QUANTUMSENTINEL-NEXUS

Advanced Security Analysis Report

Report ID:	PDF-TEST
Generated:	2025-10-03 09:22:49
Target Type:	File
Target:	test_file.apk
File Size:	N/A
Analysis Duration:	158 minutes
Engines Executed:	14/14
Total Findings:	8
Risk Level:	MEDIUM

Executive Summary

Risk Overview

Severity	Count	Percentage
Critical	0	0.0%
High	5	62.5%
Medium	3	37.5%
Low	0	0.0%
Informational	0	0.0%

Overall Assessment

The security analysis has identified **8 security findings** across 14 security engines. The overall risk level is assessed as **MEDIUM** with a risk score of **5.9/10**.

Vulnerability Details

1. [HIGH] GDPR Data Processing Violation

Property	Value
ID	CC-001
Severity	HIGH
CVSS Score	6.5
Confidence	70%
Engine	Compliance Assessment
Component	Data Processing
URL	
Parameter	

Description: Application processes personal data without explicit consent mechanism

Evidence:

- *Privacy policy analysis showing inadequate consent flows*

Remediation: Implement explicit consent mechanisms for data processing

---

2. [HIGH] Hardcoded API Keys

Property	Value
ID	SA-001
Severity	HIGH
CVSS Score	7.5
Confidence	90%
Engine	Static Analysis
Component	Source Code
URL	
Parameter	

Description: Hardcoded API keys found in application source code

Evidence:

- *API_KEY = '12345678-abcd-efgh-ijkl-mnopqrstuvwx'*

Remediation: Store sensitive keys in secure storage or environment variables

---

3. [HIGH] Path Traversal Vulnerability

Property	Value
ID	SAST-001
Severity	HIGH
CVSS Score	7.5
Confidence	92%
Engine	SAST Engine

Component	File Handler
URL	
Parameter	

Description: Application vulnerable to path traversal attacks in file handling

Evidence:
- *User input directly used in file path construction*

Remediation: Validate and sanitize file paths, use whitelisting

---

4. [HIGH] Insecure Data Storage

Property	Value
ID	MS-001
Severity	HIGH
CVSS Score	7.1
Confidence	88%
Engine	Mobile Security
Component	Local Storage
URL	
Parameter	

Description: Application stores sensitive data in unencrypted local storage

Evidence:
- *Unencrypted user credentials found in SharedPreferences*

Remediation: Encrypt sensitive data before storage using Android Keystore

---

5. [HIGH] IDOR (Insecure Direct Object Reference)

Property	Value
ID	BB-001
Severity	HIGH
CVSS Score	7.5
Confidence	93%
Engine	Bug Bounty Automation
Component	
URL	
Parameter	user_id

Description: Application allows access to other users' data through predictable IDs

Evidence:
- *Changed user_id from 123 to 124 and accessed other user's data*

Remediation: Implement proper authorization checks for all user data access

---

6. [MEDIUM] Missing Binary Protections

Property	Value
ID	BA-001
Severity	MEDIUM
CVSS Score	4.3
Confidence	95%
Engine	Binary Analysis
Component	Binary Executable
URL	
Parameter	

Description: Binary lacks important security protections like ASLR, DEP, and stack canaries

Evidence:

- *checksec output showing missing protections*

Remediation: Compile with security flags enabled (-fstack-protector, -D_FORTIFY_SOURCE=2)

---

7. [MEDIUM] Weak Code Obfuscation

Property	Value
ID	RE-001
Severity	MEDIUM
CVSS Score	5.0
Confidence	90%
Engine	Reverse Engineering
Component	Application Logic
URL	
Parameter	

Description: Application code can be easily reverse engineered due to weak obfuscation

Evidence:

- *Decompiled code showing clear function names and logic*

Remediation: Implement stronger code obfuscation and anti-tampering measures

---

8. [MEDIUM] Root/Jailbreak Detection Bypass

Property	Value
ID	MS-002
Severity	MEDIUM
CVSS Score	4.5
Confidence	85%
Engine	Mobile Security
Component	Security Controls
URL	
Parameter	

Description: Application's root detection can be easily bypassed

Evidence:

- *Frida script successfully bypassed root detection*

Remediation: Implement multiple layers of root detection and server-side validation

---

Proof of Concept

PoC #1: Hardcoded API Keys

Step-by-Step Reproduction:
1. Decompile the application using jadx or similar tool
2. Search for patterns like 'api_key', 'secret', 'token'
3. Verify the keys are valid by testing against the API

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #2: Missing Binary Protections

Step-by-Step Reproduction:
1. Run checksec tool on the binary
2. Observe missing security features
3. Verify with objdump or similar tools

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #3: Weak Code Obfuscation

Step-by-Step Reproduction:
1. Use jadx to decompile the APK
2. Observe readable function names and logic
3. Extract business logic and algorithms

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #4: Path Traversal Vulnerability

Step-by-Step Reproduction:
1. Submit filename: ../../../../etc/passwd
2. Observe server attempting to access system files
3. Confirm with: ..\..\..\windows\system32\drivers\etc\hosts

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #5: Insecure Data Storage

Step-by-Step Reproduction:
1. Install application on rooted device
2. Login with test credentials
3. Extract data from /data/data/[package]/shared_prefs/
4. Observe plaintext credentials

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #6: Root/Jailbreak Detection Bypass

Step-by-Step Reproduction:
1. Run application on rooted device
2. Attach Frida and hook root detection methods
3. Bypass checks and access restricted functionality

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

PoC #7: IDOR (Insecure Direct Object Reference)

Prerequisites:
• Valid user account
• User ID enumeration

Step-by-Step Reproduction:
1. Login as user with ID 123
2. Access /api/user/profile?user_id=123
3. Change user_id to 124
4. Observe unauthorized access to other user's profile

Expected Result:
The vulnerability should be successfully demonstrated, confirming the security issue.

Technical Analysis

Analysis Overview

This analysis was conducted using the QuantumSentinel-Nexus platform, employing 14 specialized security engines. The target was analyzed for 158 minutes, resulting in 8 security findings.

Security Engine Results

Engine	Status	Duration	Findings
Malware Detection	completed	1m	0
Compliance Assessment	completed	1m	1
Threat Intelligence	completed	2m	0
Static Analysis	completed	2m	1
Network Security	completed	2m	0
Binary Analysis	completed	4m	1
ML Intelligence	completed	8m	0
Dynamic Analysis	completed	3m	0
Penetration Testing	completed	5m	0
Reverse Engineering	completed	20m	1
SAST Engine	completed	18m	1
DAST Engine	completed	22m	0
Mobile Security	completed	25m	2
Bug Bounty Automation	completed	45m	1

Remediation Recommendations

Priority Actions

■■ HIGH PRIORITY (High Severity Issues):

• Implement explicit consent mechanisms for data processing
• Store sensitive keys in secure storage or environment variables
• Validate and sanitize file paths, use whitelisting
• Encrypt sensitive data before storage using Android Keystore
• Implement proper authorization checks for all user data access

General Security Recommendations

• Implement a regular security testing schedule using automated tools
• Establish a vulnerability management program with clear SLAs
• Provide security training for development teams
• Implement security code reviews for all changes
• Deploy runtime application self-protection (RASP) solutions
• Establish continuous security monitoring and alerting
• Implement zero-trust security architecture principles
• Regular penetration testing and security assessments

Testing Methodology

Analysis Approach

QuantumSentinel-Nexus employs a comprehensive 4-phase analysis methodology: **Phase 1: Initial Assessment** - Malware detection, compliance checking, and threat intelligence correlation **Phase 2: Core Security Analysis** - Static analysis, network security scanning, binary analysis, and ML-based threat detection **Phase 3: Advanced Threat Hunting** - Dynamic analysis, penetration testing, reverse engineering, SAST, and DAST **Phase 4: Specialized Analysis** - Mobile security analysis and automated bug bounty testing Each engine operates independently while sharing context and findings with other engines to provide comprehensive coverage.

Tools and Techniques

The analysis leverages industry-standard tools and proprietary techniques: • Static Analysis: Pattern matching, data flow analysis, control flow analysis • Dynamic Analysis: Runtime monitoring, behavior analysis, sandbox execution • Network Security: SSL/TLS analysis, API security testing, traffic inspection • Binary Analysis: Disassembly, reverse engineering, protection analysis • Mobile Security: Frida instrumentation, manifest analysis, runtime hooking • Machine Learning: Anomaly detection, behavioral modeling, threat correlation