ANDROID APK - OWASP MOBILE TOP 10 - MASVS

Mobile App Security Report

com.example.mexribon_kurash_al v1.1.0 (2)

10
VULNERABILITIES IN YOUR CODE
+3 in dependencies · 13 total

37
OBSERVATIONS

SEVERITY BREAKDOWN

0	0	3	10	0
CRITICAL	HIGH	MEDIUM	LOW	INFO

SHA-256
9aec4ef6ef7b58500c06fc5336437fab8e121637e81de7d8e02fa1c4313733de

PACKAGE SIZE	ASSESSED	STATUS
50.6 MB	2026-11-02T09:25:50+00:00	succeeded



01 Executive summary

This report covers static and bytecode analysis of com.example.mexribon_kurash_al. 13 vulnerability findings were identified: 3 medium, 10 low. The most affected area is code (5 finding(s)). The highest-severity finding is "File.setExecutable(true, false) makes a file world-executable in io.flutter.embedding.engine.FlutterJNI". Alongside these, 37 informational observation(s) describe the app's configuration, inventory and hardening posture. Findings are listed below, ordered by severity and blast radius (how many findings share a root cause).

10	37
VULNERABILITIES IN YOUR CODE · +3 in dependencies (13 total)	OBSERVATIONS

0	0	3	10	0
CRITICAL	HIGH	MEDIUM	LOW	INFO

10 vulnerabilities in your code · 3 in dependencies · 13 total | 0 marked false-positive | 6 findings | 25 inventory | 6 posture | 50 total findings

Contents

01 Executive summary 2

02 Findings overview & charts 4

03 Attack surface 6

04 Fix these first 7

05 Findings register 8

06 Detailed findings 10

07 App inventory 16

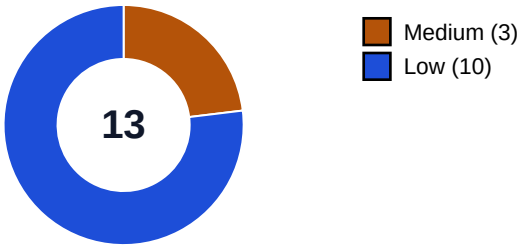
08 Security posture 17

09 Methodology, scope & limitations 19

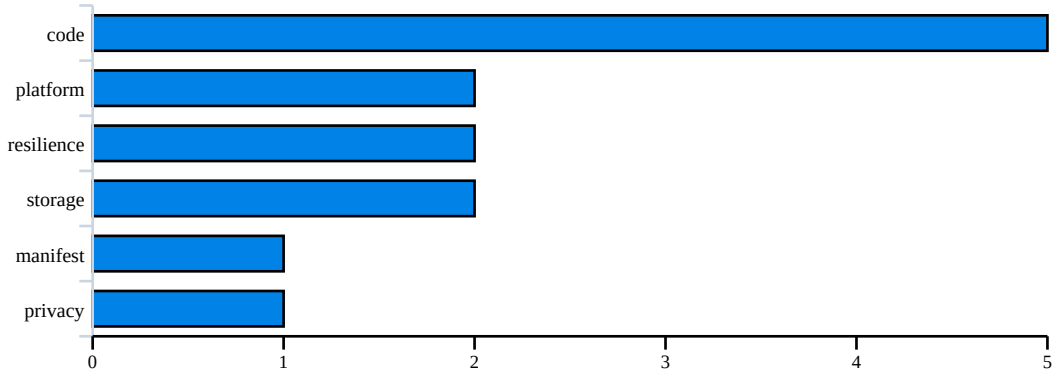
10 Glossary 20

02 Findings overview & charts

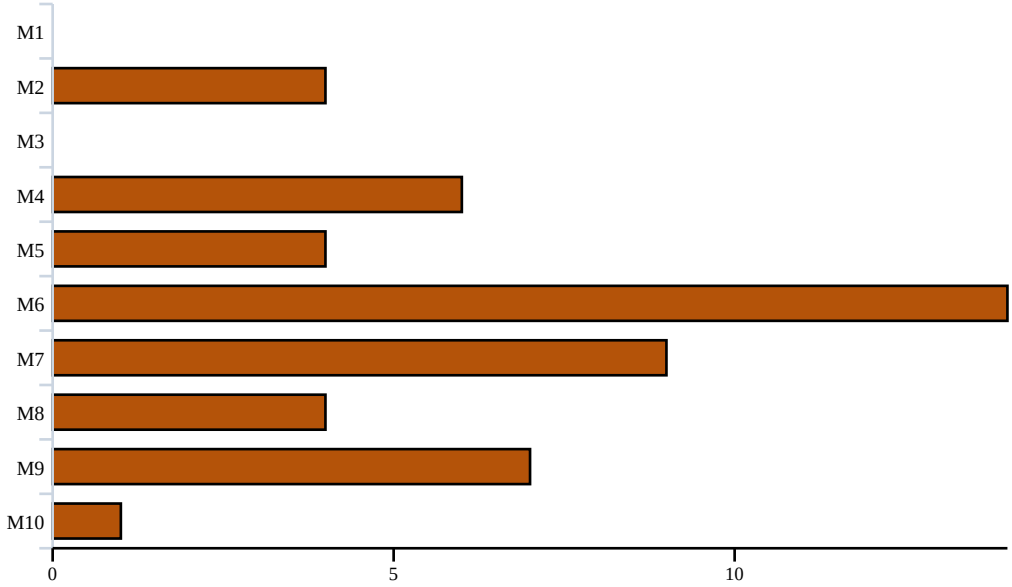
Severity distribution



Findings by category (active vulnerabilities)



OWASP Mobile Top 10 — findings observed



OWASP Mobile Top 10 coverage

OWASP category	Findings observed
M1 — Improper Credential Usage	Tested — no findings
M2 — Inadequate Supply Chain Security	4 findings observed
M3 — Insecure Authentication/Authorization	Tested — no findings
M4 — Insufficient Input/Output Validation	6 findings observed
M5 — Insecure Communication	4 findings observed
M6 — Inadequate Privacy Controls	14 findings observed
M7 — Insufficient Binary Protections	9 findings observed
M8 — Security Misconfiguration	4 findings observed
M9 — Insecure Data Storage	7 findings observed
M10 — Insufficient Cryptography	1 finding observed

MASVS control-group rollup

MASVS category	Findings observed
MASVS-STORAGE — Storage	6 findings observed
MASVS-CRYPTO — Cryptography	1 finding observed
MASVS-AUTH — Authentication	Tested — no findings
MASVS-NETWORK — Network	4 findings observed
MASVS-PLATFORM — Platform	5 findings observed
MASVS-CODE — Code Quality	15 findings observed
MASVS-RESILIENCE — Resilience	4 findings observed
MASVS-PRIVACY — Privacy	14 findings observed

03 Attack surface

Signal	Value	Why it matters
Target SDK	36	At or above the Google Play target-API floor (34).
Minimum SDK	24	Minimum supported Android API level.
Network Security Config	Absent	No network-security-config; cleartext and trust use platform defaults.
APK signing	v2	A modern APK signature scheme (v2/v3) is present.

04 Fix these first

Highest-impact distinct issues, ordered by severity then blast radius (how many findings share the root cause).

#	Issue	Sev	Category	OWASP	First step	Clears
1	File.setExecutable(true, false) makes a file world-executable in io.flutter.embedding.engine.FlutterJNI	MEDIUM	storage	M9	Call the one-arg `File.setExecutable(true)` (which defaults `ownerOnly=true`), or pass `ownerOnly=true` explicitly, so...	1
2	missing stack canaries in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	MEDIUM	code	M7	Add `-fstack-protector-strong` (preferred) or `-fstack-protector-all` to the C/C++ compiler flags.	1
3	File.setReadable(true, false) makes a file world-readable in io.flutter.embedding.engine.FlutterJNI	MEDIUM	storage	M9	Call the one-arg `File.setReadable(true)` (which defaults `ownerOnly=true`), or pass `ownerOnly=true` explicitly, so th...	1
4	No root-detection signal found	LOW	resilience	M7	Add Play Integrity API for verified device attestation.	1
5	No shouldShowRequestPermissionRationale calls (1 runtime permissions declared)	LOW	privacy	M6	Before each `requestPermissions(...)` call, check `shouldShowRequestPermissionRationale(permission)`.	1
6	FileProvider shares an over-broad `cache-path`: io.flutter.plugins.imagepicker.ImagePickerFileProvider	LOW	platform	M8	Replace the over-broad `<paths>` entry with the narrowest element + sub-path the sharing feature actually needs - e.g.	1
7	missing RELRO in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	LOW	code	M7	Add `-Wl,-z,relro -Wl,-z,now` to the linker flags.	1
8	missing _FORTIFY_SOURCE in libdartjni.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	LOW	code	M7	Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op...	1
9	missing _FORTIFY_SOURCE in libdatastore_shared_counter.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	LOW	code	M7	Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op...	1
10	missing _FORTIFY_SOURCE in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	LOW	code	M7	Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op...	1

05 Findings register

ID	Sev	Finding	Category	OWASP	MASVS	Location
AAS-29A023	MEDIUM	File.setExecutable(true, false) makes a file world-executable in io.flutter.embedding.engine.FlutterJNI	storage	M9	MASVS-STORAGE	Lio/flutter/embedding/engine/FlutterJNI;->loadLibrary(Landroid/content/Context;)V @345
AAS-B0A133	MEDIUM	File.setReadable(true, false) makes a file world-readable in io.flutter.embedding.engine.FlutterJNI	storage	M9	MASVS-STORAGE	Lio/flutter/embedding/engine/FlutterJNI;->loadLibrary(Landroid/content/Context;)V @342
AAS-B40849	MEDIUM	missing stack canaries in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	code	M7	MASVS-CODE	-
AAS-939108	LOW	FileProvider shares an over-broad `cache-path`: io.flutter.plugins.imagepicker.ImagePickerFileProvider	platform	M8	MASVS-PLATFORM	-
AAS-65689A	LOW	No anti-debug / signature-check signal found	resilience	M7	MASVS-RESILIENCE	-
AAS-618C23	LOW	No network security config declared	manifest	M5	MASVS-NETWORK	-
AAS-6F8E5A	LOW	No root-detection signal found	resilience	M7	MASVS-RESILIENCE	-
AAS-5D3CB2	LOW	No shouldShowRequestPermissionRationale calls (1 runtime permissions declared)	privacy	M6	MASVS-PRIVACY	-
AAS-98154A	LOW	Untrusted Intent/Uri input reaches a filesystem path (new File()) in t0	platform	M4	MASVS-PLATFORM	Lt0;->l(Landroid/content/Context; Landroid/net/Uri;)Ljava/lang/String; @70
AAS-C1B189	LOW	missing RELRO in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	code	M7	MASVS-CODE	-
AAS-354E11	LOW	missing _FORTIFY_SOURCE in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	code	M7	MASVS-CODE	-
AAS-AE6351	LOW	missing _FORTIFY_SOURCE in libdartjni.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	code	M7	MASVS-CODE	-
AAS-1ACF53	LOW	missing _FORTIFY_SOURCE in libdatastore_shared_counter.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)	code	M7	MASVS-CODE	-
AAS-89057D	HIGH	ObjectInputStream.readObject called in fj	code	M4	MASVS-CODE	Lfj;->b(Ljava/lang/String; Ljava/util/List;)Lxk; @124
AAS-170193	HIGH	ObjectInputStream.readObject called in zu	code	M4	MASVS-CODE	Lzu;->c(Ljava/lang/Object; Ldk;)Ljava/lang/Object; @48
AAS-14CFB6	MEDIUM	ContentProvider.openFile opens a URI-derived path without canonicalization in ta	platform	M4	MASVS-PLATFORM	Lta;->openFile(Landroid/net/Uri; Ljava/lang/String;)Landroid/os/ParcelFileDescriptor; @0
AAS-D60460	MEDIUM	Intent.getSerializableExtra called in tb	code	M4	MASVS-CODE	Ltb;->onStart()V @168

ID	Sev	Finding	Category	OWASP	MASVS	Location
AAS-5F9065	MEDIUM	Intent.getSerializableExtra called in xb	code	M4	MASVS-CODE	Lxb;->a(Lgc;)V @76
AAS-9F454F	LOW	StrictMode policy configured in release build	resilience	M8	MASVS-CODE	-

06 Detailed findings

Medium severity (3)

MEDIUM

missing stack canaries in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)

M7 · MASVS-CODE · MASTG-TEST-0044 · MASTG-TEST-0223

AAS-B40849 code.native_lib_hardening.no_stack_canaries | code.native_lib_hardening v1.0.1

`libapp.so` has missing stack canaries. No `__stack_chk_fail` reference found in the dynamic symbol table. The compiler did not enable `-fstack-protector*`, so stack-buffer overflows are not detected before they overwrite the return address.

RECOMMENDATION Add `-fstack-protector-strong` (preferred) or `-fstack-protector-all` to the C/C++ compiler flags. The former is the Android NDK default since r17.

MEDIUM

File.setExecutable(true, false) makes a file world-executable in io.flutter.embedding.engine.FlutterJNI

M9 · MASVS-STORAGE

AAS-29A023 storage.world_accessible_modes.file_permission_setter | storage.world_accessible_modes v1.1.0

`void io.flutter.embedding.engine.FlutterJNI.loadLibrary(Context)` calls `File.setExecutable(true, false)`. The second argument (`ownerOnly`) is `false`, so the permission is granted to EVERY app on the device - the file becomes world-executable. Any other app can execute the file; combined with world-writable it enables arbitrary code execution.

```
LOCATION Lio/flutter/embedding/engine/FlutterJNI;->loadLibrary(Landroid/content/Context;)V @345
VALUE True
```

RECOMMENDATION Call the one-arg `File.setExecutable(true)` (which defaults `ownerOnly=true`), or pass `ownerOnly=true` explicitly, so the permission applies only to the owning app. Keep app data in internal storage (`Context.getFilesDir()`, `MODE_PRIVATE`); if you need to share data with another app, use a `ContentProvider` gated by `android:permission` rather than loosening filesystem ACLs.

MEDIUM

File.setReadable(true, false) makes a file world-readable in io.flutter.embedding.engine.FlutterJNI

M9 · MASVS-STORAGE

AAS-B0A133 storage.world_accessible_modes.file_permission_setter | storage.world_accessible_modes v1.1.0

`void io.flutter.embedding.engine.FlutterJNI.loadLibrary(Context)` calls `File.setReadable(true, false)`. The second argument (`ownerOnly`) is `false`, so the permission is granted to EVERY app on the device - the file becomes world-readable. Any other app - or a user with a file manager - can read the file, leaking anything sensitive it holds.

```
LOCATION Lio/flutter/embedding/engine/FlutterJNI;->loadLibrary(Landroid/content/Context;)V @342
VALUE True
```

RECOMMENDATION Call the one-arg `File.setReadable(true)` (which defaults `ownerOnly=true`), or pass `ownerOnly=true` explicitly, so the permission applies only to the owning app. Keep app data in internal storage (`Context.getFilesDir()`, `MODE_PRIVATE`); if you need to share data with another app, use a `ContentProvider` gated by `android:permission` rather than loosening filesystem ACLs.

Low severity (10)

LOW missing _FORTIFY_SOURCE in libdatastore_shared_counter.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)

M7 · MASVS-CODE · MASTG-TEST-0044

AAS-1ACF53 code.native_lib_hardening.no_fortify_source | code.native_lib_hardening v1.0.1

`libdatastore_shared_counter.so` has missing _FORTIFY_SOURCE. No `__*_chk` symbols found in the dynamic symbol table. The compiler did not enable `-D_FORTIFY_SOURCE=1` or higher, which replaces a subset of unsafe libc calls (`memcpy`/`strcpy`/`sprintf`/...) with bounds-checked variants.

RECOMMENDATION Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op at `-O0`).

LOW missing RELRO in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)

M7 · MASVS-CODE · MASTG-TEST-0044

AAS-C1B189 code.native_lib_hardening.no_relro | code.native_lib_hardening v1.0.1

`libapp.so` has missing RELRO. The library has no `PT_GNU_RELRO` segment. Relocations and the Global Offset Table remain writable at runtime - any memory-corruption bug that can write a controlled pointer into the GOT trivially redirects control flow.

RECOMMENDATION Add `-Wl,-z,relro -Wl,-z,now` to the linker flags. Both are Android NDK defaults; a missing RELRO usually indicates manual `LDFLAGS` overrides.

LOW missing _FORTIFY_SOURCE in libapp.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)

M7 · MASVS-CODE · MASTG-TEST-0044

AAS-354E11 code.native_lib_hardening.no_fortify_source | code.native_lib_hardening v1.0.1

`libapp.so` has missing _FORTIFY_SOURCE. No `__*_chk` symbols found in the dynamic symbol table. The compiler did not enable `-D_FORTIFY_SOURCE=1` or higher, which replaces a subset of unsafe libc calls (`memcpy`/`strcpy`/`sprintf`/...) with bounds-checked variants.

RECOMMENDATION Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op at `-O0`).

LOW missing _FORTIFY_SOURCE in libdartjni.so (3 ABIs: arm64-v8a, armeabi-v7a, x86_64)

M7 · MASVS-CODE · MASTG-TEST-0044

AAS-AE6351 code.native_lib_hardening.no_fortify_source | code.native_lib_hardening v1.0.1

`libdartjni.so` has missing _FORTIFY_SOURCE. No `__*_chk` symbols found in the dynamic symbol table. The compiler did not enable `-D_FORTIFY_SOURCE=1` or higher, which replaces a subset of unsafe libc calls (`memcpy`/`strcpy`/`sprintf`/...) with bounds-checked variants.

RECOMMENDATION Add `-D_FORTIFY_SOURCE=2` to the C/C++ compiler flags and ensure optimisation is at `-O1` or higher (FORTIFY is a no-op at `-O0`).

LOW No network security config declared

M5 · MASVS-NETWORK · MASTG-TEST-0235

AAS-618C23 manifest.no_network_security_config | manifest.security_attributes v1.1.0

`android:networkSecurityConfig` is unset. The app falls back to the platform default trust store and has no per-domain policy hooks - no certificate pinning, no per-host cleartext restrictions, no debug-only trust anchors.

RECOMMENDATION Declare an NSC under `res/xml/` and reference it from `<application android:networkSecurityConfig="@xml/...">`. At minimum: `<base-config cleartextTrafficPermitted="false">` with `<trust-anchors><certificates src="system"/></trust-anchors>`. Add `<pin-set>` entries for high-value endpoints (login, payment APIs) once you've audited their certificate rotation schedule.

LOW

FileProvider shares an over-broad `cache-path`: io.flutter.plugins.imagepicker.ImagePickerFileProvider

M8 · MASVS-PLATFORM · MASTG-TEST-0339 · MASTG-TEST-0357

AAS-939108 platform.content_provider.overbroad_fileprovider_path | platform.content_provider_audit v1.3.0

The `FileProvider` `io.flutter.plugins.imagepicker.ImagePickerFileProvider` (authority `com.example.mexribon_kurash_al.flutter.image_provider`) declares `android:grantUriPermissions="true"` and its `<paths>` resource (`res/gX.xml`) contains `<cache-path path="."/>`, which shares the app's entire internal files / cache directory - every file the app stores there (session tokens, exports, copied databases), not just the ones meant for sharing. A grantable `FileProvider` hands any app that receives a `content://` URI (via `FLAG_GRANT_READ_URI_PERMISSION` / `FLAG_GRANT_WRITE_URI_PERMISSION`) temporary access to files beneath the shared root. With the whole base directory exposed, a caller that obtains - or coerces the app into issuing - such a grant can reach files far beyond what the sharing feature intended - the app's whole internal files directory for a bare `files-path`, up to the app's own private data when the shared root is the filesystem root.

RECOMMENDATION Replace the over-broad `<paths>` entry with the narrowest element + sub-path the sharing feature actually needs - e.g. `<files-path name="images" path="shared_images/"/>` or `<cache-path name="exports" path="exports/"/>` - never `root-path` or a bare `path="."` / `path="/"`. Scope each shared directory to a dedicated sub-folder and copy only the specific file to be shared into it before generating the URI.

LOW

No shouldShowRequestPermissionRationale calls (1 runtime permissions declared)

M6 · MASVS-PRIVACY · MASTG-TEST-0256

AAS-5D3CB2 privacy.permission_rationale.missing | privacy.permission_rationale v1.0.0

The app declares one or more runtime-grantable permissions but no class calls `Activity.shouldShowRequestPermissionRationale` or `ActivityCompat.shouldShowRequestPermissionRationale`. Users denied a permission once will be re-prompted without any context for why the app needs access - typically leading to permanent denial without informed consent, which Google's Permissions Best Practices and MASVS-PRIVACY-2 both call out as a privacy / UX smell.

RECOMMENDATION Before each `requestPermissions(...)` call, check `shouldShowRequestPermissionRationale(permission)`. When it returns true (the user denied previously), present a short UI explaining the specific feature that needs the permission and the user-visible benefit, then call `requestPermissions`. When false on a new install, going straight to the system prompt is fine. The `androidx.activity.result.contract.ActivityResultContracts.RequestPermission` lambda makes this two lines.

LOW

No root-detection signal found

M7 · MASVS-RESILIENCE · MASTG-TEST-0324

AAS-6F8E5A resilience.tamper_detection.root_detection_absent | resilience.tamper_detection v1.0.0

No reference to known root-detection libraries (RootBeer, SafetyNet, Play Integrity) and no su-path string constants in the DEX. For apps handling credentials, payments, or DRM content, MASVS-RESILIENCE-1 expects at least one runtime check + server-side attestation. Absence is a build-hygiene signal that the threat model didn't consider rooted devices.

RECOMMENDATION Add Play Integrity API for verified device attestation. Optionally pair with a client-side RootBeer check as fast-fail UX. Be aware that client-only checks are easily patched out - they exist to slow casual attackers and to feed a server-side signal, not to stop a determined reverse engineer. If the app doesn't handle sensitive data, this finding is safe to suppress.

LOW

No anti-debug / signature-check signal found

M7 · MASVS-RESILIENCE · MASTG-TEST-0324

AAS-65689A resilience.tamper_detection.anti_debug_absent | resilience.tamper_detection v1.0.0

No reference to `Debug.isDebuggerConnected`, `PackageManager.checkSignatures`, or other classic tamper-detection APIs. For apps handling sensitive data MASVS-RESILIENCE expects at least one active-debugger check + a signature self-check.

RECOMMENDATION If the threat model includes active reverse-engineering, add `Debug.isDebuggerConnected()` called at sensitive entry points (login, payment init). Pair with a signature self-check using `PackageManager.getPackageInfo(...GET_SIGNING_CERTIFICATES).signingInfo.apkContentsSigners[0].toByteArray()` and compare the SHA-256 against the expected release fingerprint. Like root-detection, client-only checks are bypassable; consider Play Integrity for the verifiable signal.

LOW

Untrusted Intent/Uri input reaches a filesystem path (new File()) in t0

M4 · MASVS-PLATFORM · MASTG-TEST-0357

AAS-98154A platform.path_traversal.tainted_path | platform.path_traversal v1.0.0

`String t0.l(Context, Uri)` builds a filesystem path via `java.io.File(String)` from data the taint analyzer traces back to incoming IPC input (`Intent.get*Extra` / `getData` / a `Uri` path part / `Bundle.getString`). The sender controls that string: a value like `../shared_prefs/prefs.xml` escapes the intended directory (path traversal, CWE-22), and even without `..` it can name a private file the feature never meant to expose - the MASTG-DEMO-0122 pattern, where an extra picks the file a `FileProvider` then shares. `t0` is not an exported component, so this is only reachable through intent redirection or the app's own intents (rated LOW for that reason). This is an intra-procedural taint result flagged for review - confirm the resolved path is not canonicalised and checked against the intended base directory before it is used.

```
LOCATION Lt0;->l(Landroid/content/Context; Landroid/net/Uri;)Ljava/lang/String; @70
```

RECOMMENDATION Never let IPC input name a file directly. Map an allow-listed key to a path the app chooses, or canonicalise and contain the result: `File f = new File(baseDir, input); if (!f.getCanonicalPath().startsWith(baseDir.getCanonicalPath() + File.separator)) throw new SecurityException();`. Reject input containing `..` or an absolute path. If the component need not be exported, set `android:exported="false"`. If this call site is already validated, suppress by rule_id.

Findings (6)

HIGH

ObjectInputStream.readObject called in zu

M4 · MASVS-CODE · MASTG-TEST-0337

AAS-170193 code.unsafe_deserialization.object_input_stream_read_object | code.unsafe_deserialization v1.1.0

`Object zu.c(Object, dk)` calls `ObjectInputStream.readObject`. Reconstructs an arbitrary class from the input byte stream. If the stream is attacker-controlled, gadget chains in the runtime classpath turn this into remote-code-execution (the Commons-Collections / `InvokerTransformer` family of exploits).

```
LOCATION Lzu;->c(Ljava/lang/Object; Ldk;)Ljava/lang/Object; @48
```

RECOMMENDATION Replace with a structured serialization protocol - JSON via Moshi / Kotlinx Serialization, or Protobuf - that explicitly validates each field's type. If interoperating with an existing Java protocol, install a `ObjectInputFilter` (Java 9+; Android API 35+) that whitelists the exact classes you expect.

HIGH

ObjectInputStream.readObject called in fj

M4 · MASVS-CODE · MASTG-TEST-0337

AAS-89057D code.unsafe_deserialization.object_input_stream_read_object | code.unsafe_deserialization v1.1.0

`xk fj.b(String, List)` calls `ObjectInputStream.readObject`. Reconstructs an arbitrary class from the input byte stream. If the stream is attacker-controlled, gadget chains in the runtime classpath turn this into remote-code-execution (the Commons-Collections / `InvokerTransformer` family of exploits).

```
LOCATION Lfj;->b(Ljava/lang/String; Ljava/util/List;)Lxk; @124
```

RECOMMENDATION Replace with a structured serialization protocol - JSON via Moshi / Kotlinx Serialization, or Protobuf - that explicitly validates each field's type. If interoperating with an existing Java protocol, install a `ObjectInputFilter` (Java 9+; Android API 35+) that whitelists the exact classes you expect.

MEDIUM

Intent.getSerializableExtra called in xb

M4 · MASVS-CODE · MASTG-TEST-0337

AAS-5F9065 code.unsafe_deserialization.intent_get_serializable_extra | code.unsafe_deserialization v1.1.0

`void xb.a(gc)` calls `Intent.getSerializableExtra`. Reads a Serializable object from intent extras. If the receiving component is exported, any app on the device can hand it crafted extras to deserialize.

```
LOCATION Lxb;->a(Lgc;)V @76
```

RECOMMENDATION On API 33+, prefer the new 2-arg signature `getSerializableExtra(name, Class)` which enforces a specific type. Better still: switch the extras to JSON strings or Parcelable types you control.

MEDIUM

Intent.getSerializableExtra called in tb

M4 · MASVS-CODE · MASTG-TEST-0337

AAS-D60460 code.unsafe_deserialization.intent_get_serializable_extra | code.unsafe_deserialization v1.1.0

`void tb.onStart()` calls `Intent.getSerializableExtra`. Reads a Serializable object from intent extras. If the receiving component is exported, any app on the device can hand it crafted extras to deserialize.

```
LOCATION Ltb;->onStart()V @168
```

RECOMMENDATION On API 33+, prefer the new 2-arg signature `getSerializableExtra(name, Class)` which enforces a specific type. Better still: switch the extras to JSON strings or Parcelable types you control.

MEDIUM

ContentProvider.openFile opens a URI-derived path without canonicalization in ta

M4 · MASVS-PLATFORM · MASTG-TEST-0357

AAS-14CFB6 platform.content_provider.openfile_no_canonical_check | platform.content_provider_audit v1.3.0

`ta` overrides `openFile` (the ContentProvider file-handle surface) but the method body contains no `File.getCanonicalPath` / `getCanonicalFile` call. When the file opened by this override is derived from the inbound `content://` URI without canonicalising the resolved path and confining it to an intended base directory, a caller can supply a traversal URI (`content://authority/../../../data/data/<pkg>/databases/...`) and read or write files the provider's UID can reach - escaping the directory the provider meant to expose. This is a heuristic (absence of a canonicalization callsite in the same method): containment may be enforced another way, so confirm the override validates the path before treating it as exploitable.

```
LOCATION Lta;->openFile(Landroid/net/Uri; Ljava/lang/String;)Landroid/os/ParcelFileDescriptor; @0
```

RECOMMENDATION Resolve the requested file relative to a fixed base directory, then call `File.getCanonicalPath()` (or `getCanonicalFile()`) on both the base and the candidate and verify the candidate `startsWith` the base's canonical path before opening it. Reject any URI whose canonical target escapes the base. Prefer narrowing the exposed surface entirely - `FileProvider` with a tightly scoped `file_paths` XML and exact-`path` grants - over hand-rolled `openFile` path handling.

LOW

StrictMode policy configured in release build

M8 · MASVS-CODE · MASTG-TEST-0041 · MASTG-TEST-0263

AAS-9F454F resilience.verbose_logs.strict_mode_in_release | resilience.verbose_logs v1.0.0

This release build (debuggable=false) installs a `StrictMode` policy via `setThreadPolicy` / `setVmPolicy` / `setDefaultPolicy`. StrictMode is a development-time diagnostic that detects disk/network on the main thread and leaked closeables; in production it adds overhead and its `penaltyLog()` violations are written to logcat, occasionally exposing internal file paths or query details. Its presence in a release build is a hygiene smell - the developer likely didn't gate it behind `BuildConfig.DEBUG`. Reported as an observation for review, not a confirmed vulnerability.

RECOMMENDATION Wrap StrictMode setup in `if (BuildConfig.DEBUG) { ... }` so R8 strips it from release builds, or remove it entirely. If a policy must ship in release, avoid `penaltyLog()` and prefer `penaltyDeath()`-free, log-free penalties to keep diagnostic details out of logcat.

07 App inventory

Bundled components

Component	Kind	Version	Category	PII
Flutter engine	library			
Flutter local notifications	library			
Google Play Services	library			

Signing

Schemes: v2 | Key: | Issues: none

App profile

minSdk/target: 24/36 | Minification: heavy (0.984) | NSC: False

08 Security posture

1 present | 7 absent | 0 partial

Control	State	What it means
Certificate pinning - makes the app trust only your servers' own certificate, rejecting any forged stand-in	absent	Without it, anyone who can plant a trusted-looking certificate on the device — a corporate proxy, malware, or a hacked certificate authority (one of the outside bodies phones trust to vouch for a site) — can quietly read or change the app's traffic, including logins and payment data. Pinning makes the app reject those substitute certificates, so only your real server can complete the connection. How to add it: Add pinning with OkHttp's CertificatePinner or TrustKit, or declare a <pin-set> in your Network Security Config, listing your server's public-key hashes. Include a backup key and a rotation plan so an expiring certificate doesn't lock users out.
GMS security provider - on launch, asks Google Play services to update the phone's HTTPS security code	absent	Older phones ship with outdated code for securing internet connections, and it has publicly known bugs. Without this call the app keeps relying on that old code, so an attacker on the same Wi-Fi could exploit those bugs to read or alter the app's supposedly private HTTPS traffic. How to add it: Call Google Play services' ProviderInstaller.installIfNeeded() (or the async installIfNeededAsync()) early in app startup, before any network requests, so an up-to-date security provider replaces the operating system's old one. Catch the GooglePlayServicesRepairable and GooglePlayServicesNotAvailable exceptions it can throw, so devices without Play services still run instead of crashing.
Screen-capture (FLAG_SECURE) - blocks screenshots and screen recording, and hides the screen from the app-switcher preview	absent	0/1 activities protected (1 unprotected: MainActivity) Without it, screen-recording malware, a stray screenshot, or the thumbnail Android keeps for the app switcher can capture whatever is shown on a sensitive screen — a password, a one-time code, an account balance, a private message. That recents thumbnail even lingers after the user leaves the screen, so anyone holding the unlocked phone can read it. How to add it: Add the flag to every Activity that shows sensitive data (login, payment, personal info) — for example getWindow().addFlags(WindowManager.LayoutParams.FLAG_SECURE) — not just some of them. Note this scan only confirms the flag is present in the code; it can't prove every screen actually applies it at runtime, so verify the sensitive screens by hand.
Tapjacking protection - makes the app ignore taps passing through another app's window layered on top	absent	0/1 activities protected Another app can draw its own window on top of yours — sometimes invisible, sometimes disguised as a harmless button or a game. You tap what you see, but the tap passes straight through to your app's real button underneath, so a tap you thought was "Cancel" actually hits "Approve" on a payment or a sensitive setting. With this protection on, your app rejects any tap that lands on it while another app's window is covering it. How to add it: Set android:filterTouchesWhenObscured="true" on the sensitive views (or on the screen's root layout), or call setFilterTouchesWhenObscured(true) in code, so the screen drops any tap that arrives while another app's window is covering it. Apply it to every screen that confirms actions or handles logins and payments — not just a few — to move from partial to full coverage.
Anti-debug - detects a debugger attaching so the app can shut down or clear secrets	absent	A debugger is a tool that freezes a running app and steps through it one line at a time. With no checks to spot one, an attacker can attach a debugger, read your keys and tokens straight out of live memory, and watch exactly how the app works while it runs. How to add it: Add runtime checks such as Debug.isDebuggerConnected(), plus timing comparisons that catch the slowdown a debugger causes, and make the app exit or wipe secrets when one trips. Combine several, since any single check is easy to bypass. Note that this scan only confirms the checking code is present — not that it actually runs or resists a determined attacker.
Root detection - checks whether the phone is 'rooted' (unlocked so apps can bypass its safety limits)	absent	On a rooted phone, hostile software gains full control and can read the app's memory, capture the data it handles, and change how it behaves while running. Detecting that lets the app respond — warn the user, limit sensitive features, or refuse to run — instead of trusting a compromised device. How to add it: Add a root-detection check — Google's Play Integrity API (its device-integrity verdict) or a maintained root-detection library — then decide what happens when it detects root: warn the user, disable sensitive features, or block access. Remember a static scan only confirms the check is present in the code, not that it runs correctly, so also test it end-to-end on a rooted device.
Signature self-check - reads its own signing certificate at runtime to catch copies re-signed by someone else	present	Attackers grab a popular app, add malware or ad-fraud code, re-sign it, and hand out the clone on third-party app stores. Because they don't have your private signing key, their re-signed copy carries a different certificate — so an app that compares its own certificate against the expected one can notice it's been tampered with and refuse to run instead of running the attacker's code.

Control	State	What it means
Forced update - includes Google Play's in-app update feature, which can prompt or require an upgrade	absent	Without a built-in update mechanism, people can keep running an old, unpatched version indefinitely — so a security bug you already fixed stays exploitable on every phone that never updates on its own. You have no in-app way to require everyone onto the fixed build; Google Play's own auto-update may catch up eventually, but you can't count on it or force it. How to add it: Add Google Play's In-App Update API (the AppUpdateManager class) and use its immediate flow, which blocks users from continuing until they install a critical update. A scan only sees that this API is referenced — it can't tell whether you wired up the optional (flexible) flow or the forcing (immediate) one — so if enforcement matters, confirm the immediate flow and test it end to end. As a backstop, also reject outdated clients with a minimum-version check on your own server.

09 Methodology, scope & limitations

This assessment was produced by Atomic App Scanner using automated **static and bytecode analysis** of the single submitted APK (SHA-256 9aec4ef6ef7b58500c06fc5336437fab8e121637e81de7d8e02fa1c4313733de, scan 1255b163-4786-410a-a68e-23131bfc3a14). Coverage spans 380+ individual checks across 86 analyzers, mapped to the OWASP Mobile Top 10 (2024), the eight OWASP MASVS control groups, and MASTG test references.

Explicitly NOT covered: no dynamic or runtime analysis; no live network/TLS interception; no server-side or backend testing; no business-logic review; and no manual penetration testing. A present hardening control is confirmed to exist in code, not proven to execute correctly at runtime.

Scan completeness: Scan completed with full static coverage.

10 Glossary

Severity

CRITICAL direct serious exposure; HIGH likely material impact; MEDIUM meaningful weakness; LOW minor / defence-in-depth; INFO informational.

Vulnerability vs observation

A vulnerability is an actionable weakness. An observation is contextual - inventory (what the app is) or posture (which hardening controls are present) - not itself a defect.

False positives

Findings a reviewer marked as not applicable render greyed with a FALSE POSITIVE badge and are excluded from the severity band, but kept visible.

Frameworks

OWASP Mobile Top 10 - the 2024 mobile security categories. MASVS - Mobile Application Security Verification Standard (eight control groups). MASTG - Mobile Application Security Testing Guide. "Tested - no findings" means a category was checked and nothing was flagged, not a certification of compliance.