.PaymentConfirmActivity`	CVSS Vector `AV:A/V:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N`

Description

PaymentConfirmActivity is exported with android:exported="true" and handles payment deep links without validating the calling package or requiring any permission. Any third-party application installed on the device can launch this activity directly with crafted Intent extras, bypassing the normal payment flow.

Business Impact

A malicious app co-installed on the device can launch the payment confirmation screen with forged payment data, potentially tricking the user into confirming a fraudulent transaction or leaking the transaction receipt displayed on the confirmation screen.

Steps to Reproduce

1. Install a third-party malicious app on the same device.
2. Use adb to launch the exported activity with crafted extras: adb shell am start -n com.client.app/.ui.PaymentConfirmActivity --es orderId "99999" --es amount "1" --es status "SUCCESS"
3. The PaymentConfirmActivity launches and displays a success screen for a fraudulent order.
4. Confirm: no permission check, no package verification — any installed app can trigger this.

Evidence

```
AndroidManifest.xml (extracted via apktool):
<activity
    android:name=".ui.PaymentConfirmActivity"
    android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="clientapp" android:host="payment"/>
  </intent-filter>
</activity>

// No android:permission attribute — any app can launch this activity
```

Remediation

- Set android:exported="false" on all activities that do not require external access.
- For activities that must handle deep links, add android:permission="com.client.app.PAYMENT_PERMISSION" and verify the calling package signature.
- Validate all Intent extras server-side before processing payment data — never trust client-side payment status.
- Use App Links (HTTPS deep links with Digital Asset Links verification) instead of custom URI schemes.
- Conduct a full audit of all exported components in AndroidManifest.xml.

AND-004: SSL/TLS Certificate Pinning Not Implemented

Severity	CVSS v3.1 Score
HIGH	7.4 / 10.0
Location	CVSS Vector
com/client/app/network/RetrofitClient.java — OkHttpClient configuration	AV:A/V:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N

Description

The application does not implement SSL/TLS certificate pinning. On a network where the attacker can present a custom CA certificate (corporate proxy, public Wi-Fi with custom CA installation), all API traffic can be intercepted and decrypted without any device compromise.

Business Impact

All API requests — including authentication tokens, user data, and payment details — can be intercepted in real-time on untrusted networks. An attacker on the same Wi-Fi network who persuades the user to install their CA certificate (phishing) can conduct a silent man-in-the-middle attack.

Steps to Reproduce

1. Set up Burp Suite with its CA certificate installed on the test device.
2. Route device traffic through Burp proxy.
3. Launch the app and log in — all API requests appear in Burp in plaintext.
4. Modify a response (e.g., change account balance) and forward — the app accepts the modified data.

Evidence

```
RetrofitClient.java (decompiled — no pin configuration):
public static OkHttpClient buildClient() {
    return new OkHttpClient.Builder()
        .connectTimeout(30, TimeUnit.SECONDS)
        .readTimeout(30, TimeUnit.SECONDS)
        // No CertificatePinner configured
        // No custom TrustManager
        .build();
}

// Burp Suite intercept — full JWT visible:
GET /api/v2/users/me HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJpZCI6NDIxM...
```

Remediation

- Implement OkHttp CertificatePinner pinning the leaf or intermediate certificate for api.[TARGET].com.
- Pin at least two certificates (primary + backup) to prevent service outage on certificate rotation.
- Use a Network Security Config (res/xml/network_security_config.xml) as an additional layer.
- Test pinning bypass resistance with Frida and SSL-Kill-Switch2 and address any identified bypasses.
- Implement a pin rotation mechanism via silent app update when certificates are due to expire.

AND-005: Sensitive Data Logged to Logcat in Production Build

Severity	CVSS v3.1 Score
MEDIUM	5.5 / 10.0
Location	CVSS Vector
com/client/app/auth/LoginViewModel.java | com/client/app/network/ApiInterceptor.java	AV:A/V:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

Description

The production APK contains Log.d() and Log.e() calls that write authentication tokens, full API responses (including user PII), and request headers (including Authorization: Bearer tokens) to Android Logcat. Any application with READ_LOGS permission (or a USB-connected adb session) can read these logs.

Business Impact

Session tokens readable from Logcat enable account takeover without any network access. On shared or enterprise devices, other applications with READ_LOGS permission can silently harvest tokens and PII.

Steps to Reproduce

1. Connect device via USB. Run: adb logcat | grep -i "token\|auth\|bearer\|response"
2. Authenticate in the app — JWT token, API response including user PII, and request headers printed to Logcat.

Evidence

```
adb logcat output during login (redacted):
D/LoginViewModel: Login response:
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJpZCI6NDI...","user":{"id":4213,"email":"alice@example.co
m","phone":"+91-98765-43210"}}
D/ApiInterceptor: --> POST /api/v2/auth/login
D/ApiInterceptor: Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJpZCI6NDI...
D/ApiInterceptor: <-- 200 OK (342ms)
```

Remediation

- Remove all Log.d/Log.v/Log.i calls that output sensitive data; use Log.e only for non-sensitive error conditions in production.
- Use BuildConfig.DEBUG guards: if (BuildConfig.DEBUG) { Log.d(TAG, sensitiveData); }
- Configure ProGuard/R8 rules to strip debug log calls from release builds.
- Use a structured logging library (Timber) configured with a no-op tree in production.

AND-006: Weak Cryptography — MD5 Used for Local Cache Integrity Check

Severity	CVSS v3.1 Score
MEDIUM	5.9 / 10.0
Location	CVSS Vector
com/client/app/cache/CacheManager.java	AV:A/V:L/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N

Description

The application uses MD5 to generate integrity hashes for locally cached API responses. MD5 is a cryptographically broken algorithm with known collision attacks. A locally privileged attacker can replace cached files with crafted content that produces the same MD5 hash.

Business Impact

Cache poisoning: a locally privileged attacker (rooted device, malicious co-installed app) can replace cached price data, product details, or user settings with arbitrary values while the integrity check passes, leading to financial discrepancies or misinformed user decisions.

Steps to Reproduce

1. Extract the cached files: adb pull /data/data/com.client.app/cache/
2. Identify the MD5 hash stored alongside a cached response.
3. Craft a replacement payload with the same MD5 hash using an MD5 collision tool.
4. Replace the cached file — the app accepts the tampered data as valid.

Evidence

```
CacheManager.java (decompiled):
private String computeHash(byte[] data) {
    try {
        MessageDigest md = MessageDigest.getInstance("MD5"); // Broken algorithm
        return Base64.encodeToString(md.digest(data), Base64.DEFAULT);
    } catch (NoSuchAlgorithmException e) {
        return "";
    }
}
```

Remediation

- Replace MD5 with SHA-256 for all integrity checking: MessageDigest.getInstance("SHA-256").
- Use HMAC-SHA256 with a secret derived from the Android Keystore for cache integrity validation.
- Consider encrypting cached sensitive data with EncryptedFile (Jetpack Security) rather than integrity-checking plaintext.

AND-007: android:allowBackup="true" — ADB Backup Exposes App Data

Severity	CVSS v3.1 Score
LOW	**3.7 / 10.0**
Location	CVSS Vector
`AndroidManifest.xml — application element`	`AV:A/V:P/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N`

Description

The application manifest does not set android:allowBackup="false". This means the entire application data directory — including databases, SharedPreferences, and cached files — can be extracted via adb backup without root access on any connected device.

Business Impact

Physical or USB access to the device (e.g., at a repair shop, border crossing, or from a borrowed charger) enables complete application data extraction, including any sensitive data stored locally. Severity is limited by the requirement for physical USB access.

Steps to Reproduce

1. Connect device via USB with USB debugging enabled.
2. Run: adb backup -f backup.ab -noapk com.client.app
3. Decrypt: java -jar abe.jar unpack backup.ab backup.tar
4. Extract: tar xf backup.tar — all app data including SharedPreferences visible.

Evidence

```
AndroidManifest.xml:
<application
    android:label="@string/app_name"
    android:icon="@mipmap/ic_launcher"
    android:theme="@style/AppTheme"
    <!-- android:allowBackup attribute absent — defaults to true on API < 31 -->
    android:supportsRtl="true">
```

Remediation

- Add android:allowBackup="false" to the <application> element in AndroidManifest.xml.
- For Android 12+, use the android:dataExtractionRules attribute to explicitly exclude sensitive data from backups.
- Remove USB debugging from all production device builds distributed to end users.

AND-008: Debuggable Flag Enabled in Release Build

Severity	CVSS v3.1 Score
LOW	**3.1 / 10.0**
Location	CVSS Vector
`AndroidManifest.xml — android:debuggable="true" | build.gradle`	`AV:A/V:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N`

Description

The production APK was built with android:debuggable="true", enabling attacker-controlled devices to attach a debugger to the running process using JDWP, inspect heap memory, set breakpoints, and bypass runtime checks.

Business Impact

With debuggable enabled, an attacker can attach Android Studio debugger or jdb to the live process, dump heap memory to extract in-memory secrets, bypass local authentication checks, and trace method calls to map the application's business logic.

Steps to Reproduce

1. Confirm: aapt dump badging base.apk | grep debuggable => application-debuggable
2. Attach debugger: adb jdwp → jdb -attach localhost:<port>
3. Set breakpoint on AuthViewModel.login() — inspect parameters and return values.
4. Dump heap: AndroidStudio → Profiler → Memory → Dump Heap — search for "token" or "password" in memory.

Evidence

```
build.gradle confirmation:
buildTypes {
    release {
        debuggable true  // Should be false
        minifyEnabled false
        signingConfig signingConfigs.release
    }
}
```

Remediation

- Set debuggable false in the release build type in build.gradle.
- Enable minifyEnabled true and shrinkResources true for release builds.
- Add a runtime debuggability check: if (Debug.isDebuggerConnected()) { System.exit(0); }
- Include debuggable checks in CI/CD pipeline to fail the build if debuggable is true in a release APK.

AND-009: Screenshot Prevention Not Enforced on Sensitive Screens

Severity	CVSS v3.1 Score
INFO	**N/A**
Location	CVSS Vector
com/client/app/ui/PaymentConfirmActivity.java | WalletFragment.java	N/A

Description

The payment confirmation screen and wallet balance screen do not set WindowManager.LayoutParams.FLAG_SECURE. This allows screenshots and screen recording of sensitive financial data, which may be captured by the OS Recent Apps thumbnail, accessible to other apps via Media Projection, or shared accidentally by the user.

Business Impact

Financial details (wallet balance, transaction history, card numbers) may appear in system screenshots, OS Recent Apps thumbnails, and Google Photos Backup. On compromised devices, screen-recording malware can capture these screens.

Steps to Reproduce

1. Navigate to the Wallet screen or Payment Confirmation screen.
2. Take a screenshot using the hardware buttons — screenshot is saved successfully to the gallery.
3. Confirm absence of FLAG_SECURE by checking Recent Apps — thumbnail shows financial data.

Evidence

```
PaymentConfirmActivity.java — FLAG SECURE absent:
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
    //     WindowManager.LayoutParams.FLAG_SECURE);  <-- commented out
    setContentView(R.layout.activity payment confirm);
}
```

Remediation

- Add getWindow().setFlags(FLAG_SECURE, FLAG_SECURE) in onCreate() for all Activities displaying financial data.
- Apply the flag to: PaymentConfirmActivity, WalletFragment, CardDetailsFragment, and any screen showing full card numbers or account balances.
- Test with Android's Recent Apps — the screen should appear as a blank/blurred thumbnail.

6. Remediation Roadmap

All findings ranked by priority with estimated effort and primary remediation action.

ID	Finding	Priority	Effort	Recommendation
AND-002	Hardcoded API Keys	Immediate	Medium	Revoke all keys; move to Android Keystore / server-side proxy.
AND-003	Exported Activity / Deep Link Hijack	Immediate	Low	Set android:exported="false"; validate all Intent extras server-side.
AND-001	Insecure Data Storage	High	Medium	EncryptedSharedPreferences + SQLCipher; clear on logout.
AND-004	No Certificate Pinning	High	Medium	Implement OkHttp CertificatePinner for api.[TARGET].com.
AND-005	Sensitive Data in Logcat	Medium	Low	Wrap Log calls in BuildConfig.DEBUG; strip with ProGuard.
AND-006	Weak Cryptography (MD5)	Medium	Low	Replace MD5 with HMAC-SHA256 using Keystore-backed key.
AND-007	ADB Backup Enabled	Low	Low	android:allowBackup="false" in AndroidManifest.xml.
AND-008	Debuggable Release Build	Low	Low	debuggable false in release buildType; minifyEnabled true.
AND-009	No Screenshot Prevention	Info	Low	Add FLAG_SECURE to financial screens in onCreate().

6.1 Retesting

Complimentary retest of all Critical and High findings included within 30 days of client completing remediation. Medium and lower on request.

7. Appendix

7.1 Tools Used

Tool	Purpose
jadx 1.5	APK decompilation to Java source code
apktool 2.9	APK unpacking for AndroidManifest.xml and resource inspection
Burp Suite Professional 2024	HTTP/S traffic interception and manipulation
Frida 16.3 + objection	Runtime instrumentation, method hooking, root/SSL-pinning bypass
MobSF (Mobile Security Framework)	Automated static and dynamic analysis
adb (Android Debug Bridge)	Device access, file extraction, Logcat monitoring
DB Browser for SQLite	SQLite database inspection
Android Backup Extractor (abe.jar)	ADB backup decryption and extraction

7.2 OWASP Mobile Top 10 2024 Coverage

Category	Findings
M1: Improper Credential Usage	AND-002 (hardcoded keys)
M2: Inadequate Supply Chain Security	Not identified
M3: Insecure Authentication / Authorization	AND-003 (exported activity)
M4: Insufficient Input/Output Validation	Not identified
M5: Insecure Communication	AND-004 (no cert pinning)
M6: Inadequate Privacy Controls	AND-001, AND-005 (storage, Logcat)
M7: Insufficient Binary Protections	AND-008 (debuggable), AND-002
M8: Security Misconfiguration	AND-007 (backup), AND-009 (screenshot)
M9: Insecure Data Storage	AND-001, AND-006
M10: Insufficient Cryptography	AND-006 (MD5)

7.3 Disclaimer

All testing was performed within the agreed scope with explicit written authorisation from FinTrack Mobile Ltd.. Testing used dedicated test accounts and anonymised staging data; no real user data was accessed or exfiltrated.

Devansh Gandhi

devansgandhi.com | info@devansgandhi.com