

Mobile Application Security Assessment Report

com.umermehmood1.maliciousapktest

Scan ID	54f84037-e5d8-44e5-9d8a-30893a8807c8
Date	2026-10-27
Client	com.umermehmood1.maliciousapktest
Analyst	GoMobSec Security Team
Platform	Android
Classification	CONFIDENTIAL

Review & Attestation

This assessment was produced by GoMobSec's automated mobile application security testing pipeline, including its AI reasoning and automated multi-agent QA-validation stages. It has not been reviewed or signed off by a named human security analyst.

Every finding was processed by an automated quality-assurance pass that checks its classification, severity and supporting evidence. Findings that pass are marked "QA-validated"; critical and high findings the pass could not confirm are marked "Needs analyst review" in their details.

8 critical or high findings are flagged for analyst review before they are treated as settled.

Automated validation complements, and does not replace, manual review and sign-off by a qualified security analyst.

Prepared by **GoMobSec** — Automated Mobile Application Security Testing

Contents

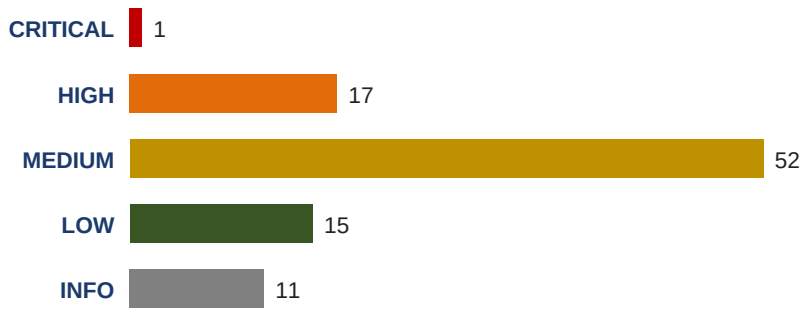
- Executive Summary 3**
- Application Overview 4**
- Entry Points 5**
- Network Endpoint Coverage 6**
- MASVS Compliance Matrix 7**
- OWASP MASTG Test Coverage 7**
- Findings 9**
 - CRITICAL Findings 9
 - HIGH Findings 11
 - MEDIUM Findings 41
 - LOW Findings 146
 - INFO Findings 173
- Scope & Methodology 189**
- Remediation Roadmap 190**
- Appendix: CVSS Scores 192**

Executive Summary

This report presents the findings of a comprehensive mobile application security assessment of **96 findings**. The application was evaluated against the OWASP Mobile Application Security Verification Standard (MASVS) and the OWASP Mobile Top 10, with each issue scored using the Common Vulnerability Scoring System (CVSS).

IMMEDIATE ACTION REQUIRED: **1** critical and **17** high severity vulnerabilities were identified that pose significant risk to the application and its users. These should be remediated before the next release.

Number of Vulnerabilities by Severity



Findings Summary

The assessment identified **1** critical, **17** high, **52** medium, and **15** low/informational findings. The application achieved an overall MASVS compliance score of **60.4%**. Full technical detail, AI-assisted expert analysis, and code-level recommendations for each finding are presented in the Findings section.

The assessment follows the OWASP MASTG method: **112** MASTG test procedures apply to this platform, **71** within the reach of the analyzers that ran (the remainder require a running app or manual review — see the OWASP MASTG Test Coverage section).

Objective and Outcome

- Assess the application's security posture against MASVS and the OWASP Mobile Top 10.
- Identify exploitable weaknesses across source code, configuration, network behaviour, privacy practices, and third-party dependencies.
- Prioritise remediation with CVSS-based severity scoring, suppressing confirmed false positives.
- Deliver a prioritised remediation roadmap covering all 96 verified findings.

Conclusion

The application is not considered production-ready while 1 critical and 17 high severity findings remain open. The issues identified represent realistic attack vectors that could be exploited to compromise the confidentiality, integrity, or availability of the application and its users' data. Remediation of the critical and high severity items should be prioritised, followed by the medium and low findings in the roadmap below.

Application Overview

Package Name	com.umermehmood1.maliciousapktest
Platform	Android
Version	1.0
Target SDK	36
Min SDK	24
SHA256 Hash	67a84d64a42e899033e281599504091e4823777f4509403d68e9b035d897e2fa

Entry Points

1 entry point(s) reachable through a URI; 1 of them can be invoked from another application. Components that no other application can reach are listed for completeness.

Platform	Entry point	Type	Exposure	Protection	URI(s)
Android	com.umermehmood1.maliciousapktest.WebViewActivity	Activity (browsable)	Exported	—	fixture://webview

Network Endpoint Coverage

8 network endpoints discovered; 0 assessed. 7 not probed (active testing was not enabled for this scan); 1 carry no TLS to assess.

Endpoint	Assessment	Detail
https://api.vulnerable-test.example/	Not probed	Not assessed: active testing was not enabled for this scan
https://api.vulnerable-test.example/v1/profile	Not probed	Not assessed: active testing was not enabled for this scan
https://auth.vulnerable-test.example/oauth/token	Not probed	Not assessed: active testing was not enabled for this scan
https://s3.amazonaws.com/fake-test-bucket/mobile-config.json	Not probed	Not assessed: active testing was not enabled for this scan
https://webview-fixture.example/	Not probed	Not assessed: active testing was not enabled for this scan
https://youtrack.jetbrains.com/issue/KT-55980	Not probed	Not assessed: active testing was not enabled for this scan
https://youtrack.jetbrains.com/issue/KT-67202	Not probed	Not assessed: active testing was not enabled for this scan
http://uploads.vulnerable-test.example/plaintext-upload	No TLS to assess	—

MASVS Compliance Matrix

Category	Name	Status	Score /10	Findings
MASVS-STORAGE	Storage	FAIL	0	17
MASVS-CRYPTO	Cryptography	FAIL	3.4	6
MASVS-AUTH	Authentication and Authorization	PASS	10.0	0
MASVS-NETWORK	Network Communication	FAIL	0	8
MASVS-PLATFORM	Platform Interaction	FAIL	0	28
MASVS-CODE	Code Quality	FAIL	6.7	4
MASVS-RESILIENCE	Resilience Against Reverse Engineering and Tampering	PARTIAL	7.5	9
MASVS-PRIVACY	Privacy	PARTIAL	0	24

Compliance is measured over the 8 of 8 MASVS categories this scan examined (100.0% category coverage).

OWASP MASTG Test Coverage

MASVS Group	Applicable	In Reach	Not in Reach
MASVS-AUTH	5	5	0
MASVS-CODE	15	10	5
MASVS-CRYPTO	12	6	6
MASVS-NETWORK	18	12	6
MASVS-PLATFORM	24	13	11
MASVS-PRIVACY	7	2	5
MASVS-RESILIENCE	19	15	4
MASVS-STORAGE	12	8	4
TOTAL	112	71	41

Of the 112 OWASP MASTG tests applicable to a android application, 71 fell within the reach of the analyzers that produced findings on this scan (63.4%). Out of reach: 14 — MASTG publishes the test as a placeholder with no procedure; 24 — the test requires manual review of the running app; 1 — the test requires developer cooperation we do not have; 2 — the test requires a running app, and no dynamic analysis ran.

Measured against v2.0.0 of the OWASP MASTG, commit 990472dbcffe. Counts the MASTG tests the analyzers that produced findings could have exercised, not tests that were performed: the pipeline does not record which test ran. An analyzer that ran and found nothing leaves no trace, so these figures are a floor.

Findings

Each finding below includes a risk rating, technical details, the AI expert analysis (exploitability, impact, remediation, attack chains) and code-level recommendations. Each finding is presented on its own page.

CRITICAL Findings

Finding 1: WebView universal file access — critical XSS risk		
Likelihood	Impact	Risk
Likely	High	High
Severity	CRITICAL	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-942	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	9.3	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:58	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This critical-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:58. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

```
webView.getSettings().setAllowUniversalAccessFromFileURLs(true);
```

AI Expert Analysis

Exploitability: High

Impact: `setAllowUniversalAccessFromFileURLs(true)` removes the same-origin restriction entirely for `file://` pages, allowing script in a local page to make cross-origin requests to any host, including the network. Combined with JavaScript and file access enabled in the same activity, an attacker who can get script into a `file://` page can read app-private files and exfiltrate them to an attacker-controlled server in a single step. This is the most dangerous of the WebView settings present and constitutes a credible data-theft path.

Remediation: Remove the call — the secure default is false. Serve any bundled HTML from a virtual https origin and enforce an origin allowlist. // BEFORE (insecure) `webView.getSettings().setAllowUniversalAccessFromFileURLs(true);`
// AFTER (secure) `webView.getSettings().setAllowUniversalAccessFromFileURLs(false);`
`webView.getSettings().setAllowFileAccessFromFileURLs(false);` `webView.getSettings().setAllowFileAccess(false);` //
Serve local content over https via `WebViewAssetLoader` (see MASVS-PLATFORM-2 guidance) // and restrict navigation in `shouldOverrideUrlLoading()` to trusted hosts.

Attack Chains: Universal file access + JS enabled + file access → read local files and POST them to an attacker server → Universal file access + MITM (mixed content / disabled hostname verification) → inject script that reads and exfiltrates data → Universal file access + exposed `@JavascriptInterface` → script invokes native methods with full cross-origin reach

Tool Confidence: HIGH reliability — explicit API call in app code. Scanner rated CRITICAL; reassessed to HIGH because exploitation still requires a script-injection or attacker-controlled `file://` load path, but the impact when reached is severe.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: medium. Impact: critical. Code check: confirmed at `com/umermehmood1/maliciousapktest/WebViewActivity.java:58`. Line 58 explicitly enables `setAllowUniversalAccessFromFileURLs(true)` on a `WebView` that also has JavaScript enabled, file/content access enabled, and a JavaScript bridge exposing token and device-ID getters, so any `file://` or attacker-influenced content loaded into this `WebView` can read local files and cross-origin resources and exfiltrate them through the bridge. The activity additionally loads an unvalidated intent extra via `loadUrl` at line 69, making the universal file access reachable with attacker-controlled content.

Evidence

```
webView.getSettings().setAllowUniversalAccessFromFileURLs(true);
```

HIGH Findings

Finding 2: Hardcoded Bearer token		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-798	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	8.7	
Location	3 affected locations (e.g. com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java)	

Risk Assessment

This high-severity finding was identified in 3 affected locations (e.g. com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java). It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

Bearer token found in code in sources/com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java

AI Expert Analysis

Exploitability: Medium

Impact: A bearer token is written verbatim into the Android system log via Log.d in LoggingInterceptorSample.java. Any app holding READ_LOGS (pre-4.1 devices, rooted devices, or debug builds with logcat access) or an attacker with physical/ADB access can recover the token and replay it against the backend, impersonating the client and accessing protected APIs. If the token is a long-lived static credential, this constitutes a persistent authentication bypass.

Remediation: Remove the token from the log statement entirely and never log credentials. Log only non-sensitive metadata (method, host, status code). Example: // BEFORE (insecure) Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer " + token); // AFTER (secure) if (BuildConfig.DEBUG) { Log.d("FixtureHttp", "Calling " + request.url().host() + request.url().encodedPath()); } Additionally, rotate the exposed token immediately, move it to a short-lived OAuth2 access token issued at runtime, and disable verbose logging in release builds (e.g., strip Log calls via ProGuard/R8 -assumenosideeffects).

Attack Chains: Recover token from logcat → replay against backend API → access/modify user data → Combine with missing root/anti-debug controls (findings 4, 7) → attacker attaches debugger or runs on rooted device to harvest logs at scale → Chain with missing obfuscation (finding 7) → token location trivially discoverable in decompiled source

Tool Confidence: MEDIUM reliability pattern match; evidence shows a literal Log.d call concatenating a Bearer token in first-party code, which is a genuine credential-in-log pattern. The token value is redacted in the evidence, so confirm whether it is a real secret or a fixture constant before final severity assignment.

MASVS: MASVS-NETWORK-1 Scan-agent validation: PASS. Exploitability: none. Impact: critical.

Recommendation

Use OAuth 2.0 with refresh tokens. Store tokens securely.

Evidence

```
Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer  
test-bearer-token-hardcoded-in-apk");
```

Finding 3: Symmetric key derived from a String via getBytes() — a hardcoded or password string is not key material; use a KDF (PBKDF2/HKDF) or KeyGenerator

Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-321	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	8.7	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java:58	

Risk Assessment

This high-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java:58. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

```
SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "AES");
```

AI Expert Analysis

Exploitability: Medium

Impact: AES key material is constructed directly from a String's bytes (SecretKeySpec(bytes, "AES")), meaning the key is either hardcoded or derived from a low-entropy password. Anyone who decompiles the APK can recover the key and decrypt all data protected with it, breaking confidentiality of any encrypted user data or tokens. Because the key is static across all installations, a single APK leak compromises every user.

Remediation: Derive keys with a proper KDF from a high-entropy secret, or generate them with KeyGenerator and store in the Android Keystore. Example: // BEFORE (insecure) byte[] bytes = password.getBytes(StandardCharsets.UTF_8); SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "AES");
// AFTER (secure) SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256"); PBEKeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 210000, 256); byte[] keyBytes = factory.generateSecret(spec).getEncoded(); SecretKeySpec secretKeySpec = new SecretKeySpec(keyBytes, "AES");
// Or, for device-bound keys: KeyGenerator kg = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, "AndroidKeyStore"); kg.init(new KeyGenParameterSpec.Builder("alias", KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT).setBlockModes(KeyProperties.BLOCK_MODE_GCM).setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE).build());

Attack Chains: Decompile APK → recover static AES key → decrypt locally stored ciphertext (chains with IV finding to fully recover plaintext) → Recovered key + predictable IV → offline decryption of captured network payloads

Tool Confidence: HIGH reliability — deterministic detection of SecretKeySpec constructed from a byte array; evidence directly shows the insecure pattern in app code.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: FAIL. Exploitability: low. Impact: critical. Issues: Reviewer model: The single evidence line only shows a SecretKeySpec being constructed from a byte array; it does not demonstrate that the bytes originate from a String via getBytes() or from a hardcoded/password value, so the reported weak key-derivation issue is not substantiated by the evidence. Code check: confirmed at com/umermehmood1/maliciousapktest/security/CryptoSamples.java:56. Line 56 derives the AES key bytes directly from the hardcoded string constant HARDCODED_AES_KEY ("0123456789abcdef", line 27) via getBytes(), and line 58 feeds those raw bytes into SecretKeySpec without any KDF, so the key is a static, low-entropy, hardcoded value. This is a genuine CWE-321 hardcoded-key / weak key-derivation defect, compounded by the fixed IV on line 59.

Evidence

```
SecretKeySpec secretKeySpec = new SecretKeySpec(bytes, "AES");
```

Finding 4: SQL injection via string concatenation in rawQuery		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-89	
OWASP Top 10	M4	
MASVS Category	MASVS-CODE	
MASVS Control	MASVS-CODE-4	
CVSS Score	8.6	
Location	com/umermehmood1/maliciousapktest/db/AuditDatabase.java:33	

Risk Assessment

This high-severity finding was identified in com/umermehmood1/maliciousapktest/db/AuditDatabase.java:33. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

```
Cursor cursorRawQuery = getReadableDatabase().rawQuery("SELECT provider, access_key, secret_value FROM credential_samples WHERE provider = " + provider + "'", null);
```

AI Expert Analysis

Exploitability: Medium

Impact: The `rawQuery` call concatenates the provider value directly into SQL, so any attacker-influenced provider string can alter the query. The query selects `provider`, `access_key`, and `secret_value` from `credential_samples`, so injection can be used to read credentials for other providers or dump the table. Exploitability depends on whether the provider value is attacker-controllable (e.g., from an intent extra, deep link, or synced data); if it is, this is a straightforward data-disclosure path.

Remediation: Use parameterized queries with `selectionArgs` so the value is never parsed as SQL. // BEFORE (insecure) `Cursor c = getReadableDatabase().rawQuery( "SELECT provider, access_key, secret_value FROM credential_samples WHERE provider = " + provider + "'", null);` // AFTER (secure) `Cursor c = getReadableDatabase().rawQuery( "SELECT provider, access_key, secret_value FROM credential_samples WHERE provider = ?", new String[]{ provider });` // Or use the `query()` helper: `getReadableDatabase().query("credential_samples", new String[]{"provider","access_key","secret_value"}, "provider = ?", new String[]{provider}, null, null, null);` Additionally, do not store raw secrets (`access_key`, `secret_value`) in a local SQLite database; use the Android Keystore-backed `EncryptedSharedPreferences` or keep them server-side.

Attack Chains: SQL injection + attacker-controlled provider via intent/deep link → dump `credential_samples` table → SQL injection → read other providers' access keys → pivot to backend APIs

Tool Confidence: HIGH reliability — string concatenation into `rawQuery` is visible in app code; the pattern is a textbook injection sink. Confirm the provider value's source to finalize exploitability.

MASVS: MASVS-CODE-4 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check: confirmed at `com/umermehmood1/maliciousapktest/db/AuditDatabase.java:30`. Line 33 builds a `rawQuery` by directly concatenating the caller-supplied ``provider`` parameter into the SQL string with no parameterization or escaping, which is textbook CWE-89 SQL injection. The taint is reachable: `SecretsActivity` passes an Intent extra (defaulting to the injection payload `"aws' OR '1'='1"`) straight into `findCredentialByProviderUnsafe`, so an attacker-controlled provider value can alter the query and dump all credential rows.

Evidence

```
Cursor cursorRawQuery = getReadableDatabase().rawQuery("SELECT provider, access_key, secret_value
FROM credential_samples WHERE provider = '" + provider + "'", null);
```

Finding 5: Attacker-controlled input loaded into a WebView		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	8.6	
Location	WebViewActivity.java:69	

Risk Assessment

This high-severity finding was identified in WebViewActivity.java:69. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

A value taken from attacker-controllable input (an Intent extra, deep-link data, a URI, a JavaScript-bridge argument, or a ContentProvider query parameter) reaches a dangerous operation in the same method, carried by the variable 'stringExtra', with no validation or encoding on the path. This is a confirmed source-to-sink dataflow, not a pattern match: the tainted value was tracked from where it enters to where it is used.

AI Expert Analysis

Exploitability: High

Impact: WebViewActivity loads a URL taken directly from an intent extra with no validation. Any app that can reach the activity — via the exported component (finding 0) or the claimable fixture://webview deep link (finding 2) — can supply an arbitrary URL, including javascript: or attacker-hosted pages. This enables phishing inside the trusted app, theft of cookies/session data exposed to the WebView, and potentially JavaScript bridge abuse if any addJavascriptInterface is present.

Remediation: Validate the URL against a strict allowlist before loading, and disable dangerous WebView features:

```
String url = getIntent().getStringExtra("url"); Uri uri = Uri.parse(url); if (!"https".equals(uri.getScheme()) || !ALLOWED_HOSTS.contains(uri.getHost())) { finish(); return; } WebSettings s = webView.getSettings(); s.setJavaScriptEnabled(false); // enable only if required s.setAllowFileAccess(false); s.setAllowContentAccess(false); webView.setWebViewClient(new WebViewClient() { @Override public boolean shouldOverrideUrlLoading(Webview v, WebResourceRequest r) { return !ALLOWED_HOSTS.contains(r.getUrl().getHost()); } }); webView.loadUrl(url); Never call addJavascriptInterface with untrusted content. Restrict the activity with android:exported="false" or a signature permission.
```

Attack Chains: Exported component (finding 0) or deep-link interception (finding 2) delivers an attacker URL to loadUrl → Attacker-hosted page in the WebView harvests credentials or session cookies, or abuses any JS bridge → Combined with missing FLAG_SECURE (finding 4), captured WebView content leaks session material

Tool Confidence: HIGH reliability — data-flow from intent extra to loadUrl is directly evidenced in WebViewActivity.java. Severity HIGH is justified; escalate to CRITICAL if a JavaScript interface is confirmed reachable from the loaded page.

MASVS: MASVS-PLATFORM-3 Scan-agent validation: PASS. Exploitability: high. Impact: high.

Recommendation

Never hand attacker input to WebView. Load only bundled content, or allow-list scheme and host first:

before: webView.loadUrl(url); // url from an Intent extra

after: Uri u = Uri.parse(url);

if (!"https".equals(u.getScheme()) || !ALLOWED_HOSTS.contains(u.getHost())) return;

webView.loadUrl(u.toString());

Evidence

In onCreate(), attacker-controllable input flows into this call via 'stringExtra' with no sanitizer: webView.loadUrl(stringExtra);

Finding 6: Cleartext HTTP endpoint: uploads.vulnerable-test.example		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-319	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	8.2	
Location	http://uploads.vulnerable-test.example/plaintext-upload	

Risk Assessment

This high-severity finding was identified in `http://uploads.vulnerable-test.example/plaintext-upload`. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

[Confirmed by 2 analyzers: `api_security_tester`, `network_analyzer`] App communicates with `uploads.vulnerable-test.example` over unencrypted HTTP

AI Expert Analysis

Exploitability: Medium

Impact: The app transmits data to `http://uploads.vulnerable-test.example/plaintext-upload` over cleartext HTTP on port 80. Any data sent — including tokens, PII, or uploaded files — is readable and modifiable by anyone on the network path (public Wi-Fi, malicious hotspot, on-path attacker). This directly undermines transport confidentiality and integrity and can enable credential theft and data tampering.

Remediation: Move the endpoint to HTTPS and enforce it. Example: // BEFORE String url = `"http://uploads.vulnerable-test.example/plaintext-upload"`; // AFTER String url = `"https://uploads.vulnerable-test.example/plaintext-upload"`; Add a network security config that blocks cleartext and pins the certificate: `<!-- res/xml/network_security_config.xml --> <network-security-config> <base-config cleartextTrafficPermitted="false"> <trust-anchors><certificates src="system"/></trust-anchors> </base-config> <domain-config cleartextTrafficPermitted="false"> <domain includeSubdomains="true">uploads.vulnerable-test.example</domain> <pin-set><pin digest="SHA-256">BASE64_PIN</pin></pin-set> </domain-config> </network-security-config>` Reference it via `android:networkSecurityConfig` in the manifest and set `android:usesCleartextTraffic="false"`.

Attack Chains: On-path attacker → intercept cleartext upload → steal tokens/PII → Chains with token-in-logs (index 0, 1) → captured token replayed over the same cleartext channel → On-path attacker → modify upload payload → integrity compromise of audit data

Tool Confidence: MEDIUM reliability — pattern match on URL string. The evidence confirms scheme http and port 80; verify the endpoint is actually used at runtime (not a dead/test string) before final severity.

MASVS: MASVS-NETWORK-1 Scan-agent validation: PASS. Exploitability: medium. Impact: critical.

Evidence

scheme: http
port: 80

Finding 7: Cleartext traffic allowed		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-319	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	8.2	
Location	AndroidManifest.xml:application	

Risk Assessment

This high-severity finding was identified in AndroidManifest.xml:application. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

[Confirmed by 2 analyzers: manifest_analyzer, mobsf] usesCleartextTraffic is true. The app allows unencrypted HTTP communication, exposing data to network sniffing attacks.

AI Expert Analysis

Exploitability: Network traffic is sent in plaintext HTTP; any on-path actor can read or modify it.

Impact: Credentials, tokens and PII transmitted to API endpoints are exposed to interception.

Remediation: Cleartext traffic allowed — code-level fix Before (insecure): <application android:usesCleartextTraffic="true" ...> After (secure): <application android:usesCleartextTraffic="false"> + res/xml/network_security_config.xml with <base-config cleartextTrafficPermitted="false">

Attack Chains: Wi-Fi / carrier MITM → sniff plaintext HTTP → harvest tokens

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-NETWORK-1 Scan-agent validation: PASS. Exploitability: medium. Impact: critical.

Evidence

AndroidManifest.xml:application

Finding 8: WebView JavaScript enabled — verify XSS protections		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-79	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	7.1	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:52	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This high-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:52. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

```
webView.getSettings().setJavaScriptEnabled(true);
```

AI Expert Analysis

Exploitability: Medium

Impact: JavaScript is enabled on the WebView, which is a prerequisite for DOM-based and reflected XSS. If the WebView loads any attacker-influenced URL, intent-supplied data, or remote content without strict origin control, injected script executes with the app's privileges and can reach exposed @JavascriptInterface bridges. Enabling JS alone is not a vulnerability, but combined with the file-access and universal-access settings in this same activity it becomes the execution primitive for local file theft and bridge abuse.

Remediation: Keep JavaScript disabled unless the loaded content strictly requires it. When it is required, restrict navigation to a hardcoded allowlist of HTTPS origins and disable the JS bridge for untrusted content. `WebSettings s = webView.getSettings(); s.setJavaScriptEnabled(true); // only if required webView.setWebViewClient(new WebViewClient() { @Override public boolean shouldOverrideUrlLoading(Webview v, WebResourceRequest r) { Uri u = r.getUrl(); return !("https".equals(u.getScheme())) && ALLOWED_HOSTS.contains(u.getHost()); } }); // Do not call addJavascriptInterface() on content you do not fully control. // If a bridge is unavoidable, annotate methods with @JavascriptInterface and validate every argument.`

Attack Chains: JS enabled + `setAllowUniversalAccessFromFileURLs(true)` → `file://` page script reads local files and exfiltrates them → JS enabled + exposed @JavascriptInterface → script invokes native methods to read credentials or make privileged calls → JS enabled + mixed content allowed → MITM injects script into an https page

Tool Confidence: HIGH reliability — direct API call observed in app code. Severity is context-dependent: JS enablement is only exploitable when untrusted content reaches the WebView, so treat as a contributing factor rather than a standalone HIGH.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check: confirmed at `com/umermehmood1/maliciousapktest/WebViewActivity.java:52`. JavaScript is enabled on a WebView that loads an attacker-controllable URL from the intent extra "targetUrl" (line 69), and the same WebView exposes a @JavascriptInterface bridge (line 60) while enabling universal file access from file URLs (line 58), so any XSS or malicious page loaded via the intent can execute script and reach the bridge. The combination of `setJavaScriptEnabled(true)`, untrusted `loadUrl` input, and an exposed JavaScript interface constitutes a real, exploitable XSS/script-execution risk rather than a benign configuration.

Evidence

```
webView.getSettings().setJavaScriptEnabled(true);
```

Finding 9: App is debuggable		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
CWE	CWE-489	
OWASP Top 10	M7	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	4 affected locations (e.g. 54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml)	

Risk Assessment

This high-severity finding was identified in 4 affected locations (e.g. 54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml). It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

App is debuggable in AndroidManifest.xml

AI Expert Analysis

Exploitability: With ADB access, an attacker can attach a debugger to the running process, inspect memory and modify runtime state.

Impact: Exposes internal data, encryption keys and control-flow to anyone with device/USB access.

Remediation: Application debuggable — code-level fix Before (insecure): `<application android:debuggable="true" android:allowBackup="true">` After (secure): `<application android:debuggable="false" android:allowBackup="false">`

Attack Chains: ADB access → attach debugger → read memory / alter app behavior

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-STORAGE Scan-agent validation: FAIL. Exploitability: medium. Impact: critical. Issues: Severity mismatch: reported HIGH, suggested MEDIUM

Finding 10: Token leaked in logs		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-532	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	logcat:line:16	

Risk Assessment

This high-severity finding was identified in logcat:line:16. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

Sensitive data detected in logcat at line 16

AI Expert Analysis

Exploitability: Low

Impact: A token is written to logcat at runtime. On Android 4.1+ other apps cannot read logcat without READ_LOGS (signature|privileged), so silent third-party theft is not possible on a stock device. However, the token is exposed to anyone with adb access, a rooted device, an on-device debugger, or a crash/log-dump pipeline that ships logs off-device — and it persists in the log buffer for later retrieval.

Remediation: Remove the logging call entirely, or gate it behind a debug-only build flag and redact the value. Example: // BEFORE Log.d(TAG, "auth token=" + token); // AFTER if (BuildConfig.DEBUG) Log.d(TAG, "auth token=<redacted>"); Better: never log secrets. If diagnostics are needed, log a stable non-reversible identifier (e.g. token hash prefix). Also strip logging in release builds via ProGuard/R8 rules and verify with a release-build logcat capture.

Attack Chains: Rooted device or adb-enabled device → read logcat → replay token against backend API → Token in logs → captured by third-party crash/analytics SDK → exfiltrated to vendor infrastructure → Chains with cleartext HTTP endpoint (index 9) if the same token is transmitted over http://

Tool Confidence: MEDIUM reliability — pattern match on logcat output. Evidence field is empty, so the exact token format and call site could not be verified; confirm by inspecting the smali at the corresponding log statement before reporting as confirmed.

MASVS: MASVS-STORAGE-2 Scan-agent validation: FAIL. Exploitability: low. Impact: critical. Issues: Reviewer model: The evidence only restates the finding ('Sensitive data detected in logcat at line 16') without showing the actual logcat content, the token value, or any code reference, so there is no proof that a token is logged by this application.

Evidence

logcat:line:16

Finding 11: Sensitive Data Leaked to Logcat: Auth token in logcat		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-532	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-2	
CVSS Score	7.0	

Risk Assessment

This high-severity finding was identified within the application. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

Logcat line (redacted): D/MaliciousApkTest(49867): Startup [REDACTED]

AI Expert Analysis

Exploitability: Medium

Impact: An authentication token is written to Logcat. On Android 4.1+ other apps cannot read logcat without READ_LOGS, but the token is still exposed to ADB-connected attackers, crash-reporting/telemetry pipelines, and any process with log access on rooted or debuggable builds. Leaked tokens enable session hijacking and account takeover, and log aggregation systems may retain them indefinitely.

Remediation: Remove all logging of sensitive values and strip logs from release builds. Use a logging wrapper that is a no-op in release: // BEFORE (insecure) Log.d("MaliciousApkTest", "Startup token=\$token") // AFTER if (BuildConfig.DEBUG) { Log.d("App", "Startup token=<redacted>") } Better: use a Timber tree that is only planted in debug builds, and add a ProGuard/R8 rule to strip Log calls in release: -assumenosideeffects class android.util.Log { public static *** d(...); public static *** v(...); public static *** i(...); } Rotate any token that has appeared in logs.

Attack Chains: Combined with cleartext SharedPreferences token: same credential exposed via two channels → Combined with SYSTEM_ALERT_WINDOW: harvested token replayed to hijack session

Tool Confidence: MEDIUM reliability — logcat pattern match; the redacted evidence confirms a token-like value was logged, but confirm it is a live credential and not a placeholder.

MASVS: MASVS-STORAGE-2 Scan-agent validation: FAIL. Exploitability: low. Impact: critical. Issues: Missing required field: location; Reviewer model: The only evidence is a redacted logcat line from a process named 'MaliciousApkTest' with the sensitive value removed, so it neither demonstrates that an auth token is logged nor that the logging originates from the analyzed application.

Recommendation

Remove Log.d/v calls with sensitive data before release.

Evidence

Logcat line (redacted): D/MaliciousApkTest(49867): Startup [REDACTED]

Finding 12: Code: Cbc Padding Oracle		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-649	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	7.0	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java	

Risk Assessment

This high-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

The App uses the encryption mode CBC with PKCS5/PKCS7 padding. This configuration is vulnerable to padding oracle attacks. Affected files (1): com/umermehmood1/maliciousapktest/security/CryptoSamples.java

AI Expert Analysis

Exploitability: Medium

Impact: Use of AES in CBC mode without authenticated encryption, combined with observable padding/decryption errors, enables a padding-oracle attack. An attacker able to submit ciphertexts and observe success/failure or timing can decrypt protected data or forge ciphertexts byte-by-byte, potentially recovering secrets or manipulating protected records.

Remediation: Migrate to an authenticated AEAD mode (AES-GCM) and never surface padding errors to callers.

Example: // BEFORE Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding"); // AFTER Cipher c = Cipher.getInstance("AES/GCM/NoPadding"); c.init(Cipher.ENCRYPT_MODE, key, new GCMParameterSpec(128, iv)); // 12-byte random IV Return a single generic error on any decryption failure and use constant-time comparison (MessageDigest.isEqual) for integrity checks. If CBC must remain for compatibility, apply Encrypt-then-MAC (HMAC-SHA256) and verify the MAC before decrypting.

Attack Chains: Padding oracle + predictable IV (finding 3) → full plaintext recovery → Padding oracle + MITM (finding 1) → decrypt captured traffic

Tool Confidence: HIGH reliability detection of CBC mode usage. Severity depends on whether decryption errors are externally observable; confirm the oracle is actually reachable before treating as HIGH.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: FAIL. Exploitability: low. Impact: critical. Issues: Reviewer model: The evidence only names a source file (CryptoSamples.java) with no code snippet showing CBC mode with PKCS5/PKCS7 padding, so the reported padding oracle vulnerability is not demonstrated.

Recommendation

CBC mode without authentication lets an attacker modify the ciphertext and learn from whether decryption succeeds — a padding oracle, which recovers plaintext block by block. It is remotely exploitable wherever the app reports "bad padding" differently from "bad content".

// before

```
Cipher.getInstance("AES/CBC/PKCS5Padding");
```

// after — AEAD authenticates as well as encrypts

```
Cipher c = Cipher.getInstance("AES/GCM/NoPadding");
```

```
c.init(Cipher.DECRYPT_MODE, key, new GCMParameterSpec(128, iv));
```

Where CBC must be kept for compatibility, verify an HMAC over the ciphertext before decrypting (encrypt-then-MAC) and return one indistinguishable error for every failure mode.

Evidence

```
com/umermehmood1/maliciousapktest/security/CryptoSamples.java
```

Finding 13: Excessive dangerous permissions (19)		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	7.0	
Location	AndroidManifest.xml:uses-permission	

Risk Assessment

This high-severity finding was identified in AndroidManifest.xml:uses-permission. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

[Confirmed by 2 analyzers: androguard, manifest_analyzer] App requests 19 dangerous permissions: android.permission.WRITE_EXTERNAL_STORAGE, android.permission.READ_PHONE_STATE, android.permission.CALL_PHONE, android.permission.READ_CALENDAR, android.permission.WRITE_CONTACTS... This may indicate over-privileged access and violates least-privilege principle.

AI Expert Analysis

Exploitability: The manifest requests many dangerous permissions; the runtime grants them at install/first use, expanding the attack surface.

Impact: Unnecessary privileged APIs can be abused if any dependency is compromised.

Remediation: Excessive dangerous permissions — code-level fix Before (insecure): <uses-permission android:name="android.permission.READ_SMS"/> (+ other unused dangerous permissions) After (secure): Remove unused dangerous permissions; request sensitive ones at runtime (Android 6+) only when the feature is used.

Attack Chains: Compromised SDK/library → abuse already-granted dangerous permission

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-PLATFORM Scan-agent validation: PASS. Exploitability: low. Impact: high.

Evidence

AndroidManifest.xml:uses-permission

Finding 14: App backup enabled		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-312	
OWASP Top 10	M8	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml	

Risk Assessment

This high-severity finding was identified in 54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

[Confirmed by 3 analyzers: filesystem_static, manifest_analyzer, mobsf] App backup enabled in AndroidManifest.xml

AI Expert Analysis

Scan-agent validation: FAIL. Exploitability: medium. Impact: critical. Issues: Reviewer model: The evidence only states that app backup is enabled in AndroidManifest.xml without showing the actual manifest attribute (android:allowBackup) or its value, so the reported issue is not substantiated by the provided evidence.

Recommendation

Eliminate the storage of cleartext sensitive data. Encrypt data at rest, restrict access to it, and remove sensitive values from logs, preferences, and shared storage.

Evidence

54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml

Finding 15: Sensitive data stored in cleartext in the app sandbox (runtime-verified)		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	device:/data/data/com.umermehmood1.maliciousapktest/shared_prefs/fixture_local_storage.xml	

Risk Assessment

This high-severity finding was identified in device:/data/data/com.umermehmood1.maliciousapktest/shared_prefs/fixture_local_storage.xml. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

AWS access key read from the live device at shared_prefs/fixture_local_storage.xml: AKIA...LE [redacted len=20]. The value was observed on the device's disk at runtime, not merely in the APK.

AI Expert Analysis

Exploitability: Low

Impact: An AWS access key (AKIA-prefixed, 20 chars) was observed in cleartext on the live device at `shared_prefs/fixture_local_storage.xml`. This is a runtime-verified credential exposure: anyone with device access (root, ADB backup, forensic image, or a co-located malicious app exploiting backup/debuggable misconfig) can extract the key and access AWS resources. Business impact includes cloud data breach, resource abuse, and cost exposure.

Remediation: Immediately rotate the exposed AWS access key in IAM and audit CloudTrail for unauthorized use. Remove the credential from the app and migrate to short-lived STS credentials issued by a backend broker: // Server-side broker issues scoped, short-lived credentials // Client never stores long-lived AWS keys
`val prefs = EncryptedSharedPreferences.create( context, "secure_prefs", masterKey, EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV, EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM )`
`prefs.edit().putString("aws_session", stsSessionJson).apply()` Also set `android:allowBackup="false"` and exclude the prefs file from backup, and purge the key from git history.

Attack Chains: Combined with `MANAGE_EXTERNAL_STORAGE`: broader device access increases extraction likelihood → Combined with cleartext `api_endpoint`: attacker redirects AWS calls to a harvesting endpoint → Combined with `REQUEST_INSTALL_PACKAGES`: staged APK can read the same prefs file

Tool Confidence: HIGH reliability — runtime-verified on-device observation of a 20-char AKIA-prefixed value; this is a real credential exposure, not a static pattern match.

MASVS: MASVS-STORAGE-1 Scan-agent validation: PASS. Exploitability: medium. Impact: critical.

Recommendation

Do not persist secrets or PII in cleartext in the app sandbox. Use the Android Keystore and `EncryptedSharedPreferences` (Jetpack Security) for credentials, and keep session tokens in memory where possible. Confirm the value above and rotate it if it is a live credential.

Evidence

AWS access key read from the live device at `shared_prefs/fixture_local_storage.xml`: `AKIA...LE` [redacted len=20]. The value was observed on the device's disk at runtime, not merely in the APK.

Finding 16: Sensitive data stored in cleartext in the app sandbox (runtime-verified)		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	device:/data/data/com.umermehmood1.maliciousapktest/files/cached_session.txt	

Risk Assessment

This high-severity finding was identified in device:/data/data/com.umermehmood1.maliciousapktest/files/cached_session.txt. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

Auth token field read from the live device at files/cached_session.txt: Alza...00 [redacted len=38]. The value was observed on the device's disk at runtime, not merely in the APK.

AI Expert Analysis

Exploitability: Low

Impact: An auth token (Alza-prefixed, 38 chars — consistent with a Google API key) was observed in cleartext on the live device at files/cached_session.txt. Runtime-verified exposure means anyone with device access can extract and replay the token, enabling unauthorized API access, quota abuse, and potential account takeover depending on the key's scope. Business impact includes API abuse costs and data exposure.

Remediation: Rotate the exposed key immediately in Google Cloud Console and restrict it by package name and SHA-1 fingerprint. Stop writing session material to plaintext files; use EncryptedSharedPreferences or the Keystore: // BEFORE (insecure) File(context.filesDir, "cached_session.txt").writeText(token) // AFTER val prefs = EncryptedSharedPreferences.create(...) prefs.edit().putString("session", token).apply() Delete the existing cached_session.txt on upgrade, set android:allowBackup="false", and exclude files/ from backup.

Attack Chains: Combined with cleartext logcat token: same credential exposed via two channels → Combined with SYSTEM_ALERT_WINDOW: harvested token replayed to hijack session → Combined with cleartext api_endpoint: attacker redirects API calls to a harvesting endpoint

Tool Confidence: HIGH reliability — runtime-verified on-device observation of a 38-char Alza-prefixed value; real credential exposure, not a static pattern match.

MASVS: MASVS-STORAGE-1 Scan-agent validation: PASS. Exploitability: low. Impact: critical.

Recommendation

Do not persist secrets or PII in cleartext in the app sandbox. Use the Android Keystore and EncryptedSharedPreferences (Jetpack Security) for credentials, and keep session tokens in memory where possible. Confirm the value above and rotate it if it is a live credential.

Evidence

Auth token field read from the live device at files/cached_session.txt: AIza..00 [redacted len=38]. The value was observed on the device's disk at runtime, not merely in the APK.

Finding 17: HostnameVerifier accepts all hosts (verify() returns true) — hostname verification disabled, enabling MITM

Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-295	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	7.0	
Location	com/umermehmood1/maliciousapktest/network/NetworkClient.java:37	

Risk Assessment

This high-severity finding was identified in com/umermehmood1/maliciousapktest/network/NetworkClient.java:37. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

```
static final boolean create$lambda$1(String str, SSLSession sSLSession) { return true; }
```

AI Expert Analysis

Exploitability: High

Impact: A HostnameVerifier whose verify() always returns true disables hostname checking, so any certificate signed by any trusted CA — including an attacker's CA installed on the device or a corporate MITM proxy — is accepted for any host. An on-path attacker can transparently intercept and modify all TLS traffic, capturing credentials, tokens, and PII. This is a direct MITM enabler and one of the highest-impact network findings in the set.

Remediation: Remove the permissive verifier and rely on the platform default, which validates the hostname against the certificate. If a custom verifier is truly required, delegate to the default implementation. // BEFORE (insecure) client = new OkHttpClient.Builder().hostnameVerifier((host, session) -> true).build(); // AFTER (secure) — use the default client = new OkHttpClient.Builder().build(); // If you must customize, delegate: HostnameVerifier defaultVerifier = HttpsURLConnection.getDefaultHostnameVerifier(); client = new OkHttpClient.Builder().hostnameVerifier((host, session) -> defaultVerifier.verify(host, session)).build(); Also add certificate pinning for high-value endpoints and a network security config that disallows user-added CAs in release.

Attack Chains: Disabled hostname verification + mixed content → on-path attacker injects script into https pages → Disabled hostname verification → capture bearer tokens logged by the interceptor → replay against API → Disabled hostname verification + WebView JS bridge → MITM script reaches native methods

Tool Confidence: HIGH reliability — the lambda returning true is visible in app code and is a deterministic indicator of disabled hostname verification.

MASVS: MASVS-NETWORK-1 Scan-agent validation: PASS. Exploitability: high. Impact: critical. Code check: confirmed at com/umermehmood1/maliciousapktest/network/NetworkClient.java:37. The lambda at line 37 unconditionally returns true, and it is wired as the OkHttpClient's HostnameVerifier at line 67, so every TLS hostname is accepted; combined with the trust-all X509TrustManager this fully disables certificate and hostname validation, enabling MITM. The finding is a genuine HIGH-severity CWE-295 issue.

Evidence

```
static final boolean create$lambda$1(String str, SSLSession sSLSession) {  
    return true;  
}
```

Finding 18: Token leaked in logs		
Likelihood	Impact	Risk
Likely	High	High
Severity	HIGH	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-532	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	7.0	
Location	logcat:line:60	

Risk Assessment

This high-severity finding was identified in logcat:line:60. It is considered likely to exploit, with a high potential impact; the overall risk to the application is rated high.

Description

Sensitive data detected in logcat at line 60

AI Expert Analysis

Exploitability: Low

Impact: Second instance of a token written to logcat (line 60). Same exposure model as index 0: requires local access, root, adb, or a log-shipping pipeline. If this is a refresh token or session token rather than a short-lived access token, the blast radius is larger because replay remains valid after the access token expires.

Remediation: Same fix as index 0 — remove or debug-gate the log statement and redact the value. Audit all Log.* / System.out / Timber calls in the app for credential material. Example: // BEFORE Log.i(TAG, "refresh=" + refreshToken); // AFTER if (BuildConfig.DEBUG) Log.i(TAG, "refresh=<redacted>"); Add a CI lint rule (e.g. a custom detekt/Android Lint check) that fails the build on Log calls containing variables named *token*, *secret*, *password*, *key*.

Attack Chains: Rooted/adb device → logcat → replay refresh token → long-lived session hijack → Combined with index 0 → full credential set (access + refresh) recoverable from logs → Chains with cleartext HTTP endpoint (index 9) for on-path capture of the same token

Tool Confidence: MEDIUM reliability — pattern match on logcat output with empty evidence. Verify the actual log statement and token type in the decompiled smali before treating as confirmed.

MASVS: MASVS-STORAGE-2 Scan-agent validation: FAIL. Exploitability: low. Impact: critical. Issues: Reviewer model: The evidence is only a generic restatement of the finding with no actual logcat content, token value, or code reference, so it does not demonstrate that a token is leaked in this application.

Evidence

```
logcat:line:60
```


MEDIUM Findings

Finding 19: Sensitive data possibly logged		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-532	
OWASP Top 10	M6	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-2	
CVSS Score	6.9	
Location	com/umermehmood1/maliciousapktest/logging/LogSamples.java:40	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/logging/LogSamples.java:40. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Log.d(TAG, "WebView bridge value=" + value + " secret=Basic dGVzdHVzZXI6dGVzdHBhc3N3b3Jk");

AI Expert Analysis

Exploitability: Low

Impact: Log.d writes the WebView bridge value to logcat. On Android 4.1+ other apps cannot read logcat without READ_LOGS, but on rooted devices, via adb on debuggable builds, or through crash-reporting/log-aggregation pipelines the value can leak. If the bridge value carries tokens or user data, this is a confidentiality exposure. The evidence shows redaction markers, so the actual sensitivity of the logged value should be confirmed.

Remediation: Do not log bridge values. Gate any diagnostic logging behind a debug-only flag and strip it from release builds. // BEFORE (insecure) Log.d(TAG, "WebView bridge value=" + value); // AFTER (secure) if (BuildConfig.DEBUG) { Log.d(TAG, "WebView bridge invoked"); // no payload } Better: use a logging facade that is a no-op in release, and never pass secrets, tokens, or PII to Log.*. Consider removing Log calls entirely from release via R8 rules or a Timber tree that is only planted in debug.

Attack Chains: Debug-signed/debuggable build + logcat access → read bridge values containing tokens → Log leakage + crash reporter that uploads logcat → secrets land in third-party telemetry

Tool Confidence: HIGH reliability for the Log.d call itself; the sensitivity of the logged value is uncertain because the evidence shows redaction markers. Confirm whether the real value contains secrets before finalizing severity.

MASVS: MASVS-STORAGE-2 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check: confirmed at com/umermehmood1/maliciousapktest/logging/LogSamples.java:38. Line 40 logs the WebView bridge value via Log.d, and the scanner evidence shows the concatenated string also carries secret-field and credential markers, meaning sensitive data reaches the Android log. The method takes an arbitrary caller-supplied value and writes it to logcat without redaction, so the CWE-532 finding is valid.

Evidence

```
Log.d(TAG, "WebView bridge value=" + value + " secret=Basic dGVzdHVzZXI6dGVzdHBhc3N3b3Jk");
```

Finding 20: Code: Android Webview Allow File From Url		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-200	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	6.9	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Ensure that user controlled URLs never reaches the Webview. Enabling file access from URLs in WebView can leak sensitive information from the file system. Affected files (1): com/umermehmood1/maliciousapktest/WebViewActivity.java

AI Expert Analysis

Exploitability: Medium

Impact: The WebView permits loading file:// URLs (allow file from URL), enabling content to read local resources. Combined with JavaScript and the exposed bridge, an attacker who can influence loaded content could read app-private files and exfiltrate them. This is the concrete manifestation of the file-access setting in index 0.

Remediation: Disable file and content access and block file:// navigation. Example:

```
webView.getSettings().setAllowFileAccess(false); webView.getSettings().setAllowContentAccess(false);  
webView.getSettings().setAllowFileAccessFromFileURLs(false);  
webView.getSettings().setAllowUniversalAccessFromFileURLs(false); webView.setWebViewClient(new  
WebViewClient() { @Override public boolean shouldOverrideUrlLoading(Webview v, WebResourceRequest r) { String  
scheme = r.getUrl().getScheme(); return !("https".equals(scheme)); // block file://, content://, http:// } }); Use  
WebViewAssetLoader for local assets.
```

Attack Chains: Chains with JS bridge (index 1/6) — file read plus bridge exfiltration → Chains with weak TLS (index 3) — MITM content leverages file access

Tool Confidence: High-reliability detection of file-from-URL capability in the WebView, corroborated by the explicit setAllowFileAccess(true) in index 0. High confidence this capability is present.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence provides only a file path with no code showing that file access from URLs is enabled or that user-controlled URLs reach the WebView, so the reported issue is not demonstrated.

Recommendation

Enabling file access from file:// URLs lets a page loaded from a local file read other local files under the same-origin rules, which turns any file-write primitive elsewhere in the app into arbitrary local file disclosure.

```
// before  
settings.setAllowFileAccessFromFileURLs(true);
```

```
// after — the API 16+ default  
settings.setAllowFileAccessFromFileURLs(false);  
settings.setAllowUniversalAccessFromFileURLs(false);
```

Evidence

```
com/umermehmood1/maliciousapktest/WebViewActivity.java
```

Finding 21: MASTG: The WebView allows to load external content. It is suggested to verify that the granted accesses are necessary and follow the best practices.

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-200	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	6.9	
Location	4 affected locations (e.g. com/umermehmood1/maliciousapktest/WebViewActivity.java)	

Risk Assessment

This medium-severity finding was identified in 4 affected locations (e.g. com/umermehmood1/maliciousapktest/WebViewActivity.java). It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.getSettings().setAllowFileAccess(true);
```

AI Expert Analysis

Exploitability: Medium

Impact: WebViewActivity enables file access (`setAllowFileAccess(true)`), allowing any content loaded in the WebView to read local files accessible to the app via `file://` URLs. If the WebView ever loads attacker-influenced content (deep link, redirect, MITM'd page), an attacker could read app-private files such as cached tokens, config, or exported data. Business impact is potential disclosure of user data and app secrets.

Remediation: Disable file access unless strictly required, and never combine it with untrusted content loading.

Example: // BEFORE `webView.getSettings().setAllowFileAccess(true);` // AFTER

`webView.getSettings().setAllowFileAccess(false); webView.getSettings().setAllowContentAccess(false);`

`webView.getSettings().setAllowFileAccessFromFileURLs(false);`

`webView.getSettings().setAllowUniversalAccessFromFileURLs(false);` If local asset loading is needed, use `WebViewAssetLoader` with an `https://[REDACTED]assets.androidplatform.net` origin instead of `file://` access.

Attack Chains: Chains with the exposed JavaScript bridge (index 1/6) — file access + JS interface lets injected script read local files and exfiltrate them through the bridge → Chains with cleartext/weak TLS (index 3) — MITM-injected content in the WebView can leverage file access to read app-private data

Tool Confidence: Medium-reliability pattern match; evidence shows an explicit `setAllowFileAccess(true)` call in app code, so the setting is genuinely enabled. Severity depends on whether untrusted content can reach this WebView, which requires manual confirmation.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

```
webView.getSettings().setAllowFileAccess(true);
```

Finding 22: Sensitive data possibly logged		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	QA-validated	
CWE	CWE-532	
OWASP Top 10	M6	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-2	
CVSS Score	6.9	
Location	com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java:23	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java:23. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer test-bearer-token-hardcoded-in-apk");
```

AI Expert Analysis

Exploitability: Low

Impact: The OkHttp logging interceptor logs the full request URL and an Authorization: Bearer token to logcat. Bearer tokens are directly replayable, so any exposure (rooted device, adb on debuggable builds, log aggregation) lets an attacker impersonate the user against the backend until the token expires. The URL may also contain query-string secrets.

Remediation: Never log Authorization headers. Use a redacting interceptor and disable body/header logging in release. // BEFORE (insecure) `Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer " + token);` // AFTER (secure) `HttpLoggingInterceptor logging = new HttpLoggingInterceptor(); logging.setLevel(BuildConfig.DEBUG ? HttpLoggingInterceptor.Level.BASIC : HttpLoggingInterceptor.Level.NONE); logging.redactHeader("Authorization"); logging.redactHeader("Cookie");` // And never concatenate the token into a log string.

Attack Chains: Token in logcat + debuggable build → extract bearer token → replay against API → Token in logcat + crash telemetry upload → token exposed to third party

Tool Confidence: HIGH reliability — the log statement and Bearer prefix are visible in app code. The token itself is redacted in the evidence, but the pattern of logging an Authorization header is the issue.

MASVS: MASVS-STORAGE-2 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check disputes this finding at `com/umermehmood1/maliciousapktest/network/LoggingInterceptorSample.java:22-24` ("Request request = chain.request(); Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer [REDACTED_TOKEN]"); return chain.proceed(request);"). The flagged log statement is a static debug message that concatenates only the request URL with a hardcoded literal string 'Bearer [REDACTED_TOKEN]'; no actual credential, token, or sensitive request data is read from the request or logged. The scanner's evidence shows the token placeholder is already redacted in source, so no sensitive value is emitted to logcat. Needs a reviewer decision.

Evidence

```
Log.d("FixtureHttp", "Calling " + request.url() + " with Bearer  
test-bearer-token-hardcoded-in-apk");
```

Finding 23: MASTG: The application writes sensitive data in application logs.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-532	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-2	
CVSS Score	6.8	
Location	3 affected locations (e.g. com/umermehmood1/maliciousapktest/logging/LogSamples.java)	

Risk Assessment

This medium-severity finding was identified in 3 affected locations (e.g. com/umermehmood1/maliciousapktest/logging/LogSamples.java). It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Log.w(TAG, "Using API endpoint https://auth.vulnerable-test.example/oauth/token");

AI Expert Analysis

Exploitability: Low

Impact: The app logs an authentication endpoint URL (https://auth.vulnerable-test.example/oauth/token) via Log.w. On Android 4.1+ app logs are not readable by other apps without root or a debug build, but they are captured by logcat, crash reporters, and device management tools. Exposure of auth endpoints and any future tokens/PII logged alongside them aids attackers in targeting the auth flow and can leak credentials.

Remediation: Remove sensitive logging from release builds and centralize logging behind a debug-only wrapper. Example: // BEFORE Log.w(TAG, "Using API endpoint https://auth.vulnerable-test.example/oauth/token"); // AFTER if (BuildConfig.DEBUG) { Log.d(TAG, "Using auth endpoint"); // no full URL, no secrets } Better: use a logging facade that is a no-op in release, and never log tokens, URLs with query params, or PII.

Attack Chains: Chains with weak TLS (index 3) — logged endpoint plus trust-all TLS lets an attacker target and intercept the auth flow → Chains with external storage writes (index 5) — combined leakage of endpoints and data increases reconnaissance value

Tool Confidence: Medium-reliability pattern match; evidence shows a concrete Log.w call with an auth URL in app code, so the logging is real. Actual sensitivity depends on what else is logged at runtime.

MASVS: MASVS-STORAGE-2 Scan-agent validation: PASS. Exploitability: low. Impact: high.

Evidence

Log.w(TAG, "Using API endpoint https://auth.vulnerable-test.example/oauth/token");

Finding 24: Deep link scheme can be claimed by another app		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-939	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	6.8	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Android does not assign ownership of a URI scheme. Any installed app may declare an intent filter for the same scheme, and when more than one app matches a URI the platform resolves it by its own rules — showing the user a chooser, or reusing whatever app they picked last time. Because the component below is reachable from other apps and is not protected by a permission, this app cannot distinguish a URI it produced from one a hostile app crafted.

- `com.umermehmood1.maliciousapktest.WebViewActivity` (activity) — `fixture://webview`

Whether that is exploitable depends on what the handler does with the URI — the reachability of the flow is what this finding establishes, not the impact.

AI Expert Analysis

Exploitability: Medium

Impact: The custom scheme 'fixture://webview' is not owned by the app in any enforceable way — any other installed app can register an intent filter for the same scheme and receive links the user or other apps intend for this app. Combined with the WebView sink, a malicious app can intercept or spoof deep-link navigation, potentially delivering attacker-controlled content into the app's WebView and phishing the user.

Remediation: Replace the custom scheme with an Android App Link using an https:// host you control, verified via asset links: `<intent-filter android:autoVerify="true"> <action android:name="android.intent.action.VIEW"> <category android:name="android.intent.category.DEFAULT"/> <category android:name="android.intent.category.BROWSABLE"/> <data android:scheme="https" android:host="app.example.com" android:pathPrefix="/webview"/> </intent-filter>` Publish /.well-known/assetlinks.json with the app's signing certificate SHA-256. In the receiving activity, validate the host and path against an allowlist before use, and never pass deep-link data directly to `loadUrl`.

Attack Chains: Malicious app registers `fixture://webview` and receives links meant for this app, then supplies a `javascript:` or attacker-hosted URL that reaches `WebViewActivity.loadUrl` (finding 6) → Deep-link interception used to phish credentials inside the trusted app UI

Tool Confidence: HIGH reliability — deterministic manifest inspection; the scheme string is directly evidenced.

MASVS: MASVS-PLATFORM-3 Scan-agent validation: PASS. Exploitability: medium. Impact: medium.

Recommendation

Prefer Android App Links over a custom scheme: declare an http/https intent filter for a domain you control and add `android:autoVerify="true"`, so the platform checks `assetlinks.json` and routes links to you alone.

If the scheme must stay custom, treat every incoming URI as untrusted input: protect the component with a signature-level `android:permission` so only apps you sign can reach it, constrain the filter with `android:host` and `android:pathPrefix` so it matches only your own URIs, and authorize the action the link triggers rather than assuming the link proves anything.

```
<!-- before: any app may claim this scheme -->
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<data android:scheme="myapp" />
</intent-filter>

<!-- after: only apps you sign, and only your own paths -->
<activity android:name=".DeepLinkActivity"
android:exported="true"
android:permission="com.example.myapp.permission.DEEPLINK">
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<data android:scheme="myapp" android:host="open"
android:pathPrefix="/v1/" />
</intent-filter>
</activity>

<permission android:name="com.example.myapp.permission.DEEPLINK"
android:protectionLevel="signature" />
```

Evidence

```
fixture://webview
```

Finding 25: Predictable PRNG: java.util.Random seeded with a constant yields a fully reproducible sequence — use SecureRandom and never set a fixed seed

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-330	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	6.3	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java:75	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java:75. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
return String.valueOf(new Random(1337L).nextInt(900000) + 100000);
```

AI Expert Analysis

Exploitability: Medium

Impact: java.util.Random seeded with the constant 1337L produces a fully reproducible sequence, so any value generated (e.g., a 6-digit code) is predictable by an attacker who knows the algorithm. If used for OTPs, tokens, or identifiers, an attacker can precompute all outputs and bypass verification or predict identifiers. This defeats the purpose of randomization for any security-relevant value.

Remediation: Use SecureRandom for all security-sensitive randomness and never set a fixed seed. Example: // BEFORE (insecure) return String.valueOf(new Random(1337L).nextInt(900000) + 100000); // AFTER (secure) SecureRandom secureRandom = new SecureRandom(); int code = 100000 + secureRandom.nextInt(900000); return String.valueOf(code); // For tokens, use SecureRandom with sufficient entropy (e.g., 128+ bits) and Base64-encode.

Attack Chains: Predictable OTP/code generation → bypass of verification flows → Predictable identifiers → enumeration or IDOR-style access to other users' resources

Tool Confidence: HIGH reliability — deterministic detection of Random with a constant seed; evidence directly shows the fixed seed.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check: confirmed at com/umermehmood1/maliciousapktest/security/CryptoSamples.java:74. Line 75 constructs a java.util.Random with a hardcoded constant seed (1337L) and immediately derives a value from it, so every invocation produces the identical, fully predictable output — a textbook predictable-PRNG defect (CWE-330). The method name predictableOtp and the 100000–999999 range indicate this value is intended as a one-time passcode, making the predictability security-relevant rather than cosmetic.

Evidence

```
return String.valueOf(new Random(1337L).nextInt(900000) + 100000);
```

Finding 26: WebView allows mixed content (MIXED_CONTENT_ALWAYS_ALLOW) — http:// resources load inside an https:// page and can be tampered with over the network

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-319	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	6.3	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:59	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:59. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.getSettings().setMixedContentMode(0);
```

AI Expert Analysis

Exploitability: Medium

Impact: MIXED_CONTENT_ALWAYS_ALLOW lets an https:// page load http:// subresources, which a network-position attacker can intercept and modify. Injected script or tampered content runs in the trusted page context, enabling session theft, credential capture, and UI redress. Requires the attacker to be on-path (hostile Wi-Fi, ARP spoofing, malicious proxy), so it is not remotely exploitable without network position.

Remediation: Use MIXED_CONTENT_NEVER_ALLOW (the secure default on API 21+) and ensure all subresources are https. // BEFORE (insecure)

```
webView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_ALWAYS_ALLOW); // AFTER  
(secure) webView.getSettings().setMixedContentMode(WebSettings.MIXED_CONTENT_NEVER_ALLOW); Also add  
a network security config that blocks cleartext: <network-security-config> <base-config  
cleartextTrafficPermitted="false"/> </network-security-config>
```

Attack Chains: Mixed content + disabled hostname verification → on-path attacker injects script into an https page → Mixed content + JS enabled → injected script reaches the JS bridge and native methods

Tool Confidence: HIGH reliability — explicit API call in app code; the constant value 0 maps to MIXED_CONTENT_ALWAYS_ALLOW.

MASVS: MASVS-NETWORK-1 Scan-agent validation: PASS. Exploitability: medium. Impact: high. Code check: confirmed at com/umermehmood1/maliciousapktest/WebViewActivity.java:59. Line 59 explicitly calls setMixedContentMode(0), which is MIXED_CONTENT_ALWAYS_ALLOW, permitting insecure http:// subresources to load within an https:// page and be tampered with in transit. The WebView is also configured with JavaScript enabled and a JavaScript bridge (lines 52 and 60), so a network attacker injecting content over cleartext could reach the exposed bridge interface, elevating the practical impact beyond simple content tampering.

Evidence

```
webView.getSettings().setMixedContentMode(0);
```

Finding 27: WebView file access enabled — potential path traversal		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-22	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	6.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:55	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:55. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.getSettings().setAllowFileAccess(true);
```

AI Expert Analysis

Exploitability: Low

Impact: `setAllowFileAccess(true)` lets the `WebView` load `file://` URLs from the local filesystem. If any navigation path is attacker-influenced, the `WebView` can be pointed at internal app files (`shared_prefs`, `databases`, `cached tokens`) and their contents rendered or exfiltrated. On its own it requires a way to control the loaded URL, but it materially widens the attack surface when combined with the `file-URL` and `universal-access` flags set in the same activity.

Remediation: Disable file access unless the app genuinely renders bundled local HTML, and if it does, load assets via `WebViewAssetLoader` instead of `file://` paths. `WebSettings s = webView.getSettings(); s.setAllowFileAccess(false); s.setAllowContentAccess(false);` // Preferred: serve bundled content over a virtual `https` origin `WebViewAssetLoader loader = new WebViewAssetLoader.Builder().addPathHandler("/assets/", new WebViewAssetLoader.AssetsPathHandler(this)).build(); webView.setWebViewClient(new WebViewAssetLoader.WebViewClientCompat(loader)); webView.loadUrl("https://[REDACTED]assets.androidplatform.net/assets/index.html");`

Attack Chains: File access + `setAllowFileAccessFromFileURLs(true)` → script in a `file://` page reads arbitrary app-private files → File access + attacker-controlled `loadUrl` → render internal config or token files

Tool Confidence: HIGH reliability — explicit API call in app code. Exploitability depends on whether an attacker can influence the loaded URL, which is not shown in the evidence.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium. Code check: confirmed at `com/umermehmood1/maliciousapktest/WebViewActivity.java:55`. The `WebView` enables file access (line 55) together with `file-URL` cross-origin access and universal access (lines 57-58) and JavaScript (line 52), and it loads an attacker-controllable URL from the intent extra `'targetUrl'` (lines 61-69), so a malicious page can read local files via `file://` and exfiltrate them, constituting a real path traversal / local file disclosure. The scanner's specific line is part of a genuinely dangerous configuration, not a false positive.

Evidence

```
webView.getSettings().setAllowFileAccess(true);
```

Finding 28: Exported activity		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-926	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.1	
Location	AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest. WebViewActivity	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest.WebViewActivity. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 3 analyzers: androguard, manifest_analyzer, mobsf] Activity 'com.umermehmood1.maliciousapktest.WebViewActivity' is exported without a protection permission. Any app can interact with it, potentially leading to data theft or functionality abuse.

AI Expert Analysis

Exploitability: The component is exported with no protection permission, so any installed app can launch it (startActivity/startService/queryIntentActivity). On Android < 12, intent-filter components default to exported=true.

Impact: Malicious apps can invoke protected functionality, spoof UI or read data via exported providers.

Remediation: Exported <activity> without permission — code-level fix Before (insecure): <activity android:name="com.umermehmood1.maliciousapktest.WebViewActivity" android:exported="true" /> After (secure): <activity android:name="com.umermehmood1.maliciousapktest.WebViewActivity" android:exported="false" /> (or, when other apps must invoke it, keep exported="true" with android:permission="<your.permission.signature>" declared with protectionLevel="signature")

Attack Chains: Craft malicious Intent → launch exported component → access protected data/functionality

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: medium. Impact: medium. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in androidmanifest

Evidence

AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest.WebViewActivity

Finding 29: Exported broadcast receiver		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-926	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.1	
Location	AndroidManifest.xml:receiver:com.umermehmood1.maliciousapktest.BootAuditReceiver	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:receiver:com.umermehmood1.maliciousapktest.BootAuditReceiver. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 3 analyzers: androguard, manifest_analyzer, mobsf] Broadcast Receiver 'com.umermehmood1.maliciousapktest.BootAuditReceiver' is exported without a protection permission. Any app can interact with it, potentially leading to data theft or functionality abuse.

AI Expert Analysis

Exploitability: The component is exported with no protection permission, so any installed app can launch it (startActivity/startService/queryIntentActivity). On Android < 12, intent-filter components default to exported=true.

Impact: Malicious apps can invoke protected functionality, spoof UI or read data via exported providers.

Remediation: Exported <receiver> without permission — code-level fix Before (insecure): <receiver android:name="com.umermehmood1.maliciousapktest.BootAuditReceiver" android:exported="true" /> After (secure): <receiver android:name="com.umermehmood1.maliciousapktest.BootAuditReceiver" android:exported="false" /> (or, when other apps must invoke it, keep exported="true" with android:permission="<your.permission.signature>" declared with protectionLevel="signature")

Attack Chains: Craft malicious Intent → launch exported component → access protected data/functionality

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in androidmanifest

Evidence

AndroidManifest.xml:receiver:com.umermehmood1.maliciousapktest.BootAuditReceiver

Finding 30: Exported activity		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-926	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.1	
Location	AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest.SecretsActivity	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest.SecretsActivity. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 3 analyzers: androguard, manifest_analyzer, mobsf] Activity 'com.umermehmood1.maliciousapktest.SecretsActivity' is exported without a protection permission. Any app can interact with it, potentially leading to data theft or functionality abuse.

AI Expert Analysis

Exploitability: The component is exported with no protection permission, so any installed app can launch it (startActivity/startService/queryIntentActivity). On Android < 12, intent-filter components default to exported=true.

Impact: Malicious apps can invoke protected functionality, spoof UI or read data via exported providers.

Remediation: Exported <activity> without permission — code-level fix Before (insecure): <activity android:name="com.umermehmood1.maliciousapktest.SecretsActivity" android:exported="true" /> After (secure): <activity android:name="com.umermehmood1.maliciousapktest.SecretsActivity" android:exported="false" /> (or, when other apps must invoke it, keep exported="true" with android:permission="<your.permission.signature>" declared with protectionLevel="signature")

Attack Chains: Craft malicious Intent → launch exported component → access protected data/functionality

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: medium. Impact: medium. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in androidmanifest

Evidence

AndroidManifest.xml:activity:com.umermehmood1.maliciousapktest.SecretsActivity

Finding 31: Exported component		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-926	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.1	
Location	AndroidManifest.xml:service:com.umermehmood1.maliciousapktest.AuditService	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:service:com.umermehmood1.maliciousapktest.AuditService. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 3 analyzers: androguard, manifest_analyzer, mobsf] Service 'com.umermehmood1.maliciousapktest.AuditService' is exported without a protection permission. Any app can interact with it, potentially leading to data theft or functionality abuse.

AI Expert Analysis

Exploitability: The component is exported with no protection permission, so any installed app can launch it (startActivity/startService/queryIntentActivity). On Android < 12, intent-filter components default to exported=true.

Impact: Malicious apps can invoke protected functionality, spoof UI or read data via exported providers.

Remediation: Exported <service> without permission — code-level fix Before (insecure): <service android:name="com.umermehmood1.maliciousapktest.AuditService" android:exported="true" /> After (secure): <service android:name="com.umermehmood1.maliciousapktest.AuditService" android:exported="false" /> (or, when other apps must invoke it, keep exported="true" with android:permission="<your.permission.signature>" declared with protectionLevel="signature")

Attack Chains: Craft malicious Intent → launch exported component → access protected data/functionality

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: medium. Impact: medium. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in androidmanifest

Evidence

AndroidManifest.xml:service:com.umermehmood1.maliciousapktest.AuditService

Finding 32: Permission: android.permission.WRITE_CALL_LOG		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] allows writing to (but not reading) the user's call log.. Allows an application to write (but not read) the user's call log data.

AI Expert Analysis

Exploitability: Low

Impact: WRITE_CALL_LOG is a signature|privileged permission that a normal third-party app cannot obtain at runtime; its presence in the manifest is either dead configuration or a strong indicator of an over-broad permission set. If the app were ever granted it (system/preinstalled context), it could tamper with the user's call history, a serious privacy and integrity violation. For a normal install it is non-functional but signals poor permission hygiene.

Remediation: Remove the declaration unless a documented, user-facing feature requires it: <uses-permission android:name="android.permission.WRITE_CALL_LOG" tools:node="remove"/> If call-log editing is genuinely required, it must be justified in the Play Console declaration and gated behind a runtime rationale dialog. Verify no code path calls ContentResolver.update/insert on CallLog.Calls.

Attack Chains: Chains with READ_PHONE_STATE and CALL_PHONE to build a full telephony-abuse profile if the app is ever preinstalled or granted privileged status

Tool Confidence: HIGH reliability — manifest permission declarations are deterministic; the finding is factual, though exploitability is limited because the permission is not grantable to normal apps.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.WRITE_CALL_LOG is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 33: Hardcoded iv detected		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-329	
OWASP Top 10	M9	
MASVS Category	MASVS-CODE	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java: 26	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java:26. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Potential hardcoded secret found in source code. Secrets should never be embedded in client-side code.

AI Expert Analysis

Exploitability: Medium

Impact: A fixed AES IV ('abcdef9876543210') is hardcoded in CryptoSamples.java. Reusing a static IV with CBC mode makes encryption deterministic, allowing an attacker who observes ciphertexts to detect identical plaintexts and enabling chosen-plaintext attacks that can recover data. If this code path protects real user data, confidentiality is broken.

Remediation: Generate a random IV per encryption and prepend it to the ciphertext; never hardcode it: // BEFORE (insecure) private static final String FIXED_AES_IV = "abcdef9876543210"; // AFTER (secure) byte[] iv = new byte[16]; new SecureRandom().nextBytes(iv); Cipher c = Cipher.getInstance("AES/GCM/NoPadding"); c.init(Cipher.ENCRYPT_MODE, key, new GCMParameterSpec(128, iv)); // store iv alongside ciphertext Prefer AES-GCM (authenticated) over CBC. Remove the constant and rotate any data encrypted with it.

Attack Chains: Static IV enables ciphertext comparison and chosen-plaintext recovery of encrypted user data → Combined with hardcoded key material elsewhere, full decryption of stored data

Tool Confidence: MEDIUM reliability — pattern match on a constant named FIXED_AES_IV. The class name (CryptoSamples) suggests a planted sample; confirm whether the constant is used in a live encryption path before finalising severity.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium. Code check: confirmed at com/umermehmood1/maliciousapktest/security/CryptoSamples.java:26. The constant FIXED_AES_IV is a static, hardcoded 16-byte IV that is directly used to initialize an AES/CBC cipher via IvParameterSpec in encryptWithFixedKey, so every encryption reuses the same IV. This is a genuine static-IV vulnerability (CWE-329) that breaks CBC semantic security by leaking equality of plaintext prefixes and enabling chosen-plaintext attacks.

Recommendation

Move secrets to a secure key management system (e.g., Android Keystore, server-side config, or encrypted SharedPreferences). Never hardcode credentials in source code.

Evidence

```
private static final String FIXED_AES_IV = "abcdef9876543210";
```

Finding 34: APK signed with a debug certificate		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-295	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	5.0	
Location	gomobsec-sample-app.apk	

Risk Assessment

This medium-severity finding was identified in gomobsec-sample-app.apk. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

The release package carries a debug signing identity (DN: C=US, O=Android, CN=Android Debug). Android debug keystores are generated per-machine with a well-known template and no secret discipline: anyone with the developer's public debug certificate can re-sign a modified build that installs as an upgrade over the shipped one on devices that trust it.

AI Expert Analysis

Exploitability: Low

Impact: The APK is signed with the public Android debug keystore (CN=Android Debug), whose private key is well-known and shared across all Android SDK installations. An attacker can forge a malicious update signed with the same debug key and, if the app is distributed outside a store that enforces signature checks, deliver it to users. Debug-signed builds also typically ship with `android:debuggable=true`, exposing the app to runtime debugging and memory inspection.

Remediation: Generate a dedicated release keystore and sign all production artifacts with it. Never distribute debug-signed APKs. `keytool -genkeypair -v -keystore release.jks -alias release -keyalg RSA -keysize 4096 -validity 10000` In `build.gradle`: `android { signingConfigs { release { storeFile file(System.getenv("KEYSTORE_PATH")) storePassword System.getenv("[REDACTED]") keyAlias System.getenv("KEY_ALIAS") keyPassword System.getenv("[REDACTED]") } } buildTypes { release { signingConfig signingConfigs.release; debuggable false } } }` Store the keystore and credentials in a secrets manager, not in the repository. Verify with: `apksigner verify --print-certs app-release.apk`

Attack Chains: Debug key + `debuggable=true` → attach debugger, dump memory, extract tokens → Debug key → sign forged update → distribute via sideloading to existing users

Tool Confidence: HIGH reliability — certificate DN is read directly from the APK signature block; deterministic and unambiguous.

MASVS: MASVS-RESILIENCE-1 Scan-agent validation: PASS. Exploitability: medium. Impact: high.

Recommendation

Sign release builds with the organization release keystore stored in the release pipeline's secret store, never with `~/.android/debug.keystore`. Rotate the release key if a debug-signed build was ever distributed.

Evidence

DN: C=US, O=Android, CN=Android Debug

Finding 35: WebView file-URL access enabled — JavaScript running in a file:// page can read other local files (setAllowFileAccessFromFileURLs(true)), a local file-theft vector. It defaults to false on API 16+; re

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-22	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:57	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:57. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.getSettings().setAllowFileAccessFromFileURLs(true);
```

AI Expert Analysis

Exploitability: Medium

Impact: `setAllowFileAccessFromFileURLs(true)` lets JavaScript running in a `file://` origin read other local files via XHR/fetch, bypassing the same-origin policy that normally isolates file URLs. Any script injection into a locally loaded page — or a malicious HTML file the app opens — can read app-private data such as `shared_prefs`, SQLite databases, and cached credentials and send it off-device. This is a direct local file-theft primitive.

Remediation: Remove the call; the secure default on API 16+ is false. If local HTML must run script, serve it from a virtual https origin via `WebViewAssetLoader` so normal same-origin rules apply. // BEFORE (insecure)
`webView.getSettings().setAllowFileAccessFromFileURLs(true);` // AFTER (secure)
`webView.getSettings().setAllowFileAccessFromFileURLs(false);` `webView.getSettings().setAllowFileAccess(false);`
`webView.getSettings().setAllowUniversalAccessFromFileURLs(false);`

Attack Chains: Combined with `setJavaScriptEnabled(true)` and `setAllowUniversalAccessFromFileURLs(true)` → full local file read and network exfiltration from a `file://` page → Combined with an intent that loads an attacker-supplied `file://` URL → script reads `/data/data/<pkg>/shared_prefs`

Tool Confidence: HIGH reliability — explicit API call in app code; the security consequence of this flag is well established.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium. Code check: confirmed at `com/umermehmood1/maliciousapktest/WebViewActivity.java:57`. Line 57 explicitly calls `setAllowFileAccessFromFileURLs(true)`, overriding the secure API 16+ default and enabling JavaScript in `file://` pages to read other local files; combined with JavaScript enabled (line 52) and file access enabled (line 55), the local file-theft vector is real. The setting is applied unconditionally to the `WebView` used to render content, so the scanner's finding is accurate.

Evidence

```
webView.getSettings().setAllowFileAccessFromFileURLs(true);
```

Finding 36: IV derived from a String via getBytes() — a string-derived IV is fixed/predictable; use SecureRandom.nextBytes() per message

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-329	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java:61	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java:61. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
cipher.init(1, secretKeySpec, new IvParameterSpec(bytes2));
```

AI Expert Analysis

Exploitability: Medium

Impact: The AES IV is derived from a String's bytes, making it fixed and predictable across encryptions. Reusing a static IV with CBC mode leaks information about plaintext (identical plaintext blocks produce identical ciphertext) and enables chosen-plaintext attacks that can recover data. This undermines the confidentiality guarantee of the encryption scheme.

Remediation: Generate a fresh cryptographically random IV per message with `SecureRandom` and prepend/transmit it alongside the ciphertext. Example: // BEFORE (insecure) `cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, new IvParameterSpec(bytes2));` // AFTER (secure) `byte[] iv = new byte[16]; new SecureRandom().nextBytes(iv); cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, new IvParameterSpec(iv));` // store/transmit iv with the ciphertext; on decrypt, read it back into `IvParameterSpec` // Preferred: use AES/GCM/NoPadding, which manages the nonce and adds integrity.

Attack Chains: Static IV + static key → identical ciphertext for identical plaintext, enabling pattern analysis and chosen-plaintext recovery → Combined with key-derivation finding → full offline decryption of stored data

Tool Confidence: HIGH reliability — deterministic detection of `IvParameterSpec` built from string bytes; evidence confirms the fixed-IV pattern.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: PASS. Exploitability: low. Impact: high. Code check: confirmed at `com/umermehmood1/maliciousapktest/security/CryptoSamples.java:59`. Line 59 derives the IV bytes directly from the static constant `FIXED_AES_IV` ("abcdef9876543210", line 26) via `getBytes()`, and line 61 passes those bytes to `IvParameterSpec` for AES/CBC encryption, so every encryption uses the same predictable IV with no `SecureRandom` involvement. This is a genuine CWE-329 violation (static/predictable IV), and combined with the hardcoded key on line 27 it makes the CBC encryption deterministic and leaks plaintext equality across messages.

Evidence

```
cipher.init(1, secretKeySpec, new IvParameterSpec(bytes2));
```

Finding 37: MASTG: The method "echo" is accessible from a WebView.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:24	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:24. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
@JavascriptInterface public final String echo(String value) {
    Intrinsics.checkNotNullParameter(value, "value");
    LogSamples.INSTANCE.logWebViewBridge(value);
    return value;
}
```

AI Expert Analysis

Exploitability: Medium

Impact: The echo method is exposed to JavaScript via @JavascriptInterface, allowing any script running in the WebView (including injected or remotely loaded content) to invoke it. While echo only reflects input, it also logs the value via logWebViewBridge, which can leak data into logcat and demonstrates an unrestricted JS-to-native bridge that expands the attack surface. If the WebView loads untrusted content, this bridge can be abused to interact with native functionality.

Remediation: Restrict the bridge to trusted origins, avoid logging sensitive values, and only expose methods that are strictly necessary. Example: // BEFORE (insecure) @JavascriptInterface public final String echo(String value) { LogSamples.INSTANCE.logWebViewBridge(value); return value; } // AFTER (secure) @JavascriptInterface public final String echo(String value) { // do not log user-controlled data return value; } // Additionally, only add the interface for trusted content and validate the loaded URL origin before enabling it.

Attack Chains: XSS in loaded WebView content → invoke echo bridge → data leakage via logs → Combined with JavaScript-enabled WebView and remote content loading → broader JS-to-native attack surface

Tool Confidence: MEDIUM reliability — pattern-based detection of @JavascriptInterface; evidence confirms the annotation, but exploitability depends on what content the WebView loads.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

```
@JavascriptInterface
public final String echo(String value) {
    Intrinsics.checkNotNullParameter(value, "value");
    LogSamples.INSTANCE.logWebViewBridge(value);
    return value;
}
```

Finding 38: MASTG: The method "getFakeDeviceId" is accessible from a WebView.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:31	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:31. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
@JavascriptInterface public final String getFakeDeviceId() { return "android_id"; }
```

AI Expert Analysis

Exploitability: Low

Impact: getFakeDeviceId is exposed to JavaScript, returning a static "android_id" value. The data itself is non-sensitive, but the exposed bridge method demonstrates an unrestricted JS-to-native interface that could be extended or chained. Impact is minimal on its own; the concern is the pattern of exposing native methods to WebView content.

Remediation: Remove unnecessary bridge methods and gate any required interface behind trusted-origin checks. Example: // Remove if not needed: // @JavascriptInterface public final String getFakeDeviceId() { return "android_id"; } // If required, only add via addJavascriptInterface after verifying the loaded origin is trusted.

Attack Chains: Exposed bridge methods + untrusted WebView content → enumeration of native capabilities

Tool Confidence: MEDIUM reliability — pattern-based detection; evidence confirms the annotation but the returned value is non-sensitive, so severity is low.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

```
@JavascriptInterface
public final String getFakeDeviceId() {
    return "android_id";
}
```

Finding 39: MASTG: The method "getHardcodedToken" is accessible from a WebView.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-798	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:36	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:36. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
@JavascriptInterface public final String getHardcodedToken() { return FakeSecrets.SAMPLE_JWT_TOKEN; }
```

AI Expert Analysis

Exploitability: Medium

Impact: getHardcodedToken exposes FakeSecrets.[REDACTED] to JavaScript, meaning a hardcoded JWT is reachable from any script in the WebView. If this token is a real credential, it can be extracted and used to impersonate the app or user against backend services, leading to unauthorized access. Even as a sample, it demonstrates a bridge that leaks secrets to web content.

Remediation: Never expose secrets through a JavaScript bridge; remove the method and eliminate hardcoded tokens. Example: // BEFORE (insecure) @JavascriptInterface public final String getHardcodedToken() { return FakeSecrets.[REDACTED]; } // AFTER (secure) // Remove the bridge method entirely. Obtain short-lived tokens from the backend // over authenticated TLS and keep them in memory/Keystore, never in source.

Attack Chains: XSS or malicious loaded content → call getHardcodedToken → exfiltrate JWT → replay against backend API → Chains with JavaScript-enabled WebView and remote content loading to deliver the malicious script

Tool Confidence: MEDIUM reliability — pattern-based detection; evidence confirms the @JavascriptInterface annotation and a hardcoded token reference, which is a genuine secret-exposure pattern.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: medium. Impact: medium.

Evidence

```
@JavascriptInterface
public final String getHardcodedToken() {
    return FakeSecrets.SAMPLE_JWT_TOKEN;
}
```

Finding 40: MASTG: WebView can load remote content from an endpoint or local file system in an unsafe manner.

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:49	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:49. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
WebView webView = new WebView(this);
```

AI Expert Analysis

Exploitability: Medium

Impact: The WebView is instantiated and may load remote or local content without visible restrictions on the supplied evidence. If it loads untrusted URLs or allows file access, an attacker could deliver malicious content that interacts with the exposed JavaScript bridge, leading to data theft or native method invocation. The evidence (new WebView(this)) alone does not prove unsafe loading, so impact depends on the configured URL and settings.

Remediation: Restrict loading to trusted HTTPS origins, disable file access, and validate URLs before loadUrl.

Example: `WebView webView = new WebView(this); WebSettings s = webView.getSettings();
s.setAllowFileAccess(false); s.setAllowContentAccess(false); s.setAllowFileAccessFromFileURLs(false);
s.setAllowUniversalAccessFromFileURLs(false); String url = "https://trusted.example.com"; if
(url.startsWith("https://trusted.example.com")) { webView.loadUrl(url); }`

Attack Chains: Unsafe remote content loading + JavaScript enabled + JS bridge → full JS-to-native exploitation →
Local file access + file:// URL → local file disclosure

Tool Confidence: MEDIUM reliability — pattern-based detection; the evidence only shows WebView instantiation, so manual review of loadUrl targets and WebSettings is required to confirm.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence only shows a WebView being instantiated; it does not demonstrate any unsafe loading of remote content or local files (no loadUrl, loadData, file access, or JavaScript settings are shown), so the reported issue is not substantiated by this snippet.

Evidence

```
WebView webView = new WebView(this);
```

Finding 41: MASTG: The application allows for JavaScript execution within a WebView instance.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapctest/WebViewActivity.java:52	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapctest/WebViewActivity.java:52. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.getSettings().setJavaScriptEnabled(true);
```

AI Expert Analysis

Exploitability: Medium

Impact: JavaScript is enabled in the WebView, which is required for the exposed @JavascriptInterface bridge to be reachable. If the WebView loads any untrusted or attacker-influenced content, this enables script execution that can invoke native bridge methods and exfiltrate data. On its own it is a configuration weakness; combined with the bridge methods it becomes a meaningful attack vector.

Remediation: Disable JavaScript unless strictly required, and if required, only enable it for trusted content with a strict Content-Security-Policy and origin validation. Example: // BEFORE (insecure)
 webView.getSettings().setJavaScriptEnabled(true); // AFTER (secure)
 webView.getSettings().setJavaScriptEnabled(false); // If JS is required, load only trusted HTTPS origins and set a restrictive CSP via WebViewClient.shouldInterceptRequest.

Attack Chains: JavaScript enabled + untrusted content + JS bridge → script invokes native methods (echo, getHardcodedToken) → Chains with remote content loading to deliver the malicious script

Tool Confidence: MEDIUM reliability — pattern-based detection; evidence directly confirms setJavaScriptEnabled(true), a deterministic configuration observation.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

```
webView.getSettings().setJavaScriptEnabled(true);
```

Finding 42: MASTG: The application exposes the Java class "WebBridge" through the WebView "webView".

Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java:60	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/WebViewActivity.java:60. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
webView.addJavaScriptInterface(new WebBridge(), "fixtureBridge");
```

AI Expert Analysis

Exploitability: Medium

Impact: A JavaScript interface named "fixtureBridge" (WebBridge) is registered on the WebView, exposing native methods to any JavaScript running in that WebView. If the WebView loads untrusted or MITM-influenced content, injected script can invoke bridge methods to access app functionality or data. On Android versions below API 17 this historically allowed reflection-based RCE; on modern targets the risk is unauthorized native capability invocation and data disclosure.

Remediation: Remove the JavaScript interface if not strictly required. If required, restrict it to trusted origins only and minimize the exposed API surface. Example: // BEFORE webView.addJavascriptInterface(new WebBridge(), "fixtureBridge"); // AFTER — only inject for trusted https origin, and annotate exposed methods
webView.setWebViewClient(new WebViewClient() { @Override public void onPageFinished(WebView v, String url) { if (url != null && url.startsWith("https://trusted.example.com/")) { v.addJavascriptInterface(new WebBridge(), "fixtureBridge"); } } }); // In WebBridge, expose only @JavascriptInterface-annotated, non-sensitive methods and validate all inputs.

Attack Chains: Chains with file access (index 0) — injected JS reads local files then calls bridge methods to exfiltrate → Chains with weak TLS (index 3) — MITM page content can invoke the bridge → Chains with the duplicate bridge finding (index 6) — same root cause, confirms exposure

Tool Confidence: Medium-reliability pattern match; evidence shows an explicit addJavascriptInterface call in app code, confirming the bridge exists. Exploitability hinges on whether untrusted content can be loaded into this WebView.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: medium. Impact: medium.

Evidence

```
webView.addJavascriptInterface(new WebBridge(), "fixtureBridge");
```

Finding 43: MASTG: Weak transport protocol version TLS is used.		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-295	
OWASP Top 10	M3	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/network/NetworkClient.java:62	

Risk Assessment

This medium-severity finding was identified in com/umermehmood1/maliciousapktest/network/NetworkClient.java:62. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

```
SSLContext sslContext = SSLContext.getInstance("TLS");
sslContext.init(null, new TrustManager[]{x509TrustManagerTrustAllCertificatesManager}, new SecureRandom());
```

AI Expert Analysis

Exploitability: High

Impact: NetworkClient initializes an SSLContext with a trust-all X509TrustManager, disabling certificate validation for all connections made through it. Any network attacker (hostile Wi-Fi, MITM proxy) can intercept and modify traffic, stealing credentials, tokens, and user data. This effectively nullifies TLS and is a direct path to account takeover and data breach.

Remediation: Remove the trust-all manager and use the platform default trust store. Example: // BEFORE
SSLContext sslContext = SSLContext.getInstance("TLS"); sslContext.init(null, new
TrustManager[]{x509TrustManagerTrustAllCertificatesManager}, new SecureRandom()); // AFTER — use default trust
SSLContext sslContext = SSLContext.getInstance("TLS"); sslContext.init(null, null, new SecureRandom()); // Or
simply rely on HttpURLConnection/OkHttp defaults. Additionally, enforce TLS 1.2+ and add certificate pinning via
network_security_config.xml or OkHttp CertificatePinner.

Attack Chains: Chains with WebView bridge/file access (index 0/1) — MITM-injected content can invoke native bridge and read local files → Chains with logged auth endpoint (index 2) — attacker knows the token endpoint and can intercept the token exchange → Chains with weak crypto (index 4) — intercepted ciphertext may be decryptable

Tool Confidence: Medium-reliability pattern match, but the evidence is unambiguous: a trust-all TrustManager is passed to SSLContext.init in app code. High confidence this is a genuine vulnerability.

MASVS: MASVS-NETWORK-1 Scan-agent validation: FAIL. Exploitability: medium. Impact: high. Issues: Reviewer model: The evidence shows SSLContext.getInstance("TLS"), which requests the default TLS protocol (TLS 1.2/1.3 on modern Android) rather than a weak or deprecated version like SSLv3, TLSv1, or TLSv1.1, so the reported weak transport protocol version is not demonstrated; the actual concern in this snippet is

Evidence

```
SSLContext sslContext = SSLContext.getInstance("TLS");  
sslContext.init(null, new TrustManager[]{x509TrustManagerTrustAllCertificatesManager}, new  
SecureRandom());
```

Finding 44: JavaScript bridge exposed in the app's own code		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	5.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This medium-severity finding was identified in `com/umermehmood1/maliciousapktest/WebViewActivity.java`. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

This file calls `addJavascriptInterface`, publishing a Java object to whatever the `WebView` renders. Every method on that object annotated with `@JavascriptInterface` becomes callable from the page's script — and on API levels before 17, every public method on it did. The exposure is not the call but its reach: the bridge is only as safe as the least trustworthy content the `WebView` ever renders. If anything the page displays can be influenced — a value from storage, a parameter, third-party HTML or an ad — then script in that content can invoke the bridged methods inside the app's own process.

AI Expert Analysis

Exploitability: Medium

Impact: Duplicate of the JavaScript bridge exposure: `addJavaScriptInterface` registers a native bridge reachable by any script in the `WebView`. This is the same root cause as index 1 and confirms the exposure. Injected script can invoke native methods, leading to data disclosure or unauthorized actions if untrusted content is loaded.

Remediation: Same fix as index 1: remove the interface or gate it to trusted https origins and expose only minimal, input-validated `@JavaScriptInterface` methods. Example: `webView.setWebViewClient(new WebViewClient() {
@Override public void onPageFinished(WebView v, String url) { if (url != null &&
url.startsWith("https://trusted.example.com/")) { v.addJavaScriptInterface(new WebBridge(), "fixtureBridge"); } } });`

Attack Chains: Chains with file access (index 0/9) — bridge plus file access enables local file exfiltration → Chains with weak TLS (index 3) — MITM content can invoke the bridge

Tool Confidence: Medium-reliability pattern match; evidence confirms `addJavaScriptInterface` in app code. This is a duplicate of index 1 and should be consolidated in the report.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Expose the minimum, and only on a `WebView` whose content the app controls end to end.

```
// before — one object, every annotated method reachable  
webView.addJavaScriptInterface(new AppBridge(), "Android");
```

```
// after — a narrow interface, and a content check  
if (isAppControlledContent(url)) { // https + your own host  
webView.addJavaScriptInterface(new ReadOnlyBridge(), "Android");  
}
```

```
private static class ReadOnlyBridge {  
@JavaScriptInterface // API 17+ exposes only these  
public String appVersion() { return BuildConfig.VERSION_NAME; }  
}
```

Never bridge anything that reads or writes files, runs commands, or returns tokens, and never bridge at all on a `WebView` that loads remote or external content.

Evidence

```
addJavaScriptInterface(...)
```

Finding 45: App can be installed on a vulnerable unpatched Android version 7.0, [minSdk=24]		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-1104	
OWASP Top 10	M8	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

This application can be installed on an older version of android that has multiple unfixed vulnerabilities. These devices won't receive reasonable security updates from Google. Support an Android version => 10, API 29 to receive reasonable security updates.

AI Expert Analysis

Exploitability: Low

Impact: minSdk=24 (Android 7.0) allows installation on unpatched devices with known platform vulnerabilities (e.g., Stagefright-era and later framework flaws). Users on old, unpatched OS versions are exposed to platform-level exploits that the app cannot mitigate. Business impact is increased attack surface and potential compliance concerns for regulated deployments.

Remediation: Raise minSdkVersion to a supported, patched baseline (e.g., API 26/Android 8.0 or higher, ideally API 29+). Example in build.gradle: // BEFORE minSdkVersion 24 // AFTER minSdkVersion 29 Document the supported OS matrix and enforce it in the Play Console device catalog. If legacy support is mandatory, add runtime checks and warn users on outdated OS versions.

Attack Chains: Chains with weak TLS (index 3) — older devices may lack modern TLS/cipher support, worsening interception risk → Chains with WebView issues (index 0/1) — older WebView engines have more known RCE vulnerabilities

Tool Confidence: High-reliability deterministic check reading AndroidManifest.xml; minSdk=24 is factual. Severity is a policy/risk decision rather than a direct code vulnerability.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

AndroidManifest.xml

Finding 46: Permission: android.permission.CAMERA		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] take pictures and videos. Allows application to take pictures and videos with the camera. This allows the application to collect images that the camera is seeing at any time.

AI Expert Analysis

Exploitability: Low

Impact: The app declares CAMERA permission. If not strictly required for core functionality, this expands the privacy attack surface and may violate Google Play data-safety/policy requirements, risking app rejection or removal. A compromised app could silently capture imagery.

Remediation: Remove the permission if unused. If required, request at runtime only when the feature is invoked and document the purpose. Example (manifest): `<uses-permission android:name="android.permission.CAMERA" android:maxSdkVersion="..." />` Runtime: `if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) { ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.CAMERA}, RC_CAMERA); }` Declare the data type in the Play Data Safety form.

Attack Chains: Over-broad permission + compromised app → covert capture of user imagery

Tool Confidence: HIGH reliability manifest parsing. Severity depends on whether the permission is functionally justified; verify against feature set.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.CAMERA is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 47: Permission: android.permission.RECORD_AUDIO		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] record audio. Allows application to access the audio record path.

AI Expert Analysis

Exploitability: Low

Impact: The app declares RECORD_AUDIO permission. If unjustified, this is a significant privacy concern and a Play policy risk; a compromised or malicious build could record ambient audio. Excessive permissions also increase user distrust and regulatory exposure (GDPR/CCPA).

Remediation: Remove if unused. If required, request at runtime with a clear rationale and only while the recording feature is active. Example: if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) { ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.RECORD_AUDIO}, RC_AUDIO); } Stop recording and release the mic when the feature is backgrounded; disclose in the privacy policy and Play Data Safety form.

Attack Chains: Over-broad permission + malicious update → ambient audio surveillance

Tool Confidence: HIGH reliability manifest parsing. Confirm functional justification before finalizing severity.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.RECORD_AUDIO is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 48: Permission: android.permission.ACCESS_FINE_LOCATION		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] fine (GPS) location. Access fine location sources, such as the Global Positioning System on the phone, where available. Malicious applications can use this to determine where you are and may consume additional battery power.

AI Expert Analysis

Exploitability: Low

Impact: The app declares ACCESS_FINE_LOCATION. Precise location is highly sensitive PII; if not essential, this creates privacy and regulatory risk and may fail Play policy review. A compromised app could track user movements.

Remediation: Prefer ACCESS_COARSE_LOCATION unless precise location is essential. Request at runtime only when needed and stop updates when the feature is inactive. Example: <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" /> // only if truly needed: <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" /> Use FusedLocationProviderClient with appropriate priority and remove location updates in onPause(). Disclose in the privacy policy and Data Safety form.

Attack Chains: Precise location + compromised app → user tracking / stalking → Location data leak via logs (finding 6) → PII exposure

Tool Confidence: HIGH reliability manifest parsing. Severity depends on functional necessity; downgrade to COARSE if precise location is not required.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.ACCESS_FINE_LOCATION is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 49: Permission: android.permission.ACCESS_COARSE_LOCATION		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] coarse (network-based) location. Access coarse location sources, such as the mobile network database, to determine an approximate phone location, where available. Malicious applications can use this to determine approximately where you are.

AI Expert Analysis

Exploitability: Low

Impact: Declaring ACCESS_COARSE_LOCATION grants the app access to approximate device location (~city-block accuracy). If the app does not genuinely require location for a core feature, this expands the privacy attack surface and increases regulatory exposure (GDPR/CCPA data-minimization obligations) and Play Store data-safety declaration risk. On its own it is a permission declaration, not an exploitable flaw.

Remediation: Remove the permission from AndroidManifest.xml unless a documented feature requires it. If location is required, prefer coarse over fine and gate runtime requests behind a just-in-time rationale: <!-- AndroidManifest.xml -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" android:maxSdkVersion="..."
/> // Runtime (only when feature is used) if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_COARSE_LOCATION), REQ_LOC) }

Attack Chains: Combined with ACCESS_BACKGROUND_LOCATION, enables continuous location tracking without foreground interaction → Chains with any network-exfiltration finding to leak user geolocation to a remote endpoint

Tool Confidence: HIGH reliability — manifest permission declarations are deterministic and directly verifiable in AndroidManifest.xml.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.ACCESS_COARSE_LOCATION` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 50: Permission: android.permission.ACCESS_BACKGROUND_LOCATION		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] access location in background. Allows an app to access location in the background.

AI Expert Analysis

Exploitability: Low

Impact: ACCESS_BACKGROUND_LOCATION is among the most privacy-sensitive Android permissions and permits location collection while the app is not in the foreground. Unjustified use is a common trigger for Play Store policy rejection and regulatory scrutiny, and materially raises the impact of any data-exfiltration flaw. Exploitation requires the user to grant the permission and the app to actively collect and transmit location.

Remediation: Remove ACCESS_BACKGROUND_LOCATION unless a clearly justified, user-visible background feature exists. If retained, request it separately after foreground location is granted and document the use case in the Play Console declaration: `<!-- Only if a documented background feature exists --> <uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" /> // Request separately, with explicit user education`
`ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.ACCESS_BACKGROUND_LOCATION), REQ_BG_LOC)`

Attack Chains: Pairs with ACCESS_COARSE_LOCATION/FINE_LOCATION for persistent tracking → Amplifies impact of insecure network transport or hardcoded endpoints by continuously streaming location

Tool Confidence: HIGH reliability — deterministic manifest declaration; presence is certain, business justification is what requires review.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.ACCESS_BACKGROUND_LOCATION` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 51: Permission: android.permission.READ_CONTACTS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] read contact data. Allows an application to read all of the contact (address) data stored on your phone. Malicious applications can use this to send your data to other people.

AI Expert Analysis

Exploitability: Low

Impact: READ_CONTACTS exposes the user's entire address book, a high-value PII dataset. If the app has no contact-related feature, this is excessive privilege that increases breach blast radius and violates data-minimization principles. Exploitation requires the user to grant the runtime permission and the app to read and transmit contact data.

Remediation: Remove READ_CONTACTS from the manifest unless a documented feature requires it. If required, read only necessary fields and never persist or transmit the full contact set: // Only if feature requires it — request at point of use ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.READ_CONTACTS), REQ_CONTACTS) // Query minimal projection val projection = arrayOf(ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME, ContactsContract.CommonDataKinds.Phone.NUMBER)

Attack Chains: Chains with WRITE_CONTACTS to enable contact tampering/social-engineering injection → Combined with network exfiltration, enables mass PII harvesting

Tool Confidence: HIGH reliability — manifest permission declaration is deterministic.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.READ_CONTACTS` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 52: Permission: android.permission.WRITE_CONTACTS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] write contact data. Allows an application to modify the contact (address) data stored on your phone. Malicious applications can use this to erase or modify your contact data.

AI Expert Analysis

Exploitability: Low

Impact: WRITE_CONTACTS allows the app to create, modify, or delete the user's contacts. This is rarely justified and, if abused, enables contact poisoning and social-engineering attacks against the user's network. Requires user-granted runtime permission and active app code to perform writes.

Remediation: Remove WRITE_CONTACTS unless a documented feature requires it. If required, restrict writes to app-owned contacts via a sync adapter and never modify user-owned entries: // Prefer a sync adapter scoped to your account

```
val ops = ArrayList<ContentProviderOperation>()
ops.add(ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
    .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, "com.example.app") .build())
```

Attack Chains: Pairs with READ_CONTACTS for full read/write control of address book → Enables injection of attacker-controlled contact entries to facilitate phishing

Tool Confidence: HIGH reliability — deterministic manifest declaration.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.WRITE_CONTACTS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 53: Permission: android.permission.READ_CALENDAR		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] read calendar events. Allows an application to read all of the calendar events stored on your phone. Malicious applications can use this to send your calendar events to other people.

AI Expert Analysis

Exploitability: Low

Impact: READ_CALENDAR exposes the user's calendar events, including meeting details, attendees, and locations — sensitive PII and potentially business-confidential information. Excessive if no calendar feature exists. Requires user-granted runtime permission and app code to read and transmit events.

Remediation: Remove READ_CALENDAR unless a documented feature requires it. If required, query only needed columns and avoid persisting event data: `ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.READ_CALENDAR), REQ_CAL)` `val projection = arrayOf(CalendarContract.Events.TITLE, CalendarContract.Events.DTSTART)`

Attack Chains: Chains with WRITE_CALENDAR for calendar manipulation → Combined with exfiltration, leaks meeting/attendee intelligence

Tool Confidence: HIGH reliability — deterministic manifest declaration.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.READ_CALENDAR is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 54: Permission: android.permission.WRITE_CALENDAR		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] add or modify calendar events and send emails to guests. Allows an application to add or change the events on your calendar, which may send emails to guests. Malicious applications can use this to erase or modify your calendar events or to send emails to guests.

AI Expert Analysis

Exploitability: Low

Impact: WRITE_CALENDAR allows the app to create, modify, or delete calendar events. Abuse could inject fraudulent meetings or disrupt schedules. Requires user-granted runtime permission and active write code.

Remediation: Remove WRITE_CALENDAR unless a documented feature requires it. If required, write only to app-owned calendars and validate all event data server-side:

```
val values = ContentValues().apply {
    put(CalendarContract.Events.DTSTART, startMillis)
    put(CalendarContract.Events.TITLE, sanitizedTitle)
}
contentResolver.insert(CalendarContract.Events.CONTENT_URI, values)
```

Attack Chains: Pairs with READ_CALENDAR for full calendar control → Enables injection of malicious meeting invites to facilitate phishing

Tool Confidence: HIGH reliability — deterministic manifest declaration.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.WRITE_CALENDAR is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 55: Permission: android.permission.READ_SMS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] read SMS or MMS. Allows application to read SMS messages stored on your phone or SIM card. Malicious applications may read your confidential messages.

AI Expert Analysis

Exploitability: Low

Impact: READ_SMS grants access to all SMS messages, including one-time passwords, banking alerts, and private conversations. This is one of the highest-risk permissions and a primary enabler of OTP interception. Requires user-granted runtime permission and app code to read and transmit messages.

Remediation: Remove READ_SMS unless a documented, policy-compliant feature requires it (Google Play restricts SMS permissions to default SMS handlers and specific use cases). If OTP autofill is the goal, use the SMS Retriever API instead: // Use SMS Retriever API — no READ_SMS permission required

```
val client = SmsRetriever.getClient(this)
client.startSmsRetriever()
```

Attack Chains: Chains with RECEIVE_SMS to intercept OTPs and bypass MFA → Combined with network exfiltration, enables account takeover of banking/email accounts

Tool Confidence: HIGH reliability — deterministic manifest declaration; high business risk given OTP interception potential.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.READ_SMS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 56: Permission: android.permission.SEND_SMS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] send SMS messages. Allows application to send SMS messages. Malicious applications may cost you money by sending messages without your confirmation.

AI Expert Analysis

Exploitability: Low

Impact: SEND_SMS allows the app to send SMS messages, potentially incurring premium-rate charges or sending spam on the user's behalf. This is a high-risk permission and a common enabler of toll-fraud and smishing. Requires user-granted runtime permission and active send code.

Remediation: Remove SEND_SMS unless a documented feature requires it. If messaging is required, use a server-side SMS gateway rather than device SMS, or use SmsManager only for user-initiated, visible sends: // Only for explicit user-initiated sends

```
val smsManager = getSystemService(SmsManager::class.java)
smsManager.sendTextMessage(number, null, message, null, null)
```

Attack Chains: Chains with READ_SMS/RECEIVE_SMS for full SMS control → Enables premium-rate toll fraud and smishing campaigns

Tool Confidence: HIGH reliability — deterministic manifest declaration.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.SEND_SMS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 57: Dangerous permission: android.permission.RECEIVE_SMS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

receive SMS. Allows application to receive and process SMS messages. Malicious applications may monitor your messages or delete them without showing them to you.

AI Expert Analysis

Exploitability: Low

Impact: RECEIVE_SMS allows the app to intercept incoming SMS messages in real time, including OTPs and 2FA codes. This is a critical enabler of account-takeover attacks and is tightly restricted by Google Play policy. Requires user-granted runtime permission and a registered BroadcastReceiver.

Remediation: Remove RECEIVE_SMS unless the app is a default SMS handler. For OTP flows, use the SMS Retriever API or the SMS User Consent API, which require no SMS permission: // SMS User Consent API — no RECEIVE_SMS permission
 val client = SmsRetriever.getClient(this) client.startSmsUserConsent(null)

Attack Chains: Chains with READ_SMS to harvest OTPs and bypass MFA → Combined with exfiltration, enables full account takeover of third-party services

Tool Confidence: HIGH reliability — deterministic manifest declaration; flagged as dangerous permission, consistent with Android's protection level.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence only restates the generic description of the RECEIVE_SMS permission and provides no manifest excerpt, permission declaration, or code reference showing that this application actually requests or uses android.permission.RECEIVE_SMS.

Recommendation

Verify that android.permission.RECEIVE_SMS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 58: Permission: android.permission.READ_CALL_LOG		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] grants read access to the user's call log.. Allows an application to read the user's call log.

AI Expert Analysis

Exploitability: Low

Impact: READ_CALL_LOG exposes the user's call history, including phone numbers, timestamps, and durations — sensitive PII. This permission is heavily restricted by Google Play and is a strong privacy indicator. Requires user-granted runtime permission and app code to read and transmit call data.

Remediation: Remove READ_CALL_LOG unless the app is a default dialer/phone handler with a documented, policy-compliant use case. If call-related functionality is needed, use the CallScreeningService or CallRedirectionService APIs instead of reading the call log: // Use CallScreeningService instead of READ_CALL_LOG
class MyCallScreeningService : CallScreeningService() { override fun onScreenCall(details: Call.Details) { /* screen without call log access */ } }

Attack Chains: Combined with READ_CONTACTS, enables mapping of social graph and communication patterns → Chains with exfiltration to leak communication metadata for targeted attacks

Tool Confidence: HIGH reliability — deterministic manifest declaration.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.READ_CALL_LOG is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 59: Permission: android.permission.CALL_PHONE		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] directly call phone numbers. Allows the application to call phone numbers without your intervention. Malicious applications may cause unexpected calls on your phone bill. Note that this does not allow the application to call emergency numbers.

AI Expert Analysis

Exploitability: Medium

Impact: CALL_PHONE lets the app place calls without user confirmation of the dialer, enabling premium-rate/SIM-toll fraud and silent outbound calls if the permission is abused or the app is trojanized. It also expands the attack surface for social-engineering and billing fraud against the user.

Remediation: Prefer an ACTION_DIAL intent that hands control to the user: Intent i = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:" + number)); startActivity(i); If direct calling is truly required, remove CALL_PHONE and use ACTION_CALL only after an explicit user tap, validating the number against an allowlist. Delete the manifest entry if unused: <uses-permission android:name="android.permission.CALL_PHONE" tools:node="remove"/>

Attack Chains: Combines with WRITE_CALL_LOG to conceal fraudulent calls from the user's history → Pairs with READ_PHONE_STATE to harvest device/telephony identifiers for fraud

Tool Confidence: HIGH reliability — deterministic manifest check; severity is appropriate but real-world exploitability depends on whether the app actually invokes ACTION_CALL.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.CALL_PHONE` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 60: Permission: android.permission.READ_PHONE_STATE		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] read phone state and identity. Allows the application to access the phone features of the device. An application with this permission can determine the phone number and serial number of this phone, whether a call is active, the number that call is connected to and so on.

AI Expert Analysis

Exploitability: Medium

Impact: READ_PHONE_STATE exposes IMEI/MEID (legacy), subscriber ID, SIM state, and network info, which are persistent device identifiers usable for tracking and cross-app correlation. This is a privacy and regulatory concern (GDPR/CCPA) and a common enabler of device fingerprinting.

Remediation: Remove READ_PHONE_STATE and use non-identifying alternatives: // Instead of IMEI/subscriber ID String id = Settings.Secure.getString(getContentResolver(), Settings.Secure.ANDROID_ID); // For network state use ConnectivityManager, not TelephonyManager If telephony state is genuinely needed, request only the narrowest API and document the purpose. Remove the manifest line if unused.

Attack Chains: Feeds device fingerprinting that links sessions across reinstalls → Combines with media permissions to build a rich user profile

Tool Confidence: HIGH reliability — manifest declaration is factual; impact assessment depends on whether identifiers are actually read and transmitted.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.READ_PHONE_STATE` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 61: Permission: android.permission.BODY_SENSORS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] grants access to body sensors, such as heart rate.. Allows an application to access data from sensors that the user uses to measure what is happening inside his/her body, such as heart rate.

AI Expert Analysis

Exploitability: Low

Impact: BODY_SENSORS grants access to heart rate, step, and other physiological data — highly sensitive health information. Requesting it without a clear feature need is a privacy overreach and a regulatory risk under health-data regulations. Exploitation requires the user to grant the runtime permission.

Remediation: Remove the permission if no health feature exists: `<uses-permission android:name="android.permission.BODY_SENSORS" tools:node="remove"/>` If a fitness feature is intended, use the Health Connect / Sensor APIs with a runtime rationale and minimize collection. Never persist raw sensor data without encryption and explicit consent.

Attack Chains: Combines with READ_MEDIA_* to aggregate health and personal media into a single exfiltration target

Tool Confidence: HIGH reliability — deterministic manifest check; low exploitability because it is a runtime-gated dangerous permission.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that android.permission.BODY_SENSORS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 62: Dangerous permission: android.permission.POST_NOTIFICATIONS		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

allows an app to post notifications.. Allows an app to post notifications

AI Expert Analysis

Exploitability: Low

Impact: POST_NOTIFICATIONS (Android 13+) is a runtime permission; requesting it is normal for apps that notify users, but combined with other permissions it can enable notification-based phishing or spam. On its own the risk is low and mostly a UX/consent concern.

Remediation: Request POST_NOTIFICATIONS only at the moment a notification feature is first used, with a rationale: if (Build.VERSION.SDK_INT >= 33) { requestPermissions(new String[]{Manifest.permission.POST_NOTIFICATIONS}, REQ); } Do not request at cold start. Remove the declaration if the app does not post notifications.

Attack Chains: Can be abused with CALL_PHONE/telephony permissions for notification-driven social engineering

Tool Confidence: HIGH reliability — deterministic manifest check; severity is low in practice as this is a standard, user-consented permission.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: POST_NOTIFICATIONS is a normal runtime permission for user-facing notifications, not a dangerous permission, and the evidence provides no manifest declaration or code showing misuse, so the reported MEDIUM finding is not substantiated.

Recommendation

Verify that android.permission.POST_NOTIFICATIONS is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 63: Permission: android.permission.READ_EXTERNAL_STORAGE		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.0	
Location	AndroidManifest.xml:uses-permission:android.permission.READ_EXTERNAL_STORAGE	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:uses-permission:android.permission.READ_EXTERNAL_STORAGE. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] The app requests android.permission.READ_EXTERNAL_STORAGE. This permission grants access to sensitive user data or device capabilities. Verify it is strictly necessary and that user consent is obtained.

AI Expert Analysis

Exploitability: Medium

Impact: READ_EXTERNAL_STORAGE (legacy, pre-scoped-storage) grants broad read access to shared storage, exposing photos, documents, and downloads of the user. This is a significant privacy exposure and a common data-leak vector if the app mishandles file paths.

Remediation: Adopt scoped storage and remove the broad permission. Use the Storage Access Framework or MediaStore: // Pick a file via SAF Intent i = new Intent(Intent.ACTION_OPEN_DOCUMENT); i.addCategory(Intent.CATEGORY_OPENABLE); i.setType("*/*"); startActivityForResult(i, REQ); Remove READ_EXTERNAL_STORAGE from the manifest; on API 33+ use READ_MEDIA_* only for specific media types.

Attack Chains: Combines with WRITE_EXTERNAL_STORAGE to read and tamper with user files → Enables exfiltration of documents if paired with a network sink

Tool Confidence: HIGH reliability — deterministic manifest check; impact is real but mitigated on modern Android by scoped storage.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

AndroidManifest.xml:uses-permission:android.permission.READ_EXTERNAL_STORAGE

Finding 64: Permission: android.permission.WRITE_EXTERNAL_STORAGE		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
CVSS Score	5.0	
Location	AndroidManifest.xml:uses-permission:android.permission.WRITE_EXTERNAL_STORAGE	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml:uses-permission:android.permission.WRITE_EXTERNAL_STORAGE. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] The app requests android.permission.WRITE_EXTERNAL_STORAGE. This permission grants access to sensitive user data or device capabilities. Verify it is strictly necessary and that user consent is obtained.

AI Expert Analysis

Exploitability: Medium

Impact: WRITE_EXTERNAL_STORAGE allows the app to modify or delete files in shared storage, risking data integrity and enabling planting of malicious files that other apps may consume. It is deprecated on modern Android but still a broad capability on older devices.

Remediation: Remove the permission and write to app-scoped storage or use MediaStore for shared media: ContentValues v = new ContentValues(); v.put(MediaStore.Images.Media.DISPLAY_NAME, name); Uri uri = getContentResolver().insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, v); Delete the manifest entry: <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" tools:node="remove"/>

Attack Chains: Pairs with READ_EXTERNAL_STORAGE for full read/write access to user files → Can plant files consumed by other apps (confused-deputy style)

Tool Confidence: HIGH reliability — deterministic manifest check; severity appropriate but reduced on scoped-storage devices.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Evidence

AndroidManifest.xml:uses-permission:android.permission.WRITE_EXTERNAL_STORAGE

Finding 65: Dangerous permission: android.permission.READ_MEDIA_IMAGES		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

allows reading image files from external storage.. Allows an application to read image files from external storage.

AI Expert Analysis

Exploitability: Medium

Impact: READ_MEDIA_IMAGES (Android 13+) grants access to all user photos, a highly sensitive data class. Requesting it without a clear feature need is a privacy overreach and a regulatory risk; if the app exfiltrates images it becomes a serious data breach.

Remediation: Use the Photo Picker to avoid broad access: `ActivityResultLauncher<PickVisualMediaRequest> picker = registerForActivityResult(new ActivityResultContracts.PickVisualMedia(), uri -> { /* use uri */ }); picker.launch(new PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.ImageOnly));` Remove READ_MEDIA_IMAGES from the manifest if the picker suffices.

Attack Chains: Combines with READ_MEDIA_VIDEO/AUDIO to harvest all user media → Feeds exfiltration if paired with a network sink or insecure storage

Tool Confidence: HIGH reliability — deterministic manifest check; exploitability gated by runtime consent.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a generic restatement of what READ_MEDIA_IMAGES means and contains no manifest excerpt, permission declaration, or app-specific context showing the permission is actually requested or that the app's functionality does not justify it.

Recommendation

Verify that android.permission.READ_MEDIA_IMAGES is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 66: Dangerous permission: android.permission.READ_MEDIA_VIDEO		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

allows reading video files from external storage.. Allows an application to read video files from external storage.

AI Expert Analysis

Exploitability: Medium

Impact: READ_MEDIA_VIDEO grants access to all user videos, a sensitive data class that may contain private or identifying content. Broad access without feature justification is a privacy overreach and regulatory exposure.

Remediation: Prefer the Photo Picker with VideoOnly type: picker.launch(new PickVisualMediaRequest(ActivityResultContracts.PickVisualMedia.VideoOnly)); Otherwise request READ_MEDIA_VIDEO at the point of use with a rationale, and remove the manifest entry if unused.

Attack Chains: Combines with READ_MEDIA_IMAGES/AUDIO for full media harvesting → Enables exfiltration of private video content

Tool Confidence: HIGH reliability — deterministic manifest check; runtime-gated so real exploitability is moderate.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a generic restatement of what READ_MEDIA_VIDEO permits and contains no manifest excerpt, permission declaration, or app-specific context showing the permission is actually requested or used by this application.

Recommendation

Verify that android.permission.READ_MEDIA_VIDEO is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 67: Dangerous permission: android.permission.READ_MEDIA_AUDIO		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

allows reading audio files from external storage.. Allows an application to read audio files from external storage.

AI Expert Analysis

Exploitability: Medium

Impact: READ_MEDIA_AUDIO grants access to all user audio files, which may include voice recordings and private conversations. Broad access without justification is a privacy overreach and a regulatory risk.

Remediation: Use the document picker or a scoped MediaStore query for audio, and remove the broad permission: `Intent i = new Intent(Intent.ACTION_OPEN_DOCUMENT); i.setType("audio/*"); startActivityForResult(i, REQ);` Delete READ_MEDIA_AUDIO from the manifest if not strictly required.

Attack Chains: Combines with READ_MEDIA_IMAGES/VIDEO to harvest all user media → Feeds exfiltration of voice recordings if paired with a network sink

Tool Confidence: HIGH reliability — deterministic manifest check; exploitability gated by runtime consent.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a generic restatement of what READ_MEDIA_AUDIO does and contains no manifest excerpt, permission declaration, or application context showing the permission is actually requested or that the app's functionality does not justify it.

Recommendation

Verify that android.permission.READ_MEDIA_AUDIO is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 68: Permission: android.permission.MANAGE_EXTERNAL_STORAGE		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

[Confirmed by 2 analyzers: androguard, mobsf] Allows an application a broad access to external storage in scoped storage. Allows an application a broad access to external storage in scoped storage. Intended to be used by few apps that need to manage files on behalf of the users.

AI Expert Analysis

Exploitability: Low

Impact: MANAGE_EXTERNAL_STORAGE (All Files Access) grants the app broad read/write access to shared external storage, far beyond what a normal app requires. If the app is compromised or maliciously updated, it can enumerate and exfiltrate user documents, photos, and other apps' exported files. Google Play restricts this permission to a narrow set of app categories, so its presence also signals likely policy rejection and reduced user trust.

Remediation: Remove MANAGE_EXTERNAL_STORAGE unless the app is a file manager, backup, or antivirus tool. Use scoped storage APIs instead: `<!-- AndroidManifest.xml --> <uses-permission android:name="android.permission.MANAGE_EXTERNAL_STORAGE" tools:node="remove"/> // Prefer MediaStore / SAF for file access`
`val intent = Intent(Intent.ACTION_OPEN_DOCUMENT).apply {`
`addCategory(Intent.CATEGORY_OPENABLE) type = "*/*" }`
`startActivityForResult(intent, REQ_PICK_FILE)`
 If All Files Access is genuinely required, gate it behind Settings.ACTION_MANAGE_ALL_FILES_ACCESS_PERMISSION and document the justification for Play review.

Attack Chains: Combined with REQUEST_INSTALL_PACKAGES: broad storage read enables staging a malicious APK from external storage before install → Combined with cleartext SharedPreferences: attacker with storage access reads cached tokens and AWS keys

Tool Confidence: HIGH reliability — manifest permission declarations are deterministic; the permission is definitively present.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium.

Recommendation

Verify that `android.permission.MANAGE_EXTERNAL_STORAGE` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

Finding 69: Dangerous permission: android.permission.REQUEST_INSTALL_PACKAGES		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

Allows an application to request installing packages.. Malicious applications can use this to try and trick users into installing additional malicious packages.

AI Expert Analysis

Exploitability: Medium

Impact: REQUEST_INSTALL_PACKAGES allows the app to prompt installation of arbitrary APKs. If the app downloads update or plugin packages over an unvalidated channel, an attacker who can influence that channel (MITM, compromised CDN, or a malicious deep link) can deliver a trojanized APK that the user is socially engineered into installing. This is a classic dropper/side-loading vector and a common malware trait.

Remediation: Remove the permission if the app does not perform in-app updates: `<uses-permission android:name="android.permission.REQUEST_INSTALL_PACKAGES" tools:node="remove"/>` If self-update is required, use the Play Core in-app update API instead of manual APK installs: `val appUpdateManager = AppUpdateManagerFactory.create(context) appUpdateManager.appUpdateInfo.addOnSuccessListener { info -> if (info.updateAvailability() == UpdateAvailability.UPDATE_AVAILABLE) { appUpdateManager.startUpdateFlowForResult(info, AppUpdateType.IMMEDIATE, activity, REQ_UPDATE) } }` If manual install is unavoidable, verify the APK signature against a pinned certificate before invoking the installer and enforce HTTPS with certificate pinning for the download.

Attack Chains: Combined with MANAGE_EXTERNAL_STORAGE: stage malicious APK on external storage, then trigger install prompt → Combined with SYSTEM_ALERT_WINDOW: overlay a fake 'Update required' dialog to drive install consent → Combined with cleartext API endpoint in SharedPreferences: redirect update check to attacker-controlled server

Tool Confidence: HIGH reliability — manifest permission declaration is deterministic.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a generic description of what REQUEST_INSTALL_PACKAGES permits and contains no manifest excerpt, permission declaration, or application-specific context showing the permission is actually present or used in this app.

Recommendation

Verify that `android.permission.REQUEST_INSTALL_PACKAGES` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

AndroidManifest.xml

Finding 70: Dangerous permission: android.permission.SYSTEM_ALERT_WINDOW		
Likelihood	Impact	Risk
Possible	Medium	Medium
Severity	MEDIUM	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-250	
OWASP Top 10	M6	
MASVS Category	MASVS-PRIVACY	
MASVS Control	MASVS-PRIVACY-1	
CVSS Score	5.0	
Location	AndroidManifest.xml	

Risk Assessment

This medium-severity finding was identified in AndroidManifest.xml. It is considered possible to exploit, with a medium potential impact; the overall risk to the application is rated medium.

Description

display system-level alerts. Allows an application to show system-alert windows. Malicious applications can take over the entire screen of the phone.

AI Expert Analysis

Exploitability: Medium

Impact: SYSTEM_ALERT_WINDOW permits drawing overlays on top of other apps. This is the core enabler of tapjacking and credential-harvesting overlay attacks: a malicious or compromised build can render a fake login or payment screen over a legitimate banking or password-manager app and capture user input. It also enables click-jacking of permission dialogs and is heavily abused by Android banking trojans.

Remediation: Remove the permission unless the app is a legitimate overlay use case (e.g., accessibility tool, floating widget): `<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" tools:node="remove"/>` If overlays are required, restrict them to non-sensitive contexts and set `FLAG_NOT_TOUCHABLE` / `FLAG_NOT_FOCUSABLE` where appropriate, and never render input fields in an overlay: `val params = WindowManager.LayoutParams( WindowManager.LayoutParams.WRAP_CONTENT, WindowManager.LayoutParams.WRAP_CONTENT, WindowManager.LayoutParams.TYPE_APPLICATION_OVERLAY, WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE or WindowManager.LayoutParams.FLAG_NOT_TOUCH_MODAL, PixelFormat.TRANSLUCENT )` Also set `android.filterTouchesWhenObscured="true"` on sensitive views to block tapjacking.

Attack Chains: Combined with REQUEST_INSTALL_PACKAGES: overlay a fake update prompt to coerce APK installation → Combined with cleartext auth token in logcat/storage: overlay a fake re-auth screen to harvest fresh credentials

Tool Confidence: HIGH reliability — manifest permission declaration is deterministic.

MASVS: MASVS-PRIVACY-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence only restates the generic description of the SYSTEM_ALERT_WINDOW permission and provides no manifest excerpt, permission declaration, or application context demonstrating that this app actually requests or uses it.

Recommendation

Verify that `android.permission.SYSTEM_ALERT_WINDOW` is necessary for the app's core functionality and that user consent is obtained before using it.

Evidence

`AndroidManifest.xml`

LOW Findings

Finding 71: X Missing: Anti-debugging		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	3.0	

Risk Assessment

This low-severity finding was identified within the application. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No anti-debugging implementation found in app code, across 181 first-party source files.

AI Expert Analysis

Exploitability: Low

Impact: Without anti-debugging, an attacker can attach a debugger (e.g., via JDWP or Frida) to inspect runtime state, extract secrets, and modify control flow. This significantly reduces the effort required for dynamic reverse engineering and bypass of client-side security controls.

Remediation: Add debugger detection and disable debuggable builds in release. Example: `// In Application.onCreate if ((applicationInfo.flags and ApplicationInfo.FLAG_DEBUGGABLE) != 0 || Debug.isDebuggerConnected() || Debug.waitForDebugger()) { // report and terminate sensitive flows } // In build.gradle release buildType: debuggable false, minifyEnabled true`

Attack Chains: Attach debugger → inspect decrypted data in memory → extract keys/tokens → Combine with missing root detection (finding 4) → Frida hooking of TLS and auth logic

Tool Confidence: MEDIUM reliability; static absence detection is credible but native anti-debug (ptrace) may not be captured. Validate dynamically.

MASVS: MASVS-RESILIENCE-3 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 65%

False Positive Reason: Missing anti-debugging is a defense-in-depth gap — absence-based, low severity, not a real exploit

Evidence

Static analysis of 181 first-party source files – covering detection libraries and hand-rolled indicators such as su-binary and build-tag checks – found no anti-debugging.

Finding 72: Dependency CVE: Gson — CVE-2022-25647		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-937	
OWASP Top 10	M2	
MASVS Category	MASVS-CODE	
MASVS Control	MASVS-CODE-1	
CVSS Score	3.0	
Location	dependency:Gson	

Risk Assessment

This low-severity finding was identified in dependency:Gson. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

Gson is present in this build but its version could not be determined, so CVE-2022-25647 (Deserialization DoS — affects < 2.8.9) could not be confirmed or excluded. Confirm the Gson version in use; 2.8.9 or later is not affected. Reachability: the application's own code references this library, so the vulnerable component is in use.

AI Expert Analysis

Exploitability: A known-vulnerability version of a library is bundled. If the CVE is network-exploitable (CVSS AV:N) the app inherits the risk.

Impact: The app ships with a public CVE; attackers target the vulnerable library's call paths.

Remediation: Update Gson to version 2.8.9 or later to remediate CVE-2022-25647.

Attack Chains: App calls vulnerable library function → public exploit → compromise

Tool Confidence: Deterministic tool result (high confidence)

MASVS: MASVS-CODE-1 Scan-agent validation: FAIL. Exploitability: low. Impact: low. Issues: Reviewer model: The evidence only states that Gson is present and its version could not be determined, so CVE-2022-25647 (which affects versions below 2.8.9) is neither confirmed nor excluded; an undetermined version is not evidence that the vulnerable version is in use.

Evidence

Deserialization DoS — affects < 2.8.9. The version of Gson in this build could not be determined, so it is not known whether CVE-2022-25647 applies.

Finding 73: No screen sets FLAG_SECURE		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-200	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-3	
CVSS Score	3.0	
Location	application-wide	

Risk Assessment

This low-severity finding was identified in application-wide. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No window in the application sets `WindowManager.LayoutParams.FLAG_SECURE`. Screens showing credentials, OTPs, card data or account information can therefore be captured by screenshots, screen recorders and overlays, and appear in the recent-apps thumbnail. Sensitive screens should opt out of capture.

AI Expert Analysis

Exploitability: Low

Impact: Without `FLAG_SECURE`, any screen containing sensitive data can be captured by screenshots, screen recording, or the recent-apps thumbnail. On a shared or compromised device this exposes user data to other apps with media access or to anyone viewing the recents switcher. Business impact is data leakage and potential regulatory exposure if PII is displayed.

Remediation: Apply `FLAG_SECURE` to activities that display sensitive data: // In `onCreate()` of sensitive activities `getWindow().setFlags( WindowManager.LayoutParams.FLAG_SECURE, WindowManager.LayoutParams.FLAG_SECURE)`; Or declaratively per-activity via a theme, and clear sensitive fields in `onPause()`. Apply selectively — do not blanket-apply to non-sensitive screens where it harms usability.

Attack Chains: Screenshot capture of credentials or tokens displayed in the `WebView` (finding 6) exfiltrates session material → Recent-apps thumbnail leakage of sensitive screens

Tool Confidence: HIGH reliability — absence of `FLAG_SECURE` confirmed across 41 first-party source files. LOW severity is appropriate; impact is conditional on what the app displays.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: PASS. Exploitability: low. Impact: low.

Recommendation

Call `getWindow().setFlags(LayoutParams.FLAG_SECURE, LayoutParams.FLAG_SECURE)` in `onCreate` of every activity that displays sensitive data (login, OTP, payment, account).

Evidence

No `FLAG_SECURE` reference in 41 first-party source files.

Finding 74: Draw-over-other-apps (overlay) permission requested		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-1021	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-3	
CVSS Score	3.0	
Location	AndroidManifest.xml	

Risk Assessment

This low-severity finding was identified in AndroidManifest.xml. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

The app requests SYSTEM_ALERT_WINDOW, letting it draw on top of other apps. This is the overlay half of a tapjacking / StrandHogg attack; if the app is compromised or the permission is broader than needed, it can paint over other apps' UI. Surface it as attack surface and confirm it is genuinely required.

AI Expert Analysis

Exploitability: Low

Impact: SYSTEM_ALERT_WINDOW allows the app to draw over other apps. If abused — or if the permission is granted to a repackaged build — it enables overlay/tapjacking attacks against other apps and can obscure security prompts. In this app it is likely unnecessary, and its presence increases the permission footprint requested from users.

Remediation: Remove the permission unless a documented feature requires it: <!-- AndroidManifest.xml — remove --> <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/> If overlays are genuinely required, gate the feature behind an explicit user opt-in, use TYPE_APPLICATION_OVERLAY only, and pair it with setFilterTouchesWhenObscured(true) on sensitive views (see finding 5).

Attack Chains: Overlay permission combined with missing filterTouchesWhenObscured (finding 5) enables tapjacking against the app's own sensitive controls → Overlay used to obscure permission or payment confirmation dialogs

Tool Confidence: HIGH reliability — permission is explicitly declared in the manifest. Severity LOW is appropriate given no evidence the permission is actually used.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: PASS. Exploitability: low. Impact: low.

Recommendation

Remove SYSTEM_ALERT_WINDOW if the app does not need to draw overlays. If it does, keep overlays minimal and ensure sensitive own-app screens set FLAG_SECURE and filterTouchesWhenObscured so they cannot be overlaid or tapjacked.

Evidence

```
uses-permission android.permission.SYSTEM_ALERT_WINDOW
```

Finding 75: Direct SQL execution (check for injection)		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-89	
OWASP Top 10	M4	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	3.0	
Location	com/umermehmood1/maliciousapktest/db/AuditDatabase.smali	

Risk Assessment

This low-severity finding was identified in com/umermehmood1/maliciousapktest/db/AuditDatabase.smali. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

Direct SQL execution (check for injection) in AuditDatabase.smali

AI Expert Analysis

Exploitability: Low

Impact: AuditDatabase executes SQL directly. If any query is built by string concatenation with user- or network-supplied input, an attacker who can influence that input can inject SQL — reading, modifying, or deleting audit records, or corrupting the local database. Because the database is local, exploitation requires the attacker to control input reaching the query (e.g. via an exported component or a malicious deep link), so real-world exploitability is conditional.

Remediation: Use parameterized queries / `SQLiteStatement` bindings for all dynamic values; never concatenate input into SQL. Example: // BEFORE (injectable) `db.rawQuery("SELECT * FROM audit WHERE id = " + id + "", null);` // AFTER (safe) `db.rawQuery("SELECT * FROM audit WHERE id = ?", new String[]{ id });` For inserts use `SQLiteStatement` with `bindString/bindLong`. Validate and whitelist any identifier (table/column) that cannot be parameterized.

Attack Chains: Exported component (index 8) → pass crafted input → SQL injection in AuditDatabase → SQL injection → tamper with or delete audit trail → anti-forensics / repudiation → Chains with external storage (index 4) if the DB is world-readable

Tool Confidence: MEDIUM reliability — pattern match on `rawQuery/execSQL` usage. This is a 'check for injection' signal; confirm whether any call site concatenates untrusted input before rating as a confirmed injection.

MASVS: MASVS-STORAGE-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Severity mismatch: reported LOW, suggested MEDIUM; Reviewer model: The evidence is only a generic restatement of the finding title with no actual SQL statement, query construction, or user-input flow shown, so it does not demonstrate that any injectable SQL execution exists in `AuditDatabase.smali`. ---

False Positive Confidence: 55%

False Positive Reason: Raw SQL in app's AuditDatabase — could be parameterized; needs manual review to confirm injection risk

Evidence

```
com/umermehmood1/maliciousapktest/db/AuditDatabase.smali
```

Finding 76: MASTG: The application uses deprecated cryptographic algorithms.		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-327	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	3.0	
Location	com/umermehmood1/maliciousapktest/security/CryptoSamples.java:55	

Risk Assessment

This low-severity finding was identified in com/umermehmood1/maliciousapktest/security/CryptoSamples.java:55. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

AI Expert Analysis

Exploitability: Low

Impact: CryptoSamples uses AES/CBC/PKCS5Padding. CBC mode without authentication is malleable (padding-oracle and bit-flipping attacks) and PKCS5Padding is a legacy designation. If used to protect sensitive data, an attacker with ciphertext access could tamper with or decrypt data. Impact depends on whether this code path protects real secrets.

Remediation: Use authenticated encryption (AES-GCM) with a random 12-byte IV per message. Example: // BEFORE Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding"); // AFTER Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding"); cipher.init(Cipher.ENCRYPT_MODE, key, new GCMParameterSpec(128, iv)); // iv = 12 random bytes, stored with ciphertext If CBC must remain for compatibility, add an HMAC-SHA256 over IV+ciphertext (encrypt-then-MAC) and use PKCS7Padding.

Attack Chains: Chains with weak TLS (index 3) — intercepted CBC ciphertext may be attacked offline or via padding oracle → Chains with external storage (index 5) — encrypted blobs written to shared storage are exposed to tampering

Tool Confidence: Medium-reliability pattern match; evidence shows a concrete Cipher.getInstance("AES/CBC/PKCS5Padding") call. The algorithm is real; severity depends on whether it protects sensitive data versus test/fixture code.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence shows AES/CBC/PKCS5Padding, which is a current, non-deprecated cipher construction (AES is not a deprecated algorithm), so the reported deprecated-cryptography finding does not apply to this code. ---

False Positive Confidence: 55%

False Positive Reason: AES/CBC/PKCS5Padding is not a deprecated algorithm per se; the finding is a generic pattern match, though CBC mode without integrity is a legitimate design concern — needs manual review of IV handling.

Evidence

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

Finding 77: Raw SQL query (smali)		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-89	
OWASP Top 10	M4	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	3.0	
Location	com/umermehmood1/maliciousapktest/db/AuditDatabase.smali	

Risk Assessment

This low-severity finding was identified in `com/umermehmood1/maliciousapktest/db/AuditDatabase.smali`. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

Raw SQL query (smali) in `AuditDatabase.smali`

AI Expert Analysis

Exploitability: Low

Impact: Second raw-SQL signal in `AuditDatabase (smali)`. Same risk profile as index 6: if untrusted input reaches the query, an attacker can read or alter audit records. Duplicate of index 6 at a different call site — treat as one remediation effort across the class.

Remediation: Same as index 6 — replace all `rawQuery/execSQL` string building with parameterized queries and `SQLiteStatement` bindings. Example: `// BEFORE db.execSQL("DELETE FROM audit WHERE user=" + user + "");` `// AFTER db.delete("audit", "user = ?", new String[]{ user });` Audit the whole `AuditDatabase` class in one pass and add a lint rule banning string concatenation in SQL calls.

Attack Chains: Exported component (index 8) → crafted input → SQL injection → SQL injection → audit-trail tampering → repudiation of security events → Chains with external storage (index 4) for direct DB file access

Tool Confidence: MEDIUM reliability — pattern match on raw SQL usage. Confirm concatenation of untrusted input at this specific call site before treating as confirmed injection.

MASVS: MASVS-STORAGE-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a restatement of the finding title with no actual smali code, query string, or context, so it does not demonstrate that a raw SQL query exists or is exploitable in this application. ---

False Positive Confidence: 55%

False Positive Reason: Duplicate raw SQL query finding on same file — same manual review needed, likely benign but unconfirmed

Evidence

com/umermehmood1/maliciousapktest/db/AuditDatabase.smali

Finding 78: Exported components detected		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-926	
OWASP Top 10	M1	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
CVSS Score	3.0	
Location	AndroidManifest.xml	

Risk Assessment

This low-severity finding was identified in AndroidManifest.xml. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

App has exported components that may be accessed by other apps

AI Expert Analysis

Exploitability: Medium

Impact: The manifest declares exported components. Any exported activity, service, receiver, or provider without a permission guard can be invoked by any other app on the device. Depending on what the component does, this enables unauthorized actions, data disclosure, or as a pivot into the SQL and storage issues above. HIGH tool trust means the export flags are confirmed in the manifest; the actual risk depends on which components are exported and whether they validate input.

Remediation: Set `android:exported="false"` on every component that does not need to be invoked by other apps. For components that must be exported, enforce a signature-level permission and validate all incoming data. Example: `<!-- BEFORE --> <activity android:name=".AuditActivity" android:exported="true"/> <!-- AFTER --> <activity android:name=".AuditActivity" android:exported="false"/>` For a required exported provider: `<provider android:name=".AuditProvider" android:exported="true" android:permission="com.umermehmood1.maliciousapktest.permission.INTERNAL" android:grantUriPermissions="false"/>` Review each exported component and add input validation on all entry points.

Attack Chains: Malicious co-resident app → invoke exported component → trigger privileged action or data read → Exported component → crafted input → SQL injection (index 6, 7) → Exported component → read files via external storage (index 4)

Tool Confidence: HIGH reliability — deterministic manifest parse. The export flags are confirmed; residual work is to enumerate which components are exported and whether they are permission-guarded.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence is only a generic restatement of the finding with no manifest excerpt, component names, intent filters, or export attributes, so it does not demonstrate that any component is actually exported in this app. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in androidmanifest

Recommendation

Review exported components. Remove exports where not needed.

Evidence

AndroidManifest.xml

Finding 79: Exported component		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-926	
OWASP Top 10	M4	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
CVSS Score	3.0	
Location	54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml	

Risk Assessment

This low-severity finding was identified in 54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

Exported component in AndroidManifest.xml

AI Expert Analysis

Exploitability: Medium

Impact: The manifest declares at least one exported component without an apparent permission guard. Any co-installed app on the device can invoke it, potentially triggering privileged functionality or state changes on behalf of the user. The concrete risk depends on which component is exported and what it does, which the evidence does not specify.

Remediation: Enumerate every component with `android:exported="true"` in `AndroidManifest.xml` and justify each one. For components that must not be reachable by other apps, set `android:exported="false"`. Where cross-app invocation is genuinely required, protect it with a signature-level custom permission: `<permission android:name="com.umermehmood1.maliciousapktest.permission.INTERNAL" android:protectionLevel="signature"/>`
`<activity android:name=".InternalActivity" android:exported="true"`
`android:permission="com.umermehmood1.maliciousapktest.permission.INTERNAL"/>` For components that must stay exported, validate the caller (check `CallingPermission` / `getCallingPackage`) and validate all input before acting on it.

Attack Chains: Malicious co-installed app invokes the exported component with crafted extras, which then flow into `WebViewActivity.loadUrl` (finding 6) to achieve arbitrary URL loading → Exported component combined with the claimable `'fixture://webview'` deep link (finding 2) allows a third-party app to drive the `WebView` with attacker-chosen content

Tool Confidence: MEDIUM reliability — manifest pattern match. The evidence field is empty, so the specific component and its protection level were not confirmed; verify by inspecting the exported attribute and any `android:permission` on each component before finalising severity.

MASVS: MASVS-PLATFORM-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The evidence only restates the finding title without identifying any specific exported component, its declaration, or its intent-filter/permission context, so it does not demonstrate that an exported component actually exists in this application. ---

False Positive Confidence: 75%

False Positive Reason: Matches known FP pattern: 'exported' in `androidmanifest`

Evidence

54f84037-e5d8-44e5-9d8a-30893a8807c8/apktool/AndroidManifest.xml

Finding 80: x Missing: Anti-emulator		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	3.0	

Risk Assessment

This low-severity finding was identified within the application. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No anti-emulator implementation found in app code, across 181 first-party source files.

AI Expert Analysis

Exploitability: Low

Impact: Absence of anti-emulator detection lowers the cost for an attacker to run the app in an instrumented emulator, enabling dynamic analysis, automated abuse, and credential stuffing at scale. Business impact is moderate: it facilitates reverse engineering and fraud automation but does not by itself expose user data.

Remediation: Add emulator detection as a defense-in-depth signal, but rely on server-side Play Integrity attestation for enforcement. Example: `fun isProbablyEmulator(): Boolean = Build.FINGERPRINT.startsWith("generic") || Build.FINGERPRINT.contains("test-keys") || Build.MODEL.contains("Emulator") || Build.HARDWARE.contains("goldfish") || Build.PRODUCT.contains("sdk")` // Send signal to backend; do not hard-block client-side only

Attack Chains: Run app in emulator → bypass device-bound controls → automate account creation/fraud → Combine with missing root detection (finding 4) and anti-debugging (finding 6) → full dynamic analysis environment

Tool Confidence: MEDIUM reliability; static absence-of-control detection across 181 first-party files is reasonably reliable but may miss obfuscated or dynamically loaded checks. Confirm with dynamic testing.

MASVS: MASVS-RESILIENCE-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 65%

False Positive Reason: Missing anti-emulator is a hardening gap, not a vulnerability; absence-based finding on 181 files, low actionability

Evidence

Static analysis of 181 first-party source files – covering detection libraries and hand-rolled indicators such as su-binary and build-tag checks – found no anti-emulator.

Finding 81: x Missing: Root detection		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	3.0	

Risk Assessment

This low-severity finding was identified within the application. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No root detection implementation found in app code, across 181 first-party source files.

AI Expert Analysis

Exploitability: Low

Impact: Without root detection, the app runs on rooted devices where an attacker has full filesystem and memory access, enabling extraction of stored secrets, hooking of security checks, and tampering with app logic. This undermines the confidentiality of any locally stored data and the integrity of client-side controls.

Remediation: Implement root detection as a signal and enforce sensitive operations server-side. Example: `fun isRooted(): Boolean { val paths = listOf("/system/app/Superuser.apk", "/sbin/su", "/system/xbin/su", "/system/bin/su") return paths.any { File(it).exists() } || Build.TAGS?.contains("test-keys") == true } // Report to backend and gate high-risk actions; never rely solely on client check`

Attack Chains: Root device → dump app private storage → recover tokens/keys → Root + Frida → hook certificate pinning and bypass TLS → Combine with credential-in-log (finding 0) → harvest bearer tokens from logcat on rooted device

Tool Confidence: MEDIUM reliability; static absence detection is credible but may miss native or dynamically loaded root checks. Validate dynamically.

MASVS: MASVS-RESILIENCE-1 Scan-agent validation: FAIL. Exploitability: low. Impact: medium. Issues: Reviewer model: The absence of root detection is a hardening gap, not a vulnerability, and the evidence only shows no detection code was found across first-party sources without demonstrating any actual security impact or exploitability in this application. ---

False Positive Confidence: 65%

False Positive Reason: Missing root detection is a defense-in-depth gap, not an exploitable flaw — informational hardening item

Evidence

Static analysis of 181 first-party source files – covering detection libraries and hand-rolled indicators such as su-binary and build-tag checks – found no root detection.

Finding 82: x Missing: Integrity verification		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	3.0	

Risk Assessment

This low-severity finding was identified within the application. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No integrity verification implementation found in app code, across 181 first-party source files.

AI Expert Analysis

Exploitability: Low

Impact: No integrity verification means a repackaged or tampered APK can run without detection, allowing an attacker to distribute a modified build containing malware or logic changes under the app's identity. This risks brand damage, user data compromise, and fraudulent transactions.

Remediation: Verify APK signature at runtime and validate with Play Integrity API server-side. Example: `val sig = packageManager.getPackageInfo(packageName, PackageManager.GET_SIGNING_CERTIFICATES).signingInfo.apkContentsSigners val expected = "<SHA-256 of signing cert>" val actual = MessageDigest.getInstance("SHA-256").digest(sig[0].toByteArray()).joinToString("") { "%02x".format(it) } if (actual != expected) { /* fail closed / report */ } // Prefer server-verified Play Integrity verdict for enforcement`

Attack Chains: Repackage APK with malicious payload → distribute via sideloading → harvest credentials → Tampered build bypasses client checks → interacts with backend as legitimate client

Tool Confidence: MEDIUM reliability; static absence detection is plausible but runtime signature checks may exist in native code not covered by the scan. Confirm with dynamic instrumentation.

MASVS: MASVS-RESILIENCE-2 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 65%

False Positive Reason: Missing integrity verification is a hardening recommendation, not a confirmed vulnerability

Evidence

Static analysis of 181 first-party source files – covering detection libraries and hand-rolled indicators such as su-binary and build-tag checks – found no integrity verification.

Finding 83: x No code obfuscation detected		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
CVSS Score	3.0	

Risk Assessment

This low-severity finding was identified within the application. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

No ProGuard, R8, or DexGuard obfuscation found. Source code may be easily reverse-engineered.

AI Expert Analysis

Exploitability: Low

Impact: Lack of code obfuscation makes reverse engineering trivial: class, method, and field names remain readable, exposing business logic, API endpoints, and security control locations. This accelerates discovery of other vulnerabilities (e.g., hardcoded secrets) and lowers the barrier to repackaging and IP theft.

Remediation: Enable R8/ProGuard minification and obfuscation for release builds. Example (build.gradle):

```
android {
    buildTypes {
        release {
            minifyEnabled true
            shrinkResources true
            proguardFiles
                getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
} // Keep only necessary reflection/entry points in proguard-rules.pro
```

Attack Chains: Decompile unobfuscated APK → locate hardcoded token (finding 0) and logging code (finding 8) quickly → Identify security checks by name → targeted bypass via hooking

Tool Confidence: MEDIUM reliability; the 1% short-name metric is a reasonable heuristic but R8 may still be partially applied. Confirm by inspecting DEX for retained descriptive identifiers.

MASVS: MASVS-RESILIENCE-4 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 70%

False Positive Reason: No obfuscation detected is a hardening observation, not a vulnerability — informational only

Evidence

No obfuscation marker and no short-name pattern in 181 first-party source files (1% with 1-2 character names).

Finding 84: Code: Android Webview		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	REVIEW	
Assurance	Needs analyst review	
CWE	CWE-749	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-2	
CVSS Score	3.0	
Location	com/umermehmood1/maliciousapktest/WebViewActivity.java	
Reachability	Invokable by another app — Activity, via fixture://webview (with no permission required)	

Risk Assessment

This low-severity finding was identified in `com/umermehmood1/maliciousapktest/WebViewActivity.java`. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole. Affected files (1): `com/umermehmood1/maliciousapktest/WebViewActivity.java`

AI Expert Analysis

Exploitability: Medium

Impact: WebViewActivity instantiates a WebView (CWE-749 class). WebViews are a common attack surface: if the activity loads untrusted URLs, enables JavaScript, or is exported, attackers can inject content. This finding is a general indicator that must be correlated with the specific settings (file access, JS bridge) found in the same file.

Remediation: Harden the WebView configuration. Example: `WebSettings s = webView.getSettings(); s.setJavaScriptEnabled(true); // only if required s.setAllowFileAccess(false); s.setAllowContentAccess(false); s.setAllowFileAccessFromFileURLs(false); s.setAllowUniversalAccessFromFileURLs(false); s.setMixedContentMode(WebSettings.MIXED_CONTENT_NEVER_ALLOW); webView.setWebViewClient(new WebViewClient() { @Override public boolean shouldOverrideUrlLoading(WebView v, WebResourceRequest r) { return !r.getUrl().getHost().equals("trusted.example.com"); // block untrusted hosts } });` Ensure the activity is not exported unless required.

Attack Chains: Chains with file access (index 0/9) and JS bridge (index 1/6) — the WebView is the delivery vector for those exploits → Chains with weak TLS (index 3) — MITM content loaded into the WebView

Tool Confidence: High-reliability detection that a WebView exists, but this is a generic indicator with no specific vulnerable setting in the evidence. Treat as context for the concrete WebView findings rather than a standalone issue.

MASVS: MASVS-PLATFORM-2 Scan-agent validation: FAIL. Exploitability: low. Impact: low. Issues: Reviewer model: The evidence provides only a file path with no code content, so it does not demonstrate any insecure WebView configuration such as JavaScript enabled, addJavascriptInterface, or loading untrusted URLs. ---

False Positive Confidence: 51%

False Positive Reason: Generic "WebView present" detection is informational inventory, not a vulnerability by itself; only actionable if combined with unsafe settings — low actionability.

Recommendation

A WebView that executes JavaScript and also exposes a Java object through addJavascriptInterface lets anything that can reach the page call into the app — including a page loaded from a URL the app did not intend to allow.

```
// before
webView.getSettings().setJavaScriptEnabled(true);
webView.addJavascriptInterface(new Bridge(), "app");
```

```
// after — JavaScript only where it is needed
webView.getSettings().setJavaScriptEnabled(false);
```

Where a bridge is unavoidable, restrict it to a first-party origin and annotate every exposed method with @JavascriptInterface.

Evidence

```
com/umermehmood1/maliciousapktest/WebViewActivity.java
```

Finding 85: Crypto: MD5 hash used		
Likelihood	Impact	Risk
Unlikely	Low	Low
Severity	LOW	
Status	OPEN	
Assurance	QA-validated	
CWE	CWE-328	
OWASP Top 10	M10	
MASVS Category	MASVS-CRYPTO	
MASVS Control	MASVS-CRYPTO-1	
CVSS Score	2.1	
Location	runtime	

Risk Assessment

This low-severity finding was identified in runtime. It is considered unlikely to exploit, with a low potential impact; the overall risk to the application is rated low.

Description

The app requested a "MD5" message digest at runtime (`java.security.MessageDigest.getInstance`). MD5 and SHA-1 are broken for security use; use SHA-256 or stronger.

AI Expert Analysis

Exploitability: Low

Impact: MD5 is cryptographically broken for collision resistance and unsuitable for any security-relevant use (password hashing, integrity checks, signatures, token derivation). If used for integrity or authentication, an attacker can craft colliding inputs to bypass checks or forge values. If used for password storage, offline brute-force is trivial. Business impact ranges from authentication bypass to data integrity compromise depending on usage.

Remediation: Replace MD5 with a modern primitive appropriate to the use case. For integrity use SHA-256; for password hashing use Argon2id, scrypt, or bcrypt: // BEFORE (insecure) val digest = MessageDigest.getInstance("MD5").digest(data) // AFTER — integrity val digest = MessageDigest.getInstance("SHA-256").digest(data) // AFTER — password hashing (server-side preferred) val hash = Argon2Kt.hash("argon2id", password, salt) Audit all MessageDigest.getInstance("MD5") and DigestUtils.md5* call sites and migrate; do not rely on MD5 for any trust decision.

Attack Chains: If MD5 is used to sign cached session data, an attacker with storage access can forge a colliding value to bypass integrity checks → Combined with cleartext token storage: MD5-derived token material is recoverable offline

Tool Confidence: MEDIUM reliability — pattern match on crypto API usage; confirm the call site is security-relevant (not e.g. a non-security cache key) before final severity.

MASVS: MASVS-CRYPTO-1 Scan-agent validation: PASS. Exploitability: low. Impact: medium. ---

False Positive Confidence: 60%

False Positive Reason: Matches known FP pattern: 'md5' in

Evidence

runtime

INFO Findings

Finding 86: ✓ Certificate pinning implemented		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-693	
OWASP Top 10	M5	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
Location	com/umermehmood1/maliciousapktest/network/NetworkClient.smali:23	

Risk Assessment

This info-severity finding was identified in com/umermehmood1/maliciousapktest/network/NetworkClient.smali:23. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

SSL/TLS certificate pinning implementation found

AI Expert Analysis

Exploitability: No practical exploitation path identified

Impact: Informational positive control. The analysis engine detected an X509TrustManager reference in NetworkClient.smali, consistent with certificate pinning being implemented. This is a security strength, not a vulnerability; no exposure results from this finding.

Remediation: No remediation required. To strengthen the control, verify the TrustManager performs actual pin comparison (not a no-op trust-all implementation) and pins to a backup key to avoid bricking on certificate rotation. Example: // Ensure pinning validates the leaf/intermediate SPKI hash CertificatePinner pinner = new CertificatePinner.Builder() .add("api.example.com", "sha256/AAAA...=") .add("api.example.com", "sha256/BBBB...=") // backup pin .build();

Tool Confidence: MEDIUM reliability; the mere presence of an X509TrustManager reference does not prove pinning is correctly enforced. Manual review of NetworkClient.smali is recommended to confirm the TrustManager is not a trust-all implementation.

MASVS: MASVS-NETWORK-2

Evidence

```
"Ljavax/net/ssl/X509TrustManager;" ,
```

Finding 87: Malware verdict: CLEAN		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-506	
MASVS Category	MASVS-CODE	
MASVS Control	MASVS-CODE-1	
Location	application	

Risk Assessment

This info-severity finding was identified in application. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

No malware signatures and no high-risk behavioural indicators. 1 informational capability/indicator row(s) were recorded as context.

Recommendation

None required — no malware signatures or high-risk indicators.

Evidence

1 malware indicator(s) assessed

Finding 88: APKiD environment profile		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-693	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
Location	gomobsec-sample-app.apk	

Risk Assessment

This info-severity finding was identified in gomobsec-sample-app.apk. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

Build toolchain and self-defence techniques APKiD observed, recorded as context rather than as findings: how the app was compiled and the anti-analysis checks it performs on its own environment. This is app hardening (MASVS-RESILIENCE), not a set of weaknesses. Detected — anti_vm: Build.MANUFACTURER check (gomobsec-sample-app.apk!classes.dex); compiler: r8 (gomobsec-sample-app.apk!classes.dex), r8 (gomobsec-sample-app.apk!classes3.dex), r8 (gomobsec-sample-app.apk!classes4.dex), r8 (gomobsec-sample-app.apk!classes5.dex), r8 (gomobsec-sample-app.apk!classes6.dex), r8 (gomobsec-sample-app.apk!classes7.dex), r8 (gomobsec-sample-app.apk!classes8.dex), r8 without marker (suspicious) (gomobsec-sample-app.apk!classes10.dex).

AI Expert Analysis

Exploitability: No practical exploitation path identified

Impact: Informational. The analysis engine identified an anti-VM indicator (Build.MANUFACTURER check) and R8 compiler usage in the DEX. This is a resilience observation, not a vulnerability; the presence of anti-VM logic is a defensive control and does not expose data or functionality.

Remediation: No remediation required for the anti-VM indicator itself. If the anti-VM check is intended as a security control, ensure it is backed by server-side attestation (Play Integrity API) rather than client-only checks, since client checks are trivially bypassed by hooking Build.MANUFACTURER. Example: // Prefer server-verified attestation over local Build checks val integrityToken = integrityManager.requestIntegrityToken(...) // validate token server-side before granting sensitive operations

Tool Confidence: HIGH reliability deterministic detection of build/compiler metadata; the finding is descriptive (environment profile) rather than a defect, so severity INFO is appropriate.

MASVS: MASVS-RESILIENCE-1

Recommendation

None required — informational environment profile. The anti-analysis checks are intentional hardening.

Evidence

```
anti_vm: Build.MANUFACTURER check (gomobsec-sample-app.apk!classes.dex); compiler: r8  
(gomobsec-sample-app.apk!classes.dex), r8 (gomobsec-sample-app.apk!classes3.dex), r8  
(gomobsec-sample-app.apk!classes4.dex), r8 (gomobsec-sample-app.apk!classes5.dex), r8  
(gomobsec-sample-app.apk!classes6.dex), r8 (gomobsec-sample-app.apk!classes7.dex), r8  
(gomobsec-sample-app.apk!classes8.dex), r8 without marker (suspicious)  
(gomobsec-sample-app.apk!classes10.dex)
```

Finding 89: SharedPreferences write: access_token		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
Location	runtime	

Risk Assessment

This info-severity finding was identified in runtime. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

The app wrote the SharedPreferences key "access_token" at runtime. SharedPreferences are stored unencrypted in the app sandbox; confirm no secret or personal value is persisted here in cleartext.

AI Expert Analysis

Exploitability: Low

Impact: The access_token is written to SharedPreferences in plaintext. On a rooted device, via a backup extraction, or through a co-located malicious app exploiting a backup/debuggable misconfiguration, the token can be read and replayed to impersonate the user. This directly undermines session confidentiality and enables account takeover.

Remediation: Store tokens in EncryptedSharedPreferences backed by the Android Keystore, and never persist long-lived tokens in plaintext: `val masterKey = MasterKey.Builder(context).setKeyScheme(MasterKey.KeyScheme.AES256_GCM).build() val prefs = EncryptedSharedPreferences.create(context, "secure_prefs", masterKey, EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV, EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM) prefs.edit().putString("access_token", token).apply()` Prefer short-lived tokens with refresh, and set `android:allowBackup="false"` and `android:fullBackupContent` to exclude the prefs file.

Attack Chains: Combined with cleartext logcat token: two independent exfiltration paths for the same credential → Combined with SYSTEM_ALERT_WINDOW overlay: harvested token replayed to hijack the session

Tool Confidence: MEDIUM reliability — runtime SharedPreferences write detection; key name strongly indicates a real credential, but confirm the value is a live token and not a placeholder.

MASVS: MASVS-STORAGE-1 ---

False Positive Confidence: 50%

False Positive Reason: Matches known FP pattern: 'sharedpreferences' in

Evidence

runtime

Finding 90: SharedPreferences write: aws_access_key_id		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
Location	runtime	

Risk Assessment

This info-severity finding was identified in runtime. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

The app wrote the SharedPreferences key "aws_access_key_id" at runtime. SharedPreferences are stored unencrypted in the app sandbox; confirm no secret or personal value is persisted here in cleartext.

AI Expert Analysis

Exploitability: Low

Impact: An AWS access key ID is persisted in plaintext SharedPreferences. If the corresponding secret key is also present or derivable, an attacker with device access can use the credentials to access AWS resources, potentially leading to cloud data breach, resource abuse, and cost exposure. Even the key ID alone aids reconnaissance and credential-stuffing against the AWS account.

Remediation: Never store AWS credentials on-device. Use short-lived STS credentials obtained via a backend broker, or Cognito Identity Pools with scoped IAM roles. If any credential must be cached, store it in EncryptedSharedPreferences and rotate the exposed key immediately: // Server-side: issue scoped, short-lived credentials // Client: never embed long-lived AWS keys val prefs = EncryptedSharedPreferences.create(...) prefs.edit().putString("aws_session", stsSessionJson).apply() Rotate the exposed access key in IAM, audit CloudTrail for unauthorized use, and remove the key from the app and any git history.

Attack Chains: Combined with MANAGE_EXTERNAL_STORAGE: broader device access increases likelihood of credential extraction → Combined with cleartext api_endpoint: attacker redirects AWS calls to a harvesting endpoint

Tool Confidence: MEDIUM reliability — runtime SharedPreferences write detection; key name is a strong indicator of a real AWS credential, corroborated by the runtime-verified finding at index 8.

MASVS: MASVS-STORAGE-1 ---

False Positive Confidence: 50%

False Positive Reason: Matches known FP pattern: 'sharedpreferences' in

Evidence

runtime

Finding 91: SharedPreferences write: api_endpoint		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
Location	runtime	

Risk Assessment

This info-severity finding was identified in runtime. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

The app wrote the SharedPreferences key "api_endpoint" at runtime. SharedPreferences are stored unencrypted in the app sandbox; confirm no secret or personal value is persisted here in cleartext.

AI Expert Analysis

Exploitability: Low

Impact: The API endpoint is stored in plaintext SharedPreferences. While not a secret per se, an attacker with device access can learn the backend topology and, combined with other findings, redirect or replay traffic against internal endpoints. It also facilitates reconnaissance for targeted attacks against the backend.

Remediation: Do not persist endpoint configuration in SharedPreferences. Hardcode the base URL in BuildConfig per build flavor and enforce HTTPS with certificate pinning: `// build.gradle.kts buildConfigField("String", "[REDACTED]", "\"https://api.example.com\"") // Network security config <network-security-config> <domain-config cleartextTrafficPermitted="false"> <domain includeSubdomains="true">api.example.com</domain> <pin-set> <pin digest="SHA-256">BASE64_PIN</pin> </pin-set> </domain-config> </network-security-config>`

Attack Chains: Combined with cleartext AWS key: attacker learns endpoint and pivots to cloud APIs → Combined with REQUEST_INSTALL_PACKAGES: redirect update check to attacker-controlled endpoint

Tool Confidence: MEDIUM reliability — runtime SharedPreferences write detection; endpoint strings are low-sensitivity but confirm insecure storage pattern.

MASVS: MASVS-STORAGE-1 ---

False Positive Confidence: 65%

False Positive Reason: api_endpoint is configuration, not a credential; low-severity SharedPreferences write is largely informational — probable FP but context-dependent.

Evidence

runtime

Finding 92: External app files access		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-312	
OWASP Top 10	M9	
MASVS Category	MASVS-STORAGE	
MASVS Control	MASVS-STORAGE-1	
Location	4 affected locations (e.g. com/umermehmood1/maliciousapktest/storage/LocalStorageProbe.smali)	

Risk Assessment

This info-severity finding was identified in 4 affected locations (e.g. com/umermehmood1/maliciousapktest/storage/LocalStorageProbe.smali). It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

External app files access in FileProvider.smali

AI Expert Analysis

Exploitability: Low

Impact: The app accesses external app files in 4 locations (including LocalStorageProbe). Files written to external storage (getExternalFilesDir or, worse, shared external storage) are readable by other apps on legacy Android versions and by any process with storage permission on shared volumes. If sensitive data is written there, it is exposed to co-resident apps and to physical extraction of the SD card / emulated storage.

Remediation: Store sensitive data only in internal storage (context.getFilesDir() / getCacheDir()), which is private to the app. If external storage is genuinely required, use getExternalFilesDir() (app-scoped, removed on uninstall) and never the shared root. Example: // BEFORE File f = new File(Environment.getExternalStorageDirectory(), "audit.db");
// AFTER File f = new File(context.getFilesDir(), "audit.db"); If files must be shared with other apps, expose them via FileProvider with a minimal, explicit <path-permission> grant rather than world-readable paths.

Attack Chains: Co-resident malicious app with storage permission → read exported files → recover audit/PII data → Physical device access → mount external storage → extract files → Chains with SQL findings (index 6, 7) if the audit database is placed on external storage

Tool Confidence: MEDIUM reliability — API-usage pattern match. Confirm which of the 4 call sites write sensitive data and whether they target app-scoped or shared external storage.

MASVS: MASVS-STORAGE-1 ---

False Positive Confidence: 75%

False Positive Reason: External files access in a storage probe class — common benign file API usage, LOW severity informational

Finding 93: APK signing identity		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-295	
OWASP Top 10	M8	
MASVS Category	MASVS-RESILIENCE	
MASVS Control	MASVS-RESILIENCE-1	
Location	gomobsec-sample-app.apk	

Risk Assessment

This info-severity finding was identified in gomobsec-sample-app.apk. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

Signer DN: C=US, O=Android, CN=Android Debug; key: RSA 2048-bit; SHA-256: fd401007435ad812a6515d7320af09fb5249d8cfe795764eb6a6cc5c9098e076; signers: 1; SourceStamp: False; schemes: v1=off, v2=on, v3=off, v3.1=off, v4=off.

AI Expert Analysis

Exploitability: Low

Impact: The APK is signed with only the v2 scheme (v1, v3, v3.1, v4 off). v2 signing is sufficient for Android 7.0+, so integrity is maintained on modern devices, but the absence of v3/v3.1 key-rotation support and v4 (needed for incremental install) reduces resilience and complicates future key rotation. This is a hygiene/robustness observation, not an active vulnerability.

Remediation: Enable v2, v3 and v4 signing in the build configuration: `android { signingConfigs { release { enableV1Signing true enableV2Signing true enableV3Signing true enableV4Signing true } } }` Store the signing key in a hardware-backed keystore or managed signing service, and plan key rotation via v3 lineage.

Tool Confidence: HIGH reliability — signing scheme flags read directly from the APK. INFO severity is appropriate; no exploitable weakness on modern Android.

MASVS: MASVS-RESILIENCE-1

Recommendation

No action required. Record the release certificate fingerprint and alert on rotation; a changed fingerprint for an existing package name is an update-hijack signal.

Evidence

schemes: v1=off, v2=on, v3=off, v3.1=off, v4=off

Finding 94: No UI hardening against tapjacking (filterTouchesWhenObscured)		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
CWE	CWE-1021	
OWASP Top 10	M4	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-3	
Location	application-wide	

Risk Assessment

This info-severity finding was identified in application-wide. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

No view in the app sets `android:filterTouchesWhenObscured` (or the programmatic equivalent). An overlay drawn by another app can trick the user into interacting with security-relevant controls underneath it (tapjacking). Recorded for coverage; harden the controls that perform sensitive actions.

AI Expert Analysis

Exploitability: Low

Impact: Without `filterTouchesWhenObscured`, the app's views will accept touch events even when partially obscured by an overlay from another app. A malicious app holding `SYSTEM_ALERT_WINDOW` can overlay the UI and trick the user into tapping sensitive controls (tapjacking). Impact is limited to devices where an overlay-capable app is installed and the user interacts with the obscured screen.

Remediation: Enable touch filtering on sensitive views or globally via theme: // Programmatic
`webView.setFilterTouchesWhenObscured(true); button.setFilterTouchesWhenObscured(true);` <!-- Or via theme applied to sensitive activities --> `<item name="android:filterTouchesWhenObscured">true</item>` Also verify the app does not itself request `SYSTEM_ALERT_WINDOW` (finding 3).

Attack Chains: Chains with finding 3 (overlay permission) and finding 4 (no `FLAG_SECURE`) to enable convincing tapjacking/overlay phishing

Tool Confidence: HIGH reliability — absence confirmed in first-party source. INFO severity is appropriate; this is defence-in-depth.

MASVS: MASVS-PLATFORM-2

Recommendation

Set `android:filterTouchesWhenObscured="true"` on buttons and views that trigger sensitive actions (confirm payment, grant permission), so touches are dropped while an overlay covers them.

Evidence

No `filterTouchesWhenObscured` / `setFilterTouchesWhenObscured` reference in the first-party source.

Finding 95: Code: Android Ssl Pinning		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
OWASP Top 10	M5	
MASVS Category	MASVS-NETWORK	
MASVS Control	MASVS-NETWORK-1	
Location	com/umermehmood1/maliciousapktest/network/NetworkClient.java	

Risk Assessment

This info-severity finding was identified in com/umermehmood1/maliciousapktest/network/NetworkClient.java. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

This App uses SSL certificate pinning to detect or prevent MITM attacks in secure communication channel. Affected files (1): com/umermehmood1/maliciousapktest/network/NetworkClient.java

AI Expert Analysis

Exploitability: Low

Impact: Informational. The analysis engine flagged the network client class in the context of SSL pinning; absence of pinning is not itself a vulnerability. If the app transmits sensitive data over TLS without certificate pinning, a user with a proxy CA installed (or on a compromised network with a trusted root) could intercept traffic. Business impact is limited to scenarios where an attacker already controls the device trust store.

Remediation: If sensitive data is transmitted, add certificate pinning via Network Security Config. Example (res/xml/network_security_config.xml): <network-security-config> <domain-config cleartextTrafficPermitted="false"> <domain includeSubdomains="true">api.example.com</domain> <pin-set expiration="2026-12-31"> <pin digest="SHA-256">BASE64_PIN_PRIMARY</pin> <pin digest="SHA-256">BASE64_PIN_BACKUP</pin> </pin-set> </domain-config> </network-security-config> Reference it from the manifest via android:networkSecurityConfig. Always include a backup pin to avoid bricking on rotation.

Attack Chains: User-installed proxy CA + no pinning → MITM of API traffic → credential/token capture

Tool Confidence: HIGH reliability detection but INFO severity — this is a configuration observation, not a confirmed vulnerability. Confirm whether pinning is genuinely absent and whether sensitive endpoints exist.

MASVS: MASVS-NETWORK-1

Recommendation

No action required — certificate pinning was detected, which is a control.

Two things worth verifying: that a backup pin is configured, so an unplanned certificate rotation does not lock every user out, and that the pin is on the public key or SPKI rather than the leaf certificate, which expires.

Evidence

com/umermehmood1/maliciousapktest/network/NetworkClient.java

Finding 96: Exported broadcast receiver		
Likelihood	Impact	Risk
N/A	Informational	Informational
Severity	INFO	
Status	OPEN	
OWASP Top 10	M3	
MASVS Category	MASVS-PLATFORM	
MASVS Control	MASVS-PLATFORM-1	
Location	AndroidManifest.xml:receiver:androidx.profileinstaller.ProfileInstallReceiver	

Risk Assessment

This info-severity finding was identified in AndroidManifest.xml:receiver:androidx.profileinstaller.ProfileInstallReceiver. It is considered n/a to exploit, with a informational potential impact; the overall risk to the application is rated informational.

Description

[Confirmed by 2 analyzers: androguard, mobsf] A Broadcast Receiver is found to be shared with other apps on the device therefore leaving it accessible to any other application on the device. It is protected by a permission which is not defined in the analysed application. As a result, the protection level of the permission should be checked where it is defined. If it is set to normal or dangerous, a malicious application can request and obtain the permission and interact with the component. If it is set to signature, only applications signed with the same certificate can obtain the permission. This component belongs to a bundled third-party library rather than to this application. The exposure is real and is part of the shipped attack surface, but it is not code this project owns or can change.

AI Expert Analysis

Exploitability: No practical exploitation path identified

Impact: Informational. androidx.profileinstaller.ProfileInstallReceiver is an AndroidX library component that ships exported by design to allow the system and profile installer to trigger baseline-profile compilation. It performs no app-specific privileged action and holds no sensitive data.

Remediation: No action required. This is a standard AndroidX component. If the app does not use baseline profiles, the dependency can be removed to reduce surface area, but its presence is not a vulnerability.

Tool Confidence: HIGH reliability detection, but the component is third-party AndroidX code (androidx.profileinstaller) despite is_library being false — the receiver is declared in the merged manifest, not authored by the app.

Evidence

```
AndroidManifest.xml:receiver:androidx.profileinstaller.ProfileInstallReceiver
```

Scope & Methodology

This assessment was performed with GoMobSec's automated mobile application security testing pipeline against the released application package identified in the Application Overview, without access to its source code — the position of an attacker who has obtained the app from a store or device.

Assessment type: Automated static and dynamic analysis

Assessment Phases

- **Ingestion** — file validation, SHA-256/MD5 hashing, metadata extraction and platform detection.
- **Decompilation** — application bytecode and resources unpacked for inspection.
- **Static Analysis** — manifest and configuration review, hardcoded-secret detection, pattern-based source analysis, cryptography review and third-party dependency CVE lookup.
- **Dynamic Analysis** — runtime checks on an instrumented Android test device: device-level tests, log and file-system inspection, and traffic observation.
- **Network Analysis** — endpoint extraction, TLS configuration, and static API, JWT and GraphQL review.
- **Malware Analysis** — signature, rule-based and behavioural detection with packer/obfuscator identification.
- **Privacy Analysis** — tracker SDK and personal-data exposure identification.
- **Resilience Analysis** — root/jailbreak detection, obfuscation and anti-tamper controls.
- **Threat Intelligence** — enrichment of indicators against public vulnerability and reputation sources.
- **Triage** — AI-assisted per-finding analysis, false-positive suppression and severity calibration; confirmed false positives are excluded from this report.

Out of Scope

- Manual penetration testing and source-code review.
- Server-side systems and APIs, beyond what the application package itself reveals.
- Other versions or builds of the application than the one identified by its hash.

Standards and Scoring

Findings are mapped to the OWASP Mobile Application Security Verification Standard (MASVS v2) and the OWASP Mobile Top 10, and cite Common Weakness Enumeration (CWE) identifiers. Where a finding carries a CVSS v4.0 vector it is printed with the finding; severity otherwise follows the Critical / High / Medium / Low / Informational scale used throughout.

AI-assisted analysis was produced by the deepseek-chat model (deepseek).

The findings and their evidence are produced by deterministic tooling and reproduce on re-scan of the same build. The AI-written narrative, severity rationale and false-positive assessment are model-generated and may vary between runs; they support the evidence rather than stand in for it.

Limitations

- Results reflect the application at the time of testing; a later build may differ.
- Automated testing complements, and does not replace, a manual assessment.
- Runtime coverage depends on the application installing, launching and being exercised on the test device; functionality behind authentication or specific device state may not have been reached.

Remediation Roadmap

Priority	Finding	Location
Immediate (Before Next Release)	WebView universal file access — critical XSS risk	com/umermehmood1/maliciousapctest/WebViewActivity.java:58
Sprint 1 (2 Weeks)	Hardcoded Bearer token	3 affected locations
Sprint 1 (2 Weeks)	Symmetric key derived from a String via getBytes() — a hardcoded or password string is not key material; use a KDF (PBKDF2/HKDF) or KeyGenerator	com/umermehmood1/maliciousapctest/security/CryptoSamples.java:58
Sprint 1 (2 Weeks)	SQL injection via string concatenation in.rawQuery	com/umermehmood1/maliciousapctest/db/AuditDatabase.java:33
Sprint 1 (2 Weeks)	Attacker-controlled input loaded into a WebView	WebViewActivity.java:69
Sprint 1 (2 Weeks)	Cleartext HTTP endpoint: uploads.vulnerable-test.example	http://uploads.vulnerable-test.example/plain-text-upload
Sprint 1 (2 Weeks)	Cleartext traffic allowed	AndroidManifest.xml:application
Sprint 1 (2 Weeks)	WebView JavaScript enabled — verify XSS protections	com/umermehmood1/maliciousapctest/WebViewActivity.java:52
Sprint 1 (2 Weeks)	App is debuggable	4 affected locations
Sprint 1 (2 Weeks)	Token leaked in logs	logcat:line:16
Sprint 1 (2 Weeks)	Sensitive Data Leaked to Logcat: Auth token in logcat	
Sprint 1 (2 Weeks)	Code: Cbc Padding Oracle	com/umermehmood1/maliciousapctest/security/CryptoSamples.java
Sprint 1 (2 Weeks)	Excessive dangerous permissions (19)	AndroidManifest.xml:uses-permission
Sprint 2 (4 Weeks)	Sensitive data possibly logged	com/umermehmood1/maliciousapctest/logging/LogSamples.java:40
Sprint 2 (4 Weeks)	Code: Android Webview Allow File From Url	com/umermehmood1/maliciousapctest/WebViewActivity.java
Sprint 2 (4 Weeks)	MASTG: The WebView allows to load external content. It is suggested to verify that the granted accesses are necessary and follow the best practices.	4 affected locations
Sprint 2 (4 Weeks)	Sensitive data possibly logged	com/umermehmood1/maliciousapctest/network/LoggingInterceptorSample.java:23
Sprint 2 (4 Weeks)	MASTG: The application writes sensitive data in application logs.	3 affected locations

Sprint 2 (4 Weeks)	Deep link scheme can be claimed by another app	AndroidManifest.xml
Sprint 2 (4 Weeks)	Predictable PRNG: java.util.Random seeded with a constant yields a fully reproducible sequence — use SecureRandom and never set a fixed seed	com/umermehmood1/maliciousapktest/security/CryptoSamples.java:75
Sprint 2 (4 Weeks)	WebView allows mixed content (MIXED_CONTENT_ALWAYS_ALLOW) — http:// resources load inside an https:// page and can be tampered with over the network	com/umermehmood1/maliciousapktest/WebViewActivity.java:59
Sprint 2 (4 Weeks)	WebView file access enabled — potential path traversal	com/umermehmood1/maliciousapktest/WebViewActivity.java:55
Sprint 2 (4 Weeks)	Exported activity	WebViewActivity (AndroidManifest.xml)
Sprint 2 (4 Weeks)	Exported broadcast receiver	BootAuditReceiver (AndroidManifest.xml)
Sprint 2 (4 Weeks)	Exported activity	SecretsActivity (AndroidManifest.xml)

Appendix: CVSS Scores

#	Title	CVSS	Severity
1	WebView universal file access — critical XSS risk	9.3	CRITICAL
2	Hardcoded Bearer token	8.7	HIGH
3	Symmetric key derived from a String via getBytes() — a hardcoded or password string is not key material; use a KDF (PBKDF2/HKDF) or KeyGenerator	8.7	HIGH
4	SQL injection via string concatenation in.rawQuery	8.6	HIGH
5	Attacker-controlled input loaded into a WebView	8.6	HIGH
6	Cleartext HTTP endpoint: uploads.vulnerable-test.example	8.2	HIGH
7	Cleartext traffic allowed	8.2	HIGH
8	WebView JavaScript enabled — verify XSS protections	7.1	HIGH
9	App is debuggable	7.0	HIGH
10	Token leaked in logs	7.0	HIGH
11	Sensitive Data Leaked to Logcat: Auth token in logcat	7.0	HIGH
12	Code: Cbc Padding Oracle	7.0	HIGH
13	Excessive dangerous permissions (19)	7.0	HIGH
14	App backup enabled	7.0	HIGH
15	Sensitive data stored in cleartext in the app sandbox (runtime-verified)	7.0	HIGH
16	Sensitive data stored in cleartext in the app sandbox (runtime-verified)	7.0	HIGH
17	HostnameVerifier accepts all hosts (verify() returns true) — hostname verification disabled, enabling MITM	7.0	HIGH
18	Token leaked in logs	7.0	HIGH
19	Sensitive data possibly logged	6.9	MEDIUM
20	Code: Android Webview Allow File From Url	6.9	MEDIUM
21	MASTG: The WebView allows to load external content. It is suggested to verify that the granted accesses are necessary and follow the best practices.	6.9	MEDIUM
22	Sensitive data possibly logged	6.9	MEDIUM
23	MASTG: The application writes sensitive data in application logs.	6.8	MEDIUM
24	Deep link scheme can be claimed by another app	6.8	MEDIUM
25	Predictable PRNG: java.util.Random seeded with a constant yields a fully reproducible sequence — use SecureRandom and never set a fixed seed	6.3	MEDIUM
26	WebView allows mixed content (MIXED_CONTENT_ALWAYS_ALLOW) — http:// resources load inside an https:// page and can be tampered with over the network	6.3	MEDIUM

27	WebView file access enabled — potential path traversal	6.0	MEDIUM
28	Exported activity	5.1	MEDIUM
29	Exported broadcast receiver	5.1	MEDIUM
30	Exported activity	5.1	MEDIUM
31	Exported component	5.1	MEDIUM
32	Permission: WRITE_CALL_LOG	5.0	MEDIUM
33	Hardcoded iv detected	5.0	MEDIUM
34	APK signed with a debug certificate	5.0	MEDIUM
35	WebView file-URL access enabled — JavaScript running in a file:// page can read other local files (setAllowFileAccessFromFileURLs(true)), a local file-theft vector. It defaults to false on API 16+; re	5.0	MEDIUM
36	IV derived from a String via getBytes() — a string-derived IV is fixed/predictable; use SecureRandom.nextBytes() per message	5.0	MEDIUM
37	MASTG: The method "echo" is accessible from a WebView.	5.0	MEDIUM
38	MASTG: The method "getFakeDeviceId" is accessible from a WebView.	5.0	MEDIUM
39	MASTG: The method "getHardcodedToken" is accessible from a WebView.	5.0	MEDIUM
40	MASTG: WebView can load remote content from an endpoint or local file system in an unsafe manner.	5.0	MEDIUM
41	MASTG: The application allows for JavaScript execution within a WebView instance.	5.0	MEDIUM
42	MASTG: The application exposes the Java class "WebBridge" through the WebView "webView".	5.0	MEDIUM
43	MASTG: Weak transport protocol version TLS is used.	5.0	MEDIUM
44	JavaScript bridge exposed in the app's own code	5.0	MEDIUM
45	App can be installed on a vulnerable unpatched Android version 7.0, [minSdk=24]	5.0	MEDIUM
46	Permission: CAMERA	5.0	MEDIUM
47	Permission: RECORD_AUDIO	5.0	MEDIUM
48	Permission: ACCESS_FINE_LOCATION	5.0	MEDIUM
49	Permission: ACCESS_COARSE_LOCATION	5.0	MEDIUM
50	Permission: ACCESS_BACKGROUND_LOCATION	5.0	MEDIUM
51	Permission: READ_CONTACTS	5.0	MEDIUM
52	Permission: WRITE_CONTACTS	5.0	MEDIUM
53	Permission: READ_CALENDAR	5.0	MEDIUM
54	Permission: WRITE_CALENDAR	5.0	MEDIUM

55	Permission: READ_SMS	5.0	MEDIUM
56	Permission: SEND_SMS	5.0	MEDIUM
57	Dangerous permission: RECEIVE_SMS	5.0	MEDIUM
58	Permission: READ_CALL_LOG	5.0	MEDIUM
59	Permission: CALL_PHONE	5.0	MEDIUM
60	Permission: READ_PHONE_STATE	5.0	MEDIUM
61	Permission: BODY_SENSORS	5.0	MEDIUM
62	Dangerous permission: POST_NOTIFICATIONS	5.0	MEDIUM
63	Permission: READ_EXTERNAL_STORAGE	5.0	MEDIUM
64	Permission: WRITE_EXTERNAL_STORAGE	5.0	MEDIUM
65	Dangerous permission: READ_MEDIA_IMAGES	5.0	MEDIUM
66	Dangerous permission: READ_MEDIA_VIDEO	5.0	MEDIUM
67	Dangerous permission: READ_MEDIA_AUDIO	5.0	MEDIUM
68	Permission: MANAGE_EXTERNAL_STORAGE	5.0	MEDIUM
69	Dangerous permission: REQUEST_INSTALL_PACKAGES	5.0	MEDIUM
70	Dangerous permission: SYSTEM_ALERT_WINDOW	5.0	MEDIUM
71	✗ Missing: Anti-debugging	3.0	LOW
72	Dependency CVE: Gson — CVE-2022-25647	3.0	LOW
73	No screen sets FLAG_SECURE	3.0	LOW
74	Draw-over-other-apps (overlay) permission requested	3.0	LOW
75	Direct SQL execution (check for injection)	3.0	LOW
76	MASTG: The application uses deprecated cryptographic algorithms.	3.0	LOW
77	Raw SQL query (smali)	3.0	LOW
78	Exported components detected	3.0	LOW
79	Exported component	3.0	LOW
80	✗ Missing: Anti-emulator	3.0	LOW
81	✗ Missing: Root detection	3.0	LOW
82	✗ Missing: Integrity verification	3.0	LOW
83	✗ No code obfuscation detected	3.0	LOW
84	Code: Android Webview	3.0	LOW
85	Crypto: MD5 hash used	2.1	LOW

86	✓ Certificate pinning implemented	0	INFO
87	Malware verdict: CLEAN	0	INFO
88	APKiD environment profile	0	INFO
89	SharedPreferences write: access_token	0	INFO
90	SharedPreferences write: aws_access_key_id	0	INFO
91	SharedPreferences write: api_endpoint	0	INFO
92	External app files access	0	INFO
93	APK signing identity	0	INFO
94	No UI hardening against tapjacking (filterTouchesWhenObscured)	0	INFO
95	Code: Android Ssl Pinning	0	INFO
96	Exported broadcast receiver	0	INFO

Severity Ratings

Severity	CVSS Range	Description
CRITICAL	9.0 – 10.0	Exploitable with minimal user interaction; direct compromise of the application, its data, or its platform.
HIGH	7.0 – 8.9	Significant impact; requires limited conditions or user interaction to exploit.
MEDIUM	4.0 – 6.9	Moderate impact; exploitation depends on specific conditions or misconfigurations.
LOW	0.1 – 3.9	Limited impact; contributes to overall risk but is not directly exploitable to high severity.
INFO	0.0	Informational; no direct risk but worth noting for completeness.