JARWIS

PENETRATION TEST REPORT

TARGET APPLICATION

com.example.mobileapp (v2.5.1)

Mobile Application Security Assessment

REPORT ID

JAR-20260104_194822-E5C1C0

ASSESSMENT DATE

January 04, 2026

ENDPOINTS TESTED

12

ASSESSMENT TYPE

Authenticated & Unauthenticated

Assessment Results

CRITICAL RISK

2 Critical	2 High	1 Medium	1 Low	0 Info

CONFIDENTIAL

Introduction

Jarwis is an AI-powered penetration testing platform that delivers comprehensive security assessments at scale. Leveraging advanced artificial intelligence and machine learning algorithms, Jarwis automates the discovery and verification of security vulnerabilities across web applications, mobile applications, APIs, and cloud infrastructure.

The platform combines the thoroughness of manual penetration testing with the speed and consistency of automated scanning, providing organizations with actionable security insights in hours rather than weeks. Jarwis systematically tests against the OWASP Top 10 and other industry-standard vulnerability classifications, delivering detailed findings with proof-of-concept evidence and remediation guidance.

> **Key Capabilities:** Automated vulnerability discovery • AI-powered verification to minimize false positives • OWASP Top 10 coverage • Detailed proof-of-concept evidence • Contextual remediation recommendations • Multi-format reporting (PDF, HTML, JSON, SARIF)

Executive Summary

Example Mobile Corp. engaged Jarwis to conduct a comprehensive penetration test of their mobile application security assessment environment to evaluate security vulnerabilities and assess potential risks to their digital infrastructure. During the time allotted for this engagement, Jarwis systematically tested the application's defense mechanisms, focusing on industry-standard attack vectors aligned with the OWASP Top Ten vulnerabilities.

Purpose

The penetration test was conducted to provide Example Mobile Corp. with a thorough security assessment of their mobile application security assessment infrastructure. Jarwis evaluated the application's resilience against common attack vectors including injection flaws, cross-site scripting, access control bypasses, authentication weaknesses, and information disclosure vulnerabilities. The assessment aimed to identify

exploitable security weaknesses that could compromise the confidentiality, integrity, and availability of Example Mobile Corp.'s systems and data.

Findings Overview

During the assessment, Jarwis identified **6 distinct security vulnerabilities** across the target environment. The findings include **2 critical-severity** vulnerabilities, **2 high-severity** vulnerabilities, **1 medium-severity** vulnerability, and **1 low-severity** vulnerability.

The **critical-severity finding** involves hardcoded api keys and secrets in apk. The mobile application contains hardcoded API keys, secrets, and credentials in the decompiled source code. These can be extracted by anyone who downloads the app from the Play Store, potentially comp

The **critical-severity finding** involves unencrypted sqlite database with sensitive data. The application stores sensitive user data including credentials and personal information in an unencrypted SQLite database on the device. Anyone with physical access or root privileges can extract th

The **high-severity finding** involves ssl certificate pinning not implemented. The application does not implement SSL certificate pinning, making it trivially vulnerable to man-in-the-middle attacks. Attackers on the same network can intercept and modify all API traffic.

The **high-severity finding** involves production app built with debuggable flag. The production application has android:debuggable=true in the manifest, allowing attackers to attach debuggers, set breakpoints, and inspect runtime memory including decrypted data and tokens.

> The critical and high-severity vulnerabilities pose immediate risks to the organization's security posture, as they provide pathways for unauthorized data access and system compromise. Immediate remediation is strongly recommended.

Findings Summary

The following table provides an overview of all vulnerabilities identified during this assessment, organized by severity level.

SEVERITY	VULNERABILITY	CATEGORY	CWE
CRITICAL	Hardcoded API Keys and Secrets in APK	Cryptographic Failures	CWE-798
CRITICAL	Unencrypted SQLite Database with Sensitive Data	Cryptographic Failures	CWE-312
HIGH	SSL Certificate Pinning Not Implemented	Security Misconfiguration	CWE-295
HIGH	Production App Built with Debuggable Flag	Insecure Design	CWE-489
MEDIUM	Sensitive Activities Exported Without Permission	Broken Access Control	CWE-926
LOW	Excessive Logging of Sensitive Information	Logging Failures	CWE-532

Conclusion

The penetration test revealed several significant security vulnerabilities that require immediate attention. The presence of 4 critical and high-severity vulnerabilities represents substantial risks that could lead to unauthorized system access and data compromise. By promptly addressing these identified vulnerabilities, the organization can significantly enhance their application security and reduce exposure to potential cyber threats.

Recommended Actions: Remediate 2 critical vulnerabilities within 24-48 hours • Address 2 high-severity findings within 7 days • Plan remediation of 1 medium-severity issues within 30 days •

Implement security monitoring and logging for affected components • Conduct follow-up assessment to verify remediation effectiveness

Detailed Findings

The following section provides comprehensive technical details for each vulnerability identified during the assessment, including proof-of-concept evidence, AI-powered analysis, and specific remediation guidance.

JAR-MOB-001

CRITICAL

Hardcoded API Keys and Secrets in APK

A02 CWE-798

DESCRIPTION

The mobile application contains hardcoded API keys, secrets, and credentials in the decompiled source code. These can be extracted by anyone who downloads the app from the Play Store, potentially compromising backend services and third-party integrations.

URL	METHOD	PARAMETER
com.example.mobileapp/BuildConfig.java	STATIC	API_KEY

EVIDENCE / PROOF OF CONCEPT

```
Found: public static final String API_KEY = "AIzaSyB4x5c7d8e9f0g1h2i3j4k5l6m7n8o9p0"; STRIPE_KEY = "sk_live_abc123"
```

AI Analysis

Jarwis AI performed static analysis on the APK and found hardcoded API keys including Google Maps, Stripe live key, and Firebase credentials in the BuildConfig class and strings.xml. These can be trivially extracted by any attacker.

HTTP DETAILS

REQUEST

```
Static Analysis: com.example.mobileapp_v2.5.1.apk
Tool: Androguard + Jarwis Mobile Scanner
```

RESPONSE

```
Extracted Secrets:
- API_KEY: AIzaSyB4x5c7d8e9f0g1h2i3j4k5l6m7n8o9p0
- STRIPE_KEY: sk_live_abc123
- FIREBASE_URL: https://app-prod.firebaseio.com
```

✓ Remediation

Never hardcode secrets in mobile apps. Use secure key storage (Android Keystore, iOS Keychain). Fetch secrets from a secure backend at runtime. Restrict API key permissions and implement key

rotation.

JAR-MOB-002

CRITICAL

Unencrypted SQLite Database with Sensitive Data

A02 CWE-312

DESCRIPTION

The application stores sensitive user data including credentials and personal information in an unencrypted SQLite database on the device. Anyone with physical access or root privileges can extract this data.

URL	METHOD	PARAMETER
com.example.mobileapp/databases/user_data.db	STATIC	database

EVIDENCE / PROOF OF CONCEPT

```
Unencrypted SQLite database containing user credentials, SSN, credit card numbers, and session tokens
```

AI Analysis

Jarwis AI analyzed the application's data storage and found an unencrypted SQLite database containing highly sensitive PII. On rooted devices or through backup extraction, this data is trivially accessible.

HTTP DETAILS

REQUEST

```
Database Analysis: /data/data/com.example.mobileapp/databases/user_data.db
```

RESPONSE

```
TABLE users (unencrypted):
id|email|password_hash|ssn|credit_card|auth_token
1|john@example.com|plaintext123|123-45-6789|4111111111111111|eyJhbG...
```

✓ Remediation

Encrypt the SQLite database using SQLCipher with a user-derived key. Use Android EncryptedSharedPreferences for small data. Never store raw credentials - use secure tokens with proper expiration.

JAR-MOB-003

HIGH

SSL Certificate Pinning Not Implemented

A05 CWE-295

DESCRIPTION

The application does not implement SSL certificate pinning, making it trivially vulnerable to man-in-the-middle attacks. Attackers on the same network can intercept and modify all API traffic.

URL	METHOD	PARAMETER
https://api.example.com/*	NETWORK	TLS

EVIDENCE / PROOF OF CONCEPT

```
All HTTPS traffic intercepted using Burp Suite with custom CA certificate. 47 API endpoints captured including auth tokens and PII.
```

AI Analysis

Jarwis AI tested the application's TLS implementation by attempting to intercept traffic with a custom CA certificate. All API traffic was successfully captured without any certificate validation errors, indicating complete absence of certificate pinning.

HTTP DETAILS

REQUEST

```
Network Interception Test
Proxy: Burp Suite Professional v2024.1
Device: Android 14, CA installed as user cert
```

RESPONSE

```
Captured Sensitive Traffic:
- POST /api/auth/login - credentials in plaintext
- GET /api/users/me - full PII exposed
- POST /api/payments - credit card data visible
```

✓ Remediation

Implement certificate pinning using OkHttp CertificatePinner or Android Network Security Config. Pin to the leaf certificate or public key hash. Include backup pins for certificate rotation.

JAR-MOB-004

HIGH

Production App Built with Debuggable Flag

A04 CWE-489

DESCRIPTION

The production application has android:debuggable=true in the manifest, allowing attackers to attach debuggers, set breakpoints, and inspect runtime memory including decrypted data and tokens.

URL	METHOD	PARAMETER
AndroidManifest.xml	STATIC	debuggable

EVIDENCE / PROOF OF CONCEPT

```
<application android:debuggable="true" android:allowBackup="true" ...>
```

AI Analysis

Jarwis AI analyzed the AndroidManifest.xml and found the debuggable flag set to true along with allowBackup. This combination allows complete runtime inspection and data extraction through ADB backup.

HTTP DETAILS

REQUEST

```
Manifest Analysis: AndroidManifest.xml
APK: com.example.mobileapp_v2.5.1.apk
```

RESPONSE

```
<application
    android:debuggable="true"
    android:allowBackup="true"
    android:networkSecurityConfig="@xml/network_security_config"
    android:name=".MainApplication">
```

✓ Remediation

Set android:debuggable="false" and android:allowBackup="false" for release builds. Implement ProGuard/R8 for code obfuscation. Add runtime debugger detection that terminates the app.

JAR-MOB-005

MEDIUM

Sensitive Activities Exported Without Permission

A01 CWE-926

DESCRIPTION

Multiple sensitive activities including admin and debug interfaces are exported without proper permission protection, allowing any app on the device to launch them directly and bypass authentication.

URL	METHOD	PARAMETER
com.example.mobileapp/.internal.AdminDashboardActivity	STATIC	exported

EVIDENCE / PROOF OF CONCEPT

```
<activity android:name=".internal.AdminDashboardActivity" android:exported="true"/> without permission requirement
```

AI Analysis

Jarwis AI found multiple exported activities that appear to be admin/debug interfaces. Any malicious app on the device can launch these activities, potentially bypassing authentication and accessing sensitive functionality.

HTTP DETAILS

REQUEST

```
Manifest Analysis: Exported Components
APK: com.example.mobileapp_v2.5.1.apk
```

RESPONSE

```
Unprotected Exported Activities:
- .internal.AdminDashboardActivity
- .debug.LogViewerActivity
- .settings.HiddenFeatureActivity
- .admin.UserManagementActivity
```

✓ Remediation

Set android:exported="false" for all internal activities. If export is required, add android:permission with signature-level permission. Implement additional authentication checks within activities.

JAR-MOB-006

LOW

Excessive Logging of Sensitive Information

A09 CWE-532

DESCRIPTION

The application logs sensitive information including user credentials, API responses with PII, and authentication tokens to logcat, which is readable by other apps with READ_LOGS permission.

URL	METHOD	PARAMETER
com.example.mobileapp/Logger	DYNAMIC	logcat

EVIDENCE / PROOF OF CONCEPT

```
D/AuthService: Login response: {email: user@example.com, token: eyJhbG..., ssn: 123-45-6789}
```

AI Analysis

Jarwis AI monitored logcat during app usage and found sensitive data being logged at DEBUG level. On older Android versions, any app with READ_LOGS permission can access this data.

HTTP DETAILS

REQUEST

```
Dynamic Analysis: Logcat monitoring during authentication flow
Device: Android 12 (API 31)
```

RESPONSE

```
Sensitive Log Entries:
D/AuthService: Authenticating user: john@example.com with password: s3cr3t!
D/ApiClient: Response: {"user": {"ssn": "123-45-6789"}}
D/TokenManager: Saved token: eyJhbGciOiJIUzI1NiIs...
```

✓ Remediation

Remove all logging of sensitive data in production builds. Use ProGuard to strip Log.d() and Log.v() calls. Implement a logging wrapper that sanitizes sensitive fields automatically.

Methodology

Jarwis employs a systematic, multi-phase approach to penetration testing that combines automated scanning with AI-powered analysis to deliver comprehensive security assessments.

1 Reconnaissance

Automated discovery of endpoints, forms, APIs, and application structure using headless browser technology and intelligent crawling algorithms.

2 Pre-Auth Testing

Security testing of publicly accessible surfaces including login forms, registration flows, and unauthenticated API endpoints.

3 Authentication Analysis

Examination of authentication mechanisms, session management, token handling, and credential storage practices.

4 Post-Auth Testing

Authenticated testing including IDOR, CSRF, privilege escalation, and access control bypass attempts.

5 AI Verification

AI-powered analysis of all findings to eliminate false positives and provide contextual remediation recommendations.

6 Reporting

Generation of comprehensive reports with detailed findings, proof-of-concept evidence, and prioritized remediation guidance.

OWASP Top 10 Coverage

This assessment covers the OWASP Top 10 (2021) security risks, the industry-standard framework for web application security.

CODE	CATEGORY	STATUS
A01	Broken Access Control	✓ Tested

CODE	CATEGORY	STATUS
A02	Cryptographic Failures	✓ Tested
A03	Injection (SQL, XSS, Command)	✓ Tested
A04	Insecure Design	✓ Tested
A05	Security Misconfiguration	✓ Tested
A06	Vulnerable and Outdated Components	✓ Tested
A07	Identification and Authentication Failures	✓ Tested
A08	Software and Data Integrity Failures	✓ Tested
A09	Security Logging and Monitoring Failures	✓ Tested
A10	Server-Side Request Forgery (SSRF)	✓ Tested

Appendix

Discovered Endpoints

The following endpoints and components were identified during the reconnaissance phase of the assessment.

1. https://api.example.com/v1/auth/login
2. https://api.example.com/v1/auth/refresh
3. https://api.example.com/v1/users/me
4. https://api.example.com/v1/users/me/settings
5. https://api.example.com/v1/products
6. https://api.example.com/v1/orders
7. https://api.example.com/v1/payments
8. https://api.example.com/v1/notifications
9. https://api.example.com/v1/analytics
10. com.example.mobileapp/.MainActivity
11. com.example.mobileapp/.auth.LoginActivity
12. com.example.mobileapp/.internal.AdminDashboardActivity

Disclaimer

⚠️ Important Notice

This security assessment report is provided for informational purposes only. The findings contained herein represent the security posture of the target application at the time of testing. Jarwis AI Security Platform does not guarantee the completeness or accuracy of this assessment. Security is a continuous process, and new vulnerabilities may be

discovered after this report is generated. The recipient of this report is responsible for implementing the recommended remediation measures and verifying their effectiveness.

Terms of Use