Android Application Security Assessment

Prepared for **Global Finance Inc**

Assessment Period: February 24, 2026 — February 27, 2026
Report Date: March 09, 2026

Version 1.0.0 | System Administrator

1. Table of Contents

2. Executive Summary

A comprehensive Android application security assessment was conducted on the FinanceGuard Mobile application (com.financeguard.mobile) for FinanceGuard Inc. The assessment evaluated the application's security posture against the OWASP Mobile Application Security Verification Standard (MASVS) and identified vulnerabilities through static analysis, dynamic analysis, network traffic inspection, and data storage review.

The assessment revealed several critical and high-severity findings that could allow an attacker to compromise sensitive financial data, bypass authentication mechanisms, and intercept communications between the application and backend services. Key concerns include insufficient certificate pinning implementation, insecure local data storage of authentication tokens, weak root detection mechanisms, and exposed debug interfaces.

It is strongly recommended that FinanceGuard Inc. prioritize the remediation of Critical and High severity findings immediately to protect customer financial data and maintain regulatory compliance.

This document is confidential and intended solely for Global Finance Inc. Distribution to unauthorized parties is prohibited. Findings are valid as of the assessment date and may not reflect subsequent changes.

2.1. Key Findings Overview

During the assessment period from **February 24, 2026** to **February 27, 2026**, a total of **3** vulnerabilities were identified in the FinanceGuard Mobile Android application of Global Finance Inc.

Severity	Count
Critical	0
High	1
Medium	2
Low	0
Informational	0

3. Application Profile & Attack Surface

The FinanceGuard Mobile application is an Android-based banking and financial management application developed for FinanceGuard Inc. customers. The application provides account management, fund transfers, bill payments, and investment portfolio tracking capabilities. The following tables detail the application's technical profile, permission usage, and exported component attack surface.

3.0.1. Application Information

Property	Value
Package Name	com.financeguard.mobile
Version Tested	4.2.1 (Build 421036)
Target SDK	API 34 (Android 14)
Minimum SDK	API 26 (Android 8.0 Oreo)
Architecture	arm64-v8a, armeabi-v7a
Obfuscation	ProGuard/R8 (partial — several classes unobfuscated)
Test Device	Google Pixel 7 — Android 14 (rooted via Magisk 26.4)

3.0.2. Dangerous Permissions

Permission	Risk Level	Justification
CAMERA	Medium	Used for check deposit — legitimate but grants camera access
READ_CONTACTS	High	Used for contact-based transfers — potential data exfiltration vector
ACCESS_FINE_LOCATION	High	ATM/branch locator — excessive precision for stated purpose
READ_PHONE_STATE	Medium	Device fingerprinting — leaks IMEI and phone number
WRITE_EXTERNAL_STORAGE	Critical	Statement export — data accessible to all apps on

3.0.3. Exported Components

Component	Type	Protection	Risk Assessment
.DeepLinkActivity	Activity	None	Accepts arbitrary URIs — potential for intent injection
.ShareReceiver	Activity	None	Handles shared content without validation
.SyncService	Service	Signature	Protected by signature permission — low risk
.TransactionProvider	Provider	None	Exposes transaction data via content:// URI queries

4. Assessment Methodology & Tools

The assessment followed a structured four-phase methodology aligned with the OWASP Mobile Application Security Testing Guide (MASTG) and the OWASP MASVS v2.0 framework. Each phase targets different layers of the application's security posture, from source code analysis through runtime behavior to network communications and data persistence.

4.1. Assessment Phases

Phase	Focus Area	Key Techniques	Primary Tools
1	**Static Analysis** Reverse engineering & code review	APK decompilation, manifest analysis, hardcoded secret scanning, dependency audit	jadx, apktool, MobSF, semgrep
2	**Dynamic Analysis** Runtime security testing	Root detection bypass, SSL pinning bypass, method hooking, auth flow testing	Frida, Objection, Magisk
3	**Network Analysis** Traffic interception & API testing	MITM proxy, TLS inspection, API endpoint fuzzing, certificate pinning validation	Burp Suite, nuclei, mitmproxy
4	**Data Storage** Local storage & crypto review	SharedPreferences extraction, SQLite analysis, file system review, Keystore audit	adb, Drozer, Objection

All testing was performed on a physical Google Pixel 7 device running Android 14 with Magisk 26.4 for systemless root access. The device was configured with Frida server 16.x for runtime instrumentation and Burp Suite CA certificate installed as a system-level trusted certificate for TLS interception. Network Security Configuration overrides were applied via apktool repackaging where certificate pinning prevented proxy-based analysis.

5. Findings Overview

The following table provides a summary of all identified vulnerabilities, sorted by severity and mapped to OWASP Mobile Top 10 categories. Each finding is detailed in the subsequent section.

#	Finding	MASVS Category	Severity	CVSS
1	SSL Pinning Bypass in Android Application	M5: Insecure Communication	High	7.5
2	Debug Mode Enabled in Production Android Application	M8: Security Misconfiguration	Medium	6.2
3	No Root Detection	M7: Insufficient Binary Protections	Medium	4.6

6. MASVS-STORAGE — Storage

Mobile applications handle a wide variety of sensitive data, such as personally identifiable information (PII), cryptographic material, secrets, and API keys, that often need to be stored locally. This sensitive data may be stored in private locations, such as the app\'s internal storage, or in public folders that are accessible by the user or other apps installed on the device. However, sensitive data can also be unintentionally stored or exposed to publicly accessible locations, typically as a side-effect of using certain APIs or system capabilities such as backups or logs.

This category is designed to help developers ensure that any sensitive data intentionally stored by the app is properly protected, regardless of the target location. It also covers unintentional leaks that can occur due to improper use of APIs or system capabilities.

7. MASVS-CRYPTO — Cryptography

Cryptography is essential for mobile apps because mobile devices are highly portable and can be easily lost or stolen. This means that an attacker who gains physical access to a device can potentially access all the sensitive data stored on it, including passwords, financial information, and personally identifiable information. Cryptography provides a means of protecting this sensitive data by encrypting it so that it cannot be easily read or accessed by an unauthorized user.

The purpose of the controls in this category is to ensure that the verified app uses cryptography according to industry best practices, which are typically defined in external standards such as NIST.SP.800-175B and NIST.SP.800-57. This category also focuses on the management of cryptographic keys throughout their lifecycle, including key generation, storage, and protection. Poor key management can compromise even the strongest cryptography, so it is crucial for developers to follow the recommended best practices.

8. MASVS-AUTH — Authentication and Authorization

Authentication and authorization are essential components of most mobile apps, especially those that connect to a remote service. These mechanisms provide an added layer of security and help prevent unauthorized access to sensitive user data. Although the enforcement of these mechanisms must be on the remote endpoint, it is equally important for the app to follow relevant best practices to ensure the secure use of the involved protocols.

Mobile apps often use different forms of authentication, such as biometrics, PIN, or multi-factor authentication code generators, to validate user identity. These mechanisms must be implemented correctly to ensure their effectiveness in preventing unauthorized access. Additionally, some apps may rely solely on local app authentication and may not have a remote endpoint. In such cases, it is critical to ensure that local authentication mechanisms are secure and implemented following industry best practices.

9. MASVS-NETWORK — Network Communication

Secure networking is a critical aspect of mobile app security, particularly for apps that communicate over the network. In order to ensure the confidentiality and integrity of data in transit, developers typically rely on encryption and authentication of the remote endpoint, such as through the use of TLS. However, there are numerous ways in which a developer may accidentally disable the platform secure defaults or bypass them entirely by utilizing low-level APIs or third-party libraries.

This category is designed to ensure that the mobile app sets up secure connections under any circumstances. Specifically, it focuses on verifying that the app establishes a secure, encrypted channel for network communication. Additionally, this category covers situations where a developer may choose to trust only specific Certificate Authorities (CAs), which is commonly referred to as certificate pinning or public key pinning.

9.1. SSL Pinning Bypass in Android Application

Severity	High	CVSS Score	7.5 (High)
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N		
Affected Component	Network Security Layer	MASVS Requirement	MASVS-NETWORK

9.1.1. Description

SSL Pinning is a security mechanism that ensures an Android application communicates only with a trusted server by verifying the server's SSL certificate against a predefined certificate or public key. If SSL Pinning is improperly implemented or bypassed, an attacker can intercept or manipulate the communication between the client and server, leading to man-in-the-middle (MITM) attacks.
This vulnerability occurs when the application fails to enforce strict certificate pinning, allowing an attacker to present a forged certificate or exploit weak pinning implementations to bypass the intended security controls.

9.1.2. Exploitation

The exploitation of this vulnerability typically involves the following steps:

- **Intercept Traffic:** An attacker uses a proxy tool (e.g., Burp Suite, mitmproxy) to intercept the application's network traffic.
- **Bypass SSL Pinning:** The attacker exploits weak or misconfigured SSL pinning to present a forged certificate or manipulate the pinning logic.
- **Exfiltrate Data:** Once the SSL pinning is bypassed, the attacker can intercept sensitive data transmitted between the client and server, such as authentication tokens, personal information, or financial data.

9.1.3. Impact

The impact of a successful SSL Pinning bypass can be severe, including:

- **Data Theft:** Sensitive user data, such as login credentials, personal information, or financial details, can be stolen.
- **Session Hijacking:** An attacker can hijack user sessions by stealing session tokens or cookies.
- **Man-in-the-Middle (MITM) Attacks:** The attacker can modify or inject malicious content into the communication stream.
- **Reputation Damage:** The application's security reputation may be compromised, leading to loss of user trust.

9.1.4. Remediation

To remediate this vulnerability, the following steps should be taken:

- **Implement Proper SSL Pinning:** Use a robust SSL pinning library (e.g., Certificate Pinning, Network Security Configuration) to enforce strict certificate validation.
- **Use Network Security Configuration:** Define a network security configuration in the AndroidManifest.xml to enforce certificate pinning and other security policies.
- **Regularly Update Certificates:** Ensure that pinned certificates are regularly updated and rotated to prevent expiration or compromise.
- **Test for Bypasses:** Conduct thorough testing to ensure that the SSL pinning implementation cannot be bypassed using common techniques (e.g., certificate substitution, dynamic analysis tools).

9.1.5. References

- Android Developer Guide: Security with HTTPS and Certificate Pinning
- OWASP Mobile Security Testing Guide: Testing for SSL/TLS Configuration
- MITRE CVE Database

10. MASVS-PLATFORM — Platform Interaction

The security of mobile apps heavily depends on their interaction with the mobile platform, which often involves exposing data or functionality intentionally through the use of platform-provided inter-process communication (IPC) mechanisms and WebViews to enhance the user experience. However, these mechanisms can also be exploited by attackers or other installed apps, potentially compromising the app\'s security.

Furthermore, sensitive data, such as passwords, credit card details, and one-time passwords in notifications, is often displayed in the app\'s user interface. It is essential to ensure that this data is not unintentionally leaked through platform mechanisms such as auto-generated screenshots or accidental disclosure through shoulder surfing or device sharing. This category comprises controls that ensure the app\'s interactions with the mobile platform occur securely.

10.1. No Root Detection

Severity	Medium	CVSS Score	4.6 (Medium)
CVSS Vector	CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N		
Affected Component	Root Detection Module	MASVS Requirement	MASVS-PLATFORM

10.1.1. Description

This vulnerability occurs when an Android application fails to properly detect if the device is rooted, allowing attackers to bypass security mechanisms that rely on root status checks. Root detection is crucial for enforcing security policies, as rooted devices can be manipulated to access sensitive data or execute privileged operations.

10.1.2. Exploitation

The vulnerability can be exploited by an attacker who has rooted the device. Since the application does not properly detect the root status, the attacker can use root privileges to:

- Bypass security checks
- Access sensitive data
- Execute privileged operations
- Modify application behavior

This can be achieved by using tools like `Magisk` or `SuperSU` to hide root status from the application.

10.1.3. Impact

The impact of this vulnerability includes:

- **Data Exposure:** Sensitive data stored or processed by the application may be accessed by unauthorized users.
- **Privilege Escalation:** Attackers can gain elevated privileges to perform actions that should be restricted.
- **Security Bypass:** Security mechanisms relying on root detection can be circumvented, leading to unauthorized access or manipulation.

10.1.4. Remediation

To mitigate this vulnerability, the following steps should be taken:

- **Implement Robust Root Detection:** Use multiple root detection methods, such as checking for root binaries, detecting Magisk, or verifying system integrity.
- **Use Secure Storage:** Store sensitive data in secure storage mechanisms that are resistant to root access.
- **Apply Code Obfuscation:** Obfuscate the root detection logic to make it harder for attackers to bypass.
- **Regular Updates:** Keep the application and its dependencies up to date to address known vulnerabilities.

10.1.5. References

For more information, refer to the following resources:

- Android Security Best Practices
- OWASP Mobile Security Testing Guide
- CVE Details

11. MASVS-CODE — Code Quality

Mobile apps have many data entry points, including the UI, IPC, network, and file system, which might receive data that has been inadvertently modified by untrusted actors. By treating this data as untrusted input and properly verifying and sanitizing it before use, developers can prevent classical injection attacks, such as SQL injection, XSS, or insecure deserialization. However, other common coding vulnerabilities, such as memory corruption flaws, are hard to detect in penetration testing but easy to prevent with secure architecture and coding practices.

This category covers coding vulnerabilities that arise from external sources such as app data entry points, the OS, and third-party software components. Developers should verify and sanitize all incoming data to prevent injection attacks and bypass of security checks. They should also enforce app updates and ensure that the app runs up-to-date platforms to protect users from known vulnerabilities.

11.1. Debug Mode Enabled in Production Android Application

Severity	Medium	CVSS Score	6.2 (Medium)
CVSS Vector	CVSS:3.1/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N		
Affected Component	Application Build Configuration	MASVS Requirement	MASVS-CODE

11.1.1. Description

Debugging features, such as verbose logging, debug flags, or development APIs, are inadvertently left enabled in the production build of an Android application. This can expose sensitive information, internal APIs, or debugging interfaces to attackers, potentially leading to reverse engineering, data leakage, or unauthorized access.

11.1.2. Exploitation

The vulnerability can be exploited by analyzing the application's behavior or network traffic to identify debug endpoints, logging mechanisms, or exposed APIs. Attackers may use tools like `adb logcat` to extract sensitive logs or interact with debug interfaces to manipulate application behavior.

11.1.3. Impact

The impact includes exposure of sensitive data, reverse engineering of application logic, unauthorized access to debug features, and potential privilege escalation. Attackers may also use debug interfaces to bypass security controls or exfiltrate data.

11.1.4. Remediation

- Ensure debug flags are disabled in production builds by setting `BuildConfig.DEBUG = false`.
- Remove or obfuscate debug logging statements in production code.
- Disable remote debugging and development APIs in production configurations.
- Use ProGuard or R8 to strip debug symbols and sensitive information from the release build.
- Implement runtime checks to disable debug features in production environments.

11.1.5. References

- Android Debugging Documentation
- OWASP MSTG: Testing for Improper Platform Usage

12. MASVS-RESILIENCE — Resilience Against Reverse Engineering and Tampering

Defense-in-depth measures such as code obfuscation, anti-debugging, anti-tampering, etc. are important to increase app resilience against reverse engineering and specific client-side attacks. They add multiple layers of security controls to the app, making it more difficult for attackers to successfully reverse engineer and extract valuable intellectual property or sensitive data from it.

The controls in this category aim to ensure that the app is running on a trusted platform, prevent tampering at runtime and ensure the integrity of the app\'s intended functionality. Additionally, the controls impede comprehension by making it difficult to figure out how the app works using static analysis and prevent dynamic analysis and instrumentation that could allow an attacker to modify the code at runtime.

13. MASVS-PRIVACY — Privacy

The main goal of MASVS-PRIVACY is to provide a baseline for user privacy. It is not intended to cover all aspects of user privacy, especially when other standards and regulations such as ENISA or the GDPR already do that. We focus on the app itself, looking at what can be tested using information that\'s publicly available or found within the app through methods like static or dynamic analysis.

MASVS-PRIVACY is not intended to serve as an exhaustive or exclusive reference. While it provides valuable guidance on app-centric privacy considerations, it should never replace comprehensive assessments, such as a Data Protection Impact Assessment (DPIA) mandated by the General Data Protection Regulation (GDPR) or other pertinent legal and regulatory frameworks.

14. Remediation Priority Matrix

The following priority matrix maps each identified vulnerability to its recommended remediation action, estimated implementation effort, and severity. Items should be addressed in priority order, focusing on Critical and High severity findings first. Effort estimates reflect typical mobile development cycles: Low (1-2 days), Medium (3-5 days), High (1-2 weeks), Very High (2+ weeks).

14.1. Prioritized Actions

Priority	Finding	Recommended Action	Effort	Severity
1	SSL Pinning Bypass in Android Application	To remediate this vulnerability, the following steps should be taken:Implement Proper SSL Pinning: Use a robust SSL p...	Medium	High
2	Debug Mode Enabled in Production Android Application	Ensure debug flags are disabled in production builds by setting BuildConfig.DEBUG = false.Remove or obfuscate debug l...	Low	Medium
3	No Root Detection	To mitigate this vulnerability, the following steps should be taken:Implement Robust Root Detection: Use multiple roo...	Medium	Medium

15. Appendix

15.1. Glossary

Term	Definition
APK	Android Package Kit — the file format used for distributing and installing Android applications
MASVS	Mobile Application Security Verification Standard — OWASP framework defining mobile application security requirements
MASTG	Mobile Application Security Testing Guide — OWASP comprehensive manual for mobile app security testing
Root Detection	Security mechanism to detect if an Android device has been rooted, potentially bypassing security controls
Certificate Pinning	Security technique that associates a host with its expected certificate or public key to prevent MITM attacks
Frida	Dynamic instrumentation toolkit for injecting JavaScript into native apps for runtime analysis and manipulation
IPC	Inter-Process Communication — mechanism for Android components to communicate with each other
CVSS	Common Vulnerability Scoring System — standardized framework for rating vulnerability severity
RASP	Runtime Application Self-Protection — security technology that detects and blocks attacks in real-time within the app
ProGuard/R8	Code shrinking and obfuscation tools for Android applications to protect against reverse engineering
Android Keystore	Hardware-backed cryptographic key storage system providing secure key generation and usage
Magisk	Systemless root solution for Android that modifies boot image without altering the system partition
Drozer	Android security assessment framework for testing exported components and IPC endpoints

15.2. Severity Rating Scale

Severity	CVSS Range	Description
Critical	9.0 – 10.0	Vulnerabilities that can be exploited trivially and lead to complete system compromise. Immediate remediation required.
High	7.0 – 8.9	Vulnerabilities that can lead to significant data exposure or system compromise with minimal complexity. Should be addressed within 30 days.
Medium	4.0 – 6.9	Vulnerabilities that require specific conditions to exploit but could lead to moderate impact. Should be addressed within 90 days.
Low	0.1 – 3.9	Vulnerabilities with limited impact and significant exploitation barriers. Should be addressed as part of regular security maintenance.
Info	N/A	Observations and best practice recommendations that do not represent an immediate security risk but improve overall security posture.