

## MOBILE APPLICATION PENETRATION TEST · COMPOSITE SAMPLE

# Composite Sample Mobile Application Penetration Test

Prepared for [Client] by PXLSecurity LTD.

**This is a sample, not a client deliverable.** It is a composite assembled from the shape of several anonymised PXLSec engagements against iOS, Android and cross-platform applications and the APIs behind them. Every finding has been rewritten with synthetic, illustrative data on reserved example names, packages and addresses. It describes no single real client, and no real client, vendor, product, host, person or credential appears anywhere in it.

|                    |                                                      |
|--------------------|------------------------------------------------------|
| Document reference | PXL-2026-0311                                        |
| Version            | 1.0 (Final)                                          |
| Classification     | Confidential — sample, contains no client data       |
| Assessment type    | Mobile application penetration test (grey box)       |
| Platforms          | iOS, Android and Windows release builds              |
| Assessed against   | OWASP MASVS, using the MASTG as the testing guide    |
| Engagement window  | [Engagement dates]                                   |
| Prepared by        | PXLSecurity LTD. · Sofia, Bulgaria · VAT BG203239331 |

## CONTENTS

# What is in this report

Written to be read two ways: section 1 stands alone, and the detailed findings carry everything an engineer needs to reproduce and fix each issue.

|          | SECTION                                                                                                                                        | PAGE      |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Executive summary</b><br>What was tested, the headline risk picture and the three things to fix first — written for a non-technical reader. | <b>3</b>  |
| <b>2</b> | <b>Engagement scope</b><br>Platforms, builds, accounts, the API behind the app, and the assumptions the result depends on.                     | <b>5</b>  |
| <b>3</b> | <b>Methodology</b><br>The five stages, the eight MASVS control groups covered, and the safety rules we test under.                             | <b>6</b>  |
| <b>4</b> | <b>Severity ratings</b><br>The scale, and why a resilience bypass is not automatically critical.                                               | <b>8</b>  |
| <b>5</b> | <b>Summary of findings</b><br>Every finding with its identifier, severity, MASVS group and status, in remediation order.                       | <b>9</b>  |
| <b>6</b> | <b>Detailed findings</b><br>One section per finding: description, evidence, reproduction, impact and a specific fix.                           | <b>10</b> |
| <b>7</b> | <b>Remediation plan</b><br>The fixes grouped into waves, with an owner and a sequence.                                                         | <b>24</b> |
| <b>8</b> | <b>Retest and closure</b><br>What the retest covers and what the closure letter states.                                                        | <b>26</b> |
| <b>A</b> | <b>Appendix A — Builds, accounts and tooling</b><br>What we asked for and why each item mattered.                                              | <b>27</b> |
| <b>B</b> | <b>Appendix B — Out of scope, and what this test is not</b><br>What was excluded, and the adjacent work this engagement does not perform.      | <b>28</b> |
| <b>C</b> | <b>Appendix C — References and what they are</b><br>The MASVS, the MASTG and the rest, described accurately, with no certification implied.    | <b>29</b> |
| <b>D</b> | <b>Appendix D — How to read this report</b><br>Who should read which section, and how to act on it.                                            | <b>30</b> |

**A note on what this sample is.** Findings F-01 to F-09 are drawn from the shape of several different anonymised engagements and presented here as if they came from one, so that a prospective client can see the full range of the work in a single read. Every transcript, package name, identifier, account and score is synthetic, and no device model, framework version or commercial tool is named as a client's.

## SECTION 1

## Executive summary

PXLSecurity LTD. was engaged by [Client] to carry out a manual penetration test of a mobile application and the API behind it. This section is written for a non-technical reader. It says what was tested, how serious the result is and what to do first.

### What was tested

Senior engineers tested the application by hand over the agreed window, working from the shipped release builds rather than from source, because that is what an attacker has. The composite scope behind this sample covered the iOS, Android and Windows builds of a single cross-platform product, the data the application stores on the device, its behaviour under instrumentation on a device the tester controlled, and the API the application talks to. Testing was structured around the OWASP Mobile Application Security Verification Standard, and each finding below names the control group it falls under.

Two things are worth stating plainly before the results, because they shape how the findings should be read. First, a mobile application is the one part of a system that is handed to the attacker: once published, the binary, the data it writes and the running process are all available to anybody who installs it, so a weakness compiled into the artefact cannot be withdrawn by a change on the server. Second, controls that live only on the device — integrity checks, pinning, anti-tampering — raise the cost of an attack and do not prevent one. We expect to defeat them, and we report what they were protecting rather than the fact that they fell.

### The headline

|                      |                      |                  |                       |                          |
|----------------------|----------------------|------------------|-----------------------|--------------------------|
| <b>9</b><br>findings | <b>1</b><br>critical | <b>3</b><br>high | <b>3</b><br>platforms | <b>8</b><br>MASVS groups |
|----------------------|----------------------|------------------|-----------------------|--------------------------|

The result is driven by one finding. The application authenticates to its backend using a client secret compiled into the bundle, identical in every installed copy and present on all three platforms. Recovering it required no exploitation and no account: the release package was unpacked as an archive and the value read out of it. That single issue converts a published application into a working credential for an internal data API, held by everyone who has ever downloaded it.

Behind it sit three high-severity issues that share a pattern: each is a control that exists and is almost right. The offline database is encrypted, but with a key the device itself can reproduce. Certificate pinning is implemented, but a management-profile default switches it off silently in the configuration most customers actually deploy. A local helper service is protected by a token, and then hands that token to any caller that asks. None of these is an absent control; all three are controls that do not hold against the threat they were built for.



Findings by severity. Counts are from this composite sample and are not a benchmark of any population of applications.

### The three things to do first

- 1 **Rotate the backend credential, today, before any code change.** It is live, it is public, and rotation is the only action that reduces exposure immediately. Expect that rotating it will require a client release, and plan that release now rather than after.
- 2 **Stop deriving the offline database key from the device.** Move it into hardware-backed storage so that holding the device does not yield the data on it. This is the change that makes a lost or stolen device a contained event.
- 3 **Invert the pinning default.** When the management profile is silent, enforce pinning. The configuration a customer arrives at by following the documentation should be the safe one, not the permissive one.

**What this does not say.** This report describes what we were able to do within the agreed scope and window. It is not a certification, an audit opinion or a statement that the application is free of other weaknesses, and PXLSecurity LTD. is not an auditor, assessor or certification body. Structuring the work around the OWASP MASVS is a way of making coverage legible; it is not an assessment against the standard and implies no scheme membership.

## SECTION 2

## Engagement scope

What was in scope, what we were given, and the assumptions the result rests on. A finding is only as meaningful as the scope it was found in.

|                        |                                                                                                                                                                           |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Targets</b>         | The iOS, Android and Windows release builds of a single cross-platform application, plus the backend API it consumes.                                                     |
| <b>Artefacts</b>       | Signed release builds supplied by [Client], matching the versions in public and managed distribution at the time of testing.                                              |
| <b>Assessment type</b> | Grey box. No source code; test accounts and a description of the roles were provided. The artefact was treated as an attacker would treat it.                             |
| <b>Accounts</b>        | Two accounts per role in an isolated test tenant, so that cross-account and cross-tenant behaviour could be proved rather than inferred.                                  |
| <b>Devices</b>         | Tester-controlled physical devices and emulators, in both default and privileged states, plus a managed-deployment profile for the enterprise configuration.              |
| <b>Window</b>          | [Engagement dates], with the backend testing confined to the agreed non-production environment except where noted.                                                        |
| <b>Standards</b>       | OWASP MASVS as the verification standard; OWASP MASTG as the testing guide; OWASP API Security Top 10 for the server side.                                                |
| <b>Exclusions</b>      | See Appendix B. In short: no denial of service, no testing of the operating systems themselves, no attack on the app stores or the customer's device-management platform. |

### Assumptions this result depends on

- The builds tested are the builds that ship. If the release pipeline rebuilds or re-signs after the artefact we were given, the findings describe a different binary from the one your customers install.
- The managed-deployment profile we were given reflects how customers actually deploy. F-03 turns on that profile's defaults, so a materially different profile changes that finding's severity.
- The test tenant behaves like a production tenant. Where a server-side control differs between the two, a finding proved in test may behave differently in production — we flag any finding where this is in question, and none of the findings below are.
- Nothing in this report assumes a compromised operating system. Where a finding requires a privileged device state, the finding says so and is rated accordingly.

## SECTION 3

## Methodology

How the engagement ran, what the testing covered, and the rules it ran under.



|                              |                                                                                                                               |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>Scope &amp; builds</b>    | Platforms, release artefacts, accounts, devices and rules of engagement agreed in writing before anything is touched.         |
| <b>Static analysis</b>       | The shipped artefact unpacked and reverse-engineered: strings, key material, endpoints, manifest, exported surface and logic. |
| <b>Dynamic &amp; runtime</b> | Instrumented testing on a tester-controlled device: hooking, traffic interception, storage inspection and control bypass.     |
| <b>API &amp; chaining</b>    | What the client exposes is followed through into the service behind it, and findings are chained to establish real impact.    |
| <b>Report &amp; retest</b>   | Findings mapped to the MASVS control groups and prioritised, then verified at retest and confirmed in writing.                |

### Why the order matters

The sequence above is the attacker's, not an auditor's. Reading the artefact comes first because it is free, silent and needs no account: anything recovered there is available to everybody who installs the application, which is why F-01 is the most serious finding in this report despite being the least technically demanding. Instrumentation comes second, because it establishes what the device-side controls are actually protecting. The API comes last, because by that point we know what the client believes and can test whether the server believes it too.

### Coverage against the MASVS control groups

The OWASP Mobile Application Security Verification Standard organises mobile security into control groups. We structure both the testing and this report around them, so that coverage is legible and gaps are visible. The right-hand column shows which findings in this sample fell into each group — a group with no findings was still tested.

| Control group          | What it covers                                                         | Findings         |
|------------------------|------------------------------------------------------------------------|------------------|
| <b>MASVS - STORAGE</b> | Data stored on the device, and what leaves it in logs and backups      | F-01, F-02, F-06 |
| <b>MASVS - CRYPTO</b>  | Key management and the correct use of cryptographic primitives         | F-02             |
| <b>MASVS - AUTH</b>    | Authentication and authorisation, including local and biometric gating | F-08             |
| <b>MASVS - NETWORK</b> | Transport security and certificate validation                          | F-03             |

| Control group      | What it covers                                                              | Findings   |
|--------------------|-----------------------------------------------------------------------------|------------|
| MASVS - PLATFORM   | Interaction with the platform: IPC, exported components, WebViews           | F-04, F-07 |
| MASVS - CODE       | Build configuration, dependencies and what the artefact carries             | F-01       |
| MASVS - RESILIENCE | Integrity checks, anti-tampering and the cost of reverse engineering        | F-05, F-09 |
| MASVS - PRIVACY    | Handling of personal data and what the application discloses about its user | F-06       |

*Structuring testing around the MASVS is not an assessment against it. PXLSec is not a certification body and this table is a coverage aid, not a conformance claim.*

## How tooling is used

Tools unpack, decompile, instrument and intercept; they do not decide what matters. Static analysis of a mobile artefact produces a large volume of low-value output — every permission, every exported component, every string that resembles a key — and the work is in separating the three that are real from the three hundred that are not. Every finding in this report was confirmed by hand against a running application, and a finding that could not be demonstrated is not in the report.

## Safety rules we test under

- Read-only confirmation wherever a read proves the point. Where a credential's validity is the finding, a status code establishes it and the response body is discarded unread.
- No customer data is retrieved, copied or retained. Where a store was opened, a row count was read and nothing else.
- State changes are performed against synthetic records in an isolated test tenant, and reverted.
- No denial of service, and no technique whose failure mode is a broken device or a corrupted store.
- Bypass techniques are not published. Where a control was defeated, the report states what the control protected and why it fell, not the script that did it.

## SECTION 4

## Severity ratings

What each rating means here, and how CVSS informs it without deciding it.

| Rating        | What it means in this report                                                                                                                                                                        |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Critical      | Direct, unauthenticated access to customer data or to an authority that reaches it, available to anyone who installs the application. Fix before the next release, and rotate anything exposed now. |
| High          | A control that exists but does not hold, leading to data exposure or account access under conditions an attacker can realistically arrange.                                                         |
| Medium        | A weakness that materially lowers the cost of attack or compounds another finding, but does not on its own reach customer data.                                                                     |
| Low           | A narrow exposure requiring conditions the attacker cannot easily arrange, or a feature that does not deliver the protection it implies.                                                            |
| Informational | An observation worth acting on that is not a vulnerability. No score is assigned.                                                                                                                   |

### Why CVSS does not decide the rating

A CVSS vector is given for each scored finding because readers ask for one and because it travels well between tools. It is an input, not the answer. CVSS has no way of expressing the fact that a secret inside a shipped artefact cannot be revoked, which is precisely what makes F-01 the finding it is; nor that F-03's severity comes from the configuration most customers deploy rather than from the difficulty of the attack. Where our rating and the arithmetic diverge, the rating reflects the business consequence and this report says why.

**A note specific to mobile.** Reports in this field often rate every bypass of a resilience control as critical, because a bypass is dramatic and easy to demonstrate. We do not. Root detection, pinning and anti-tampering raise the cost of attack on a device the attacker already controls; defeating them is expected. F-05 is rated Medium for that reason, and the rating describes what the bypass gave access to rather than the fact that it worked. A report that leads with a jailbreak bypass as its headline finding is usually padding.

## SECTION 5

## Summary of findings

Every finding, in the order we would fix them.

| ID   | Finding                                                              | Severity      | MASVS             | Status |
|------|----------------------------------------------------------------------|---------------|-------------------|--------|
| F-01 | Backend API credentials compiled into the shipped application bundle | Critical      | STORAGE · CODE    | Open   |
| F-02 | Offline database key derivable from material readable on the device  | High          | STORAGE · CRYPTO  | Open   |
| F-03 | Certificate pinning disabled by a management-profile default         | High          | NETWORK           | Open   |
| F-04 | Embedded local service discloses its own authorisation token         | High          | PLATFORM          | Open   |
| F-05 | Root and jailbreak detection defeated at a single decision point     | Medium        | RESILIENCE        | Open   |
| F-06 | Session token written to the device log and included in backups      | Medium        | STORAGE · PRIVACY | Open   |
| F-07 | Exported component performs a state change without authentication    | Medium        | PLATFORM          | Open   |
| F-08 | Biometric prompt gates the interface rather than the credential      | Low           | AUTH              | Open   |
| F-09 | Obfuscation applied, but not across the security-relevant code       | Informational | RESILIENCE        | Open   |

Severity counts: 1 critical, 3 high, 3 medium, 1 low, 1 informational. Status is as at the date of this report; the retest letter records closure.

## SECTION 6

## Detailed findings

One section per finding: severity and illustrative CVSS, affected component, what was found, the evidence, a reproduction path, the impact and a fix.

**About the evidence below.** In a real report each capture is a redacted screenshot of the engineer's own session, with the full request and response pairs in the appendix. Every capture here is a synthetic reconstruction — the operations are representative; the packages, hosts, identifiers, keys and output are invented. **No working payload, hook script or bypass recipe appears anywhere in this document:** reproduction is written at the level a vendor advisory uses.

**CRITICAL****F-01 Backend API credentials compiled into the shipped application bundle**

| SEVERITY        | CVSS v3.1 (ILLUSTRATIVE)                       | AFFECTED COMPONENT                                          | MASVS                 |
|-----------------|------------------------------------------------|-------------------------------------------------------------|-----------------------|
| <b>Critical</b> | <b>9.1</b> AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N | Release builds for all three platforms — application bundle | <b>STORAGE · CODE</b> |

**Class:** Use of hard-coded credentials (CWE-798) · insufficiently protected credentials (CWE-522) · **Reference:** OWASP MASVS-STORAGE-2 and MASVS-CODE-4 · MASTG static analysis of the application package · OWASP Mobile Top 10 M1 (improper credential usage) · **Status:** Open · *Composite element: cross-platform application distributed to other organisations*

### Description

The application authenticates to its backend with a client identifier and secret that are compiled into the bundle. They are not in a configuration file fetched at runtime, not held in the platform keystore and not issued per installation: they are constants in the shipped artefact, identical in every copy, and the same three values appear in the iOS, Android and Windows builds.

Recovering them required no exploitation. The release package was unpacked as an archive and its string table read. The credential proved live against the production API, returning an authenticated response to a read-only probe. This is the finding that defines the engagement, because it is not reachable by the usual remedies: the secret cannot be revoked by a server-side configuration change without breaking every installed copy, and it remains valid on every device that does not take an update, which in a distributed application is a population the vendor does not control.

## Evidence

### Backend API credentials compiled into the shipped application bundle

SYNTHETIC EXAMPLE

Strings recovered from the public release build authenticate against an internal data API.

```
analyst@wkstn ~$ bash

analyst@wkstn:~/mobile$ unzip -q haldon-release.apk -d ./unpacked
Synthetic transcript — invented package, host and key material
analyst@wkstn:~/mobile$ strings unpacked/assets/app.bundle | grep -iE 'api_(key|secret)'
```

```
api_base      = https://api.example.test/v3
api_client_id = haldon-mobile-prod
api_secret    = sk_live_7f3a...<truncated - fabricated value>

# the same three values appear in the iOS and Windows builds

analyst@wkstn:~/mobile$ curl -s -o /dev/null -w '%{http_code}' \
-u haldon-mobile-prod:sk_live_7f3a...<truncated - fabricated value> \
https://api.example.test/v3/records?limit=1

200  - credential is live and authenticated

Read-only check only. No record was retrieved, stored or exfiltrated.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-01 — the release package is unpacked, the credential is read out of the bundle, and a read-only probe confirms it authenticates against the production API. Illustrative reconstruction using synthetic data on the reserved example.test domain and the documentation package namespace com.example.haldon; not client evidence.

## Reproduction steps

- 1 Obtain the release artefact the way any user would — from the public listing or the managed distribution catalogue. No device compromise, no account and no prior access is required.
- 2 Unpack the package as an archive and search the bundled assets and string tables for credential-shaped constants. In the sample transcript this returns the API base, the client identifier and the client secret directly.
- 3 Confirm the credential is live with a single read-only request that asks for the smallest possible response. The sample shows 200 — authenticated against the production API.
- 4 Stop there. The response body was discarded unread and no record was retrieved, stored or exfiltrated; the finding is the credential's validity, and that is established by the status code alone.

## Impact

Anyone who downloads the application holds a working credential for an internal data API. That is the entire population of the app's distribution, which for an application published to other organisations is unbounded and unauthenticated. The access is indistinguishable from legitimate application traffic, so it produces no anomaly for anyone to detect, and it persists independently of the vendor's own session handling: revoking a user's account does not revoke this. Because the same values ship on all three platforms, there is no partial exposure and no platform that can be treated as unaffected.

## Recommendation

- 1 Treat the credential as compromised and rotate it now, ahead of any code change. Review the API's own access logs for use from outside the expected client population, and expect that a rotation will require a client release.
- 2 Remove the shared secret from the artefact entirely. The application should authenticate as the user, with a short-lived token issued on sign-in, rather than as itself with a constant the bundle carries.

- 3 Where the client genuinely needs its own identity, issue it per installation at first run and store it in the platform keystore or keychain, so that recovering one device's credential does not yield every device's.
- 4 Add a release gate that scans the built artefact — not the source tree — for credential-shaped constants, so this class of defect is caught in the thing that actually ships.

HIGH

**F-02 Offline database key derivable from material readable on the device**

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                | AFFECTED COMPONENT                                    | MASVS            |
|----------|-----------------------------------------|-------------------------------------------------------|------------------|
| High     | 7.4 AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N | Android and iOS builds — encrypted offline data store | STORAGE · CRYPTO |

**Class:** Use of a risky cryptographic algorithm (CWE-327) · insufficient key derivation (CWE-916) · **Reference:** OWASP MASVS-CRYPTO-1 and MASVS-STORAGE-1 · MASTG testing of data storage and key management · OWASP Mobile Top 10 M10 (insufficient cryptography) · **Status:** Open · *Composite element: application caching records on the device*

**Description**

The application supports working offline and caches records in an encrypted local database. The encryption is real and the library behind it is sound; the weakness is in where the key comes from. It is derived by hashing a device identifier concatenated with a fixed application string — no salt, no key-derivation function and a single iteration.

Every input to that derivation is readable on the device the database sits on, which means the key travels with the thing it is supposed to protect. Anyone with the device, a backup of it or filesystem-level access can reproduce the key and open the store. The protection holds against someone who copies the database file alone and nothing else, which is not the threat the feature was designed against.

**Evidence****Offline database key derivable from material readable on the device**

SYNTHETIC EXAMPLE

The at-rest encryption key is derived from an unsalted device identifier, so possession of the device is enough to reproduce it.

```
analyst@wkstn ~$ ls -l /data/data/com.example.haldon/databases/
-rw-r--r-- 1 offline.db 4.1M encrypted

# key derivation recovered by decompiling the storage class
key = SHA256( ANDROID_ID + "haldon" )    no salt, no KDF, 1 iteration

analyst@wkstn ~$ python3 derive_key.py --android-id 9a1c...<truncated - fabricated value>
derived key: 4e9b...<truncated - fabricated value>

analyst@wkstn ~$ sqlcipher offline.db
sqlite> PRAGMA key = '4e9b...<truncated - fabricated value>';
sqlite> SELECT count(*) FROM contacts;

ok - store opened; row count read, no rows retrieved

Hardware-backed key storage would make the key unreproducible off the device.
```

*Illustrative example — synthetic data, not client evidence.*

*Evidence for F-02 — the derivation is recovered from the decompiled storage class, the key is reproduced from a device identifier, and the store opens. Illustrative reconstruction using synthetic identifiers and fabricated key material; not client evidence.*

### Reproduction steps

- 1 Locate the encrypted store in the application's private data directory and confirm it is encrypted rather than merely inaccessible.
- 2 Decompile the storage class and read the derivation. In the sample it is a single unsalted hash over a device identifier and a constant string.
- 3 Reproduce the key from the device identifier alone, with no secret from the user and no material from the server.
- 4 Open the store with the derived key and confirm access by reading a row count. Testing stopped at the count: no rows were read, extracted or retained.

### Impact

The offline store holds the records the user has worked with, which is often a substantial subset of what their account can reach. A lost or stolen device, a device handed on without a wipe, or an unencrypted backup extracted from a workstation all yield that data in plaintext. Because the key derivation is identical across installations, the technique is not specific to one device and scales to any device recovered. The encryption satisfies a checklist that asks whether data at rest is encrypted while providing very little of the protection that question is meant to establish.

### Recommendation

- 1 Move the key into hardware-backed storage — the Android Keystore or the iOS Keychain with an appropriate protection class — so that it is generated on the device, never leaves the secure element and cannot be reproduced from anything readable.
- 2 If a user secret must take part in the derivation, use a current password-based key-derivation function with a per-installation random salt and a deliberately costly iteration count. Never derive a key from device identifiers alone.
- 3 Bind the key's release to user authentication, so that unlocking the device is required before the store can be opened, and re-derive on credential change.
- 4 Reduce what is cached. The strongest control available here is to hold fewer records offline and expire them sooner; the data that is not on the device needs no key at all.

HIGH

### F-03 Certificate pinning disabled by a management-profile default

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                | AFFECTED COMPONENT                                          | MASVS   |
|----------|-----------------------------------------|-------------------------------------------------------------|---------|
| High     | 7.4 AV:A/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N | All platforms — transport security under managed deployment | NETWORK |

**Class:** Improper certificate validation (CWE-295) · insecure default initialisation (CWE-1188) · **Reference:** OWASP MASVS-NETWORK-1 and MASVS-NETWORK-2 · MASTG testing of network communication · OWASP Mobile Top 10 M5 (insecure communication) · **Status:** Open · *Composite element: application deployed through managed device fleets*

### Description

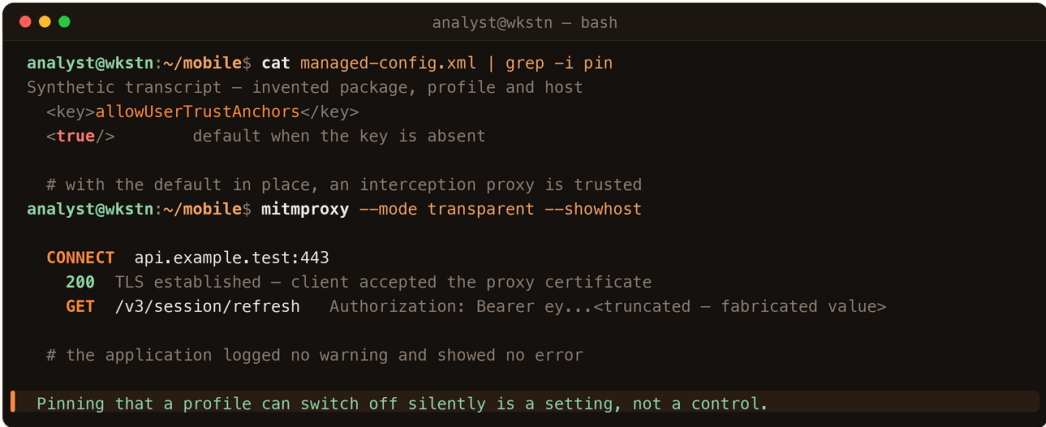
The application implements certificate pinning, and in a default installation it works. The problem appears in the configuration the product is actually deployed under. When the application is installed through a customer's device-management platform, a managed-configuration key governs whether user-installed trust anchors are honoured, and the value that applies when the key is absent is the permissive one.

Customer administrators deploying the application do not normally set a key they have not been told about, so the permissive default is the configuration the product runs under in the field. With it in place the pinning is not enforced, an interception proxy is trusted, and the full session — including the bearer token on the refresh call — is readable. Nothing in the application surfaces the change: no warning, no error, no indicator that a control the vendor believes is active has been switched off by a default nobody chose.

## Evidence

**Certificate pinning disabled by a management-profile default**SYNTHETIC EXAMPLE

A managed deployment default turns pinning off without surfacing the change, allowing full interception of the session.



i Illustrative example — synthetic data, not client evidence.

Evidence for F-03 — the managed-configuration default is read from the profile, and with it in place the client accepts an interception proxy and the session is readable. Illustrative reconstruction using a synthetic profile and the reserved example.test domain; not client evidence.

## Reproduction steps

- 1 Read the managed-configuration schema the application publishes and identify the key governing trust anchors, together with the value that applies when it is unset.
- 2 Deploy the application under a management profile that does not set the key — the configuration a customer administrator produces by following the documentation as written.
- 3 Place an interception proxy in the path and present a certificate chaining to a trust anchor installed on the device. In the sample the client completes the handshake and proceeds.
- 4 Observe the session traffic, confirming that the authorisation material is readable. Testing was performed against a test account on an isolated network segment; no production session was intercepted.

## Impact

On any network the attacker can position themselves on — a shared office wireless network, a captive portal, a compromised home router — the session is readable and modifiable. That yields the bearer token, which is reusable away from the device and away from the network, so a transient position on the network converts into persistent account access. The severity is driven less by the technical difficulty, which is real, than by the silence: the deployment most customers run is the vulnerable one, and neither the customer nor the vendor has any signal that pinning is not in effect.

## Recommendation

- 1 Change the default. When the managed-configuration key is absent the application should enforce pinning, so that the configuration a customer gets by not acting is the safe one.

- 2 Surface the state. If a profile does disable pinning, the application should say so in a place an administrator will see, and refuse to handle credentials while a user-installed anchor is in the chain.
- 3 Document the key explicitly in the deployment guidance, with the security consequence stated plainly rather than described as a compatibility option.
- 4 Make the token less valuable on its own: shorten its lifetime, bind it to the device or installation where the platform allows, and require re-authentication for sensitive operations rather than treating a valid bearer token as sufficient.

HIGH

**F-04 Embedded local service discloses its own authorisation token**

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                | AFFECTED COMPONENT                                                      | MASVS    |
|----------|-----------------------------------------|-------------------------------------------------------------------------|----------|
| High     | 7.1 AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N | Android and Windows builds — loopback bridge for the embedded web layer | PLATFORM |

**Class:** Exposure of sensitive information to an unauthorised actor (CWE-200) · missing authentication for a critical function (CWE-306) · **Reference:** OWASP MASVS-PLATFORM-1 and MASVS-PLATFORM-3 · MASTG testing of platform interaction and local IPC · OWASP Mobile Top 10 M4 (insufficient input/output validation) · **Status:** Open · *Composite element: hybrid application with a native-to-web bridge*

**Description**

The hybrid architecture runs a small HTTP service on the loopback interface so the embedded web layer can reach native capability. The service is protected by a bearer token, which is the correct instinct. The defect is that one of its own endpoints returns that token to any caller, without requiring it.

A loopback port is not a privacy boundary on a mobile device: any installed application can connect to it, with no special permission and no user prompt. So the sequence available to an unprivileged application on the same device is to connect, ask the bridge for its configuration, receive the token, and then use the token to reach everything the bridge exposes — which is the authenticated user's data, through a component that was never designed to be an external interface.

**Evidence****Embedded local service discloses its own authorisation token**

SYNTHETIC EXAMPLE

A loopback helper the app runs for its web layer returns the token that protects it, to any unauthenticated caller on the device.

```
analyst@wkstn ~ bash

analyst@wkstn:~/mobile$ adb shell netstat -tlnp | grep 127.0.0.1
Synthetic transcript — invented package, port and token
tcp 127.0.0.1:18642 LISTEN com.example.haldon

# any app on the device can reach a loopback port
analyst@wkstn:~/mobile$ curl -s http://127.0.0.1:18642/bridge/config

{"version":"3.2.1","bridgeToken":"brg_c41f...<truncated - fabricated value>"}

# that token is the only thing guarding the bridge
analyst@wkstn:~/mobile$ curl -s -H 'X-Bridge-Token: brg_c41f...<truncated - fabricated value>' \
http://127.0.0.1:18642/bridge/contacts?limit=1

200 - authorised; response discarded unread

The endpoint that hands out the token needs the same authorisation as the data behind it.
```

Illustrative example — synthetic data, not client evidence.

Evidence for F-04 — the loopback service is enumerated, its configuration endpoint returns the bridge token unauthenticated, and the token then authorises a data call. Illustrative reconstruction using a synthetic port and fabricated token; not client evidence.

### Reproduction steps

- 1 Enumerate listening sockets on the device and identify the loopback service, confirming which application owns it.
- 2 Request the service's configuration endpoint with no credential. In the sample the response carries the bridge token in full.
- 3 Replay a data request using the token just obtained. The sample shows 200 — the request is authorised.
- 4 Stop at the status code: the response body was discarded unread, and the interaction was performed from a purpose-built test helper against a test account.

### Impact

Any application the user installs — including one with no permissions of consequence — can read the data this bridge fronts, for as long as the target application is running. The user is given no prompt, because from the platform's point of view nothing requiring consent has happened. On the Windows build the same loopback service is reachable by any process in the user's session, which widens the caller population further. The practical effect is that the application's own data boundary is delegated to a token it publishes.

### Recommendation

- 1 Stop returning the token. The configuration endpoint should expose the version and capability information the web layer needs and never the credential that protects the bridge.
- 2 Require authorisation on every endpoint of the bridge, including the ones that appear to be metadata. An endpoint that can be reached without the token must not disclose anything the token protects.
- 3 Bind the token to the caller rather than to the port: generate it per application launch, pass it to the web layer through the native channel that already exists, and reject any caller that cannot demonstrate it came from that channel.
- 4 Prefer the platform's own native-to-web bridge over a loopback HTTP service where one is available, since it carries an identity the operating system enforces and no port that other applications can reach.

MEDIUM

### F-05 Root and jailbreak detection defeated at a single decision point

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                | AFFECTED COMPONENT                               | MASVS      |
|----------|-----------------------------------------|--------------------------------------------------|------------|
| Medium   | 5.3 AV:P/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N | Android and iOS builds — device integrity checks | RESILIENCE |

**Class:** Reliance on a client-side security control (CWE-602) · insufficient anti-tampering protection · **Reference:** OWASP MASVS-RESILIENCE-1 and MASVS-RESILIENCE-3 · MASTG resiliency testing · OWASP Mobile Top 10 M7 (insufficient binary protections) · **Status:** Open · *Composite element: application with device-integrity gating*

### Description

The application performs several independent integrity checks and then funnels all of them into one method that returns a single boolean. Everything downstream — whether the application starts, whether the offline store unlocks, whether certain features appear — reads that one value.

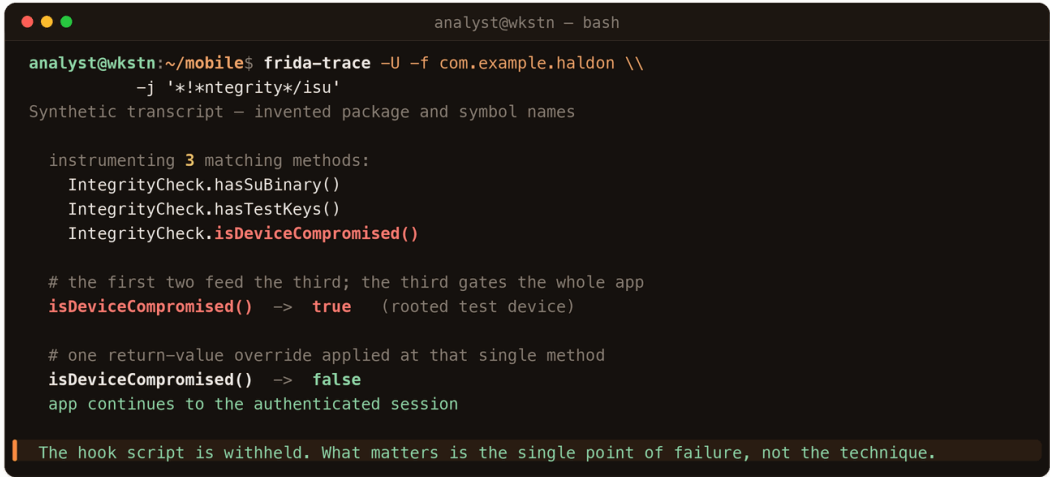
That design makes the mechanism exactly as strong as its weakest point and no stronger, because the checks themselves never have to be defeated. Overriding one return value at runtime disables the entire mechanism at once. We record this as Medium rather than High deliberately. A resilience control of this kind is not a security

boundary and is not expected to stop a determined attacker with the device in hand; it raises cost. The finding is that the cost it raises here is close to zero, not that a bypass was possible at all.

## Evidence

**Root and jailbreak detection defeated at a single decision point**SYNTHETIC EXAMPLE

Every check funnels into one boolean, so one runtime hook disables the whole mechanism.



```
analyst@wkstn: ~/mobile$ frida-trace -U -f com.example.haldon \\  
-j '.*!.*integrity*/isu'  
Synthetic transcript — invented package and symbol names  
  
instrumenting 3 matching methods:  
  IntegrityCheck.hasSuBinary()  
  IntegrityCheck.hasTestKeys()  
  IntegrityCheck.isDeviceCompromised()  
  
# the first two feed the third; the third gates the whole app  
isDeviceCompromised() -> true (rooted test device)  
  
# one return-value override applied at that single method  
isDeviceCompromised() -> false  
app continues to the authenticated session  
  
The hook script is withheld. What matters is the single point of failure, not the technique.
```

i Illustrative example — synthetic data, not client evidence.

Evidence for F-05 — tracing shows three checks collapsing into one decision method, and a single return-value override carries the application into the authenticated session. The instrumentation script is deliberately not published. Illustrative reconstruction using synthetic symbol names; not client evidence.

## Reproduction steps

- 1 Run the application on a device in a state the checks are meant to reject and confirm it refuses to proceed.
- 2 Trace the integrity-related methods to establish the relationship between them. In the sample, two checks feed a third, and the third is what the rest of the application consults.
- 3 Apply a single return-value override at that one method. The application proceeds to the authenticated session.
- 4 Record the structural weakness rather than the technique: the hook script is not included in this report, because the single point of failure is the finding and the method of reaching it is not.

## Impact

On a device under the attacker's control, every protection that depends on this boolean is available to be turned off together, which includes the gate on the offline store examined in F-02. On its own the finding does not expose another user's data; its significance is that it removes the obstacle in front of the findings that do, and it does so in one step. For a customer relying on integrity checks as a compensating control in a risk assessment, that reliance is not supported by what the control actually provides.

## Recommendation

- 1 Distribute the decision. Have the protections that matter consult the underlying signals independently rather than a single shared boolean, so that no one override disables everything.
- 2 Move what can be moved to the server. Where the platform offers an attested device signal that a backend can verify, a server-side decision is worth more than any client-side check, because the attacker does not control the verifier.



- Count the occurrences of the token within the extracted log. Testing used a test account's token throughout; no production session material was handled.

### Impact

A bearer token in a log is valid until it expires, wherever the log travels. The backup path takes it off the device by design and places it somewhere with a different and generally weaker set of controls, where it may persist far longer than the session. Anyone with access to a user's workstation or backup store holds the session without needing the device, the password or the second factor. The combination is what matters: either the logging or the backup setting alone would be a lesser finding.

### Recommendation

- Remove credential material from logging entirely, in every build configuration. Log the fact of a refresh and its outcome, never the token.
- Disable platform backup for the application, or define an explicit backup rule set that excludes logs, caches and any store holding credential material.
- Add a build-time check that fails the release when logging calls reference credential-shaped values, so that a debugging line cannot reach production again.
- Shorten token lifetime and support server-side revocation, so that a token recovered from an old archive has a much smaller window in which it is worth anything.

**MEDIUM**

## F-07 Exported component performs a state change without authentication

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                | AFFECTED COMPONENT                                  | MASVS    |
|----------|-----------------------------------------|-----------------------------------------------------|----------|
| Medium   | 5.9 AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N | Android build — deep-link handler for quick actions | PLATFORM |

**Class:** Missing authentication for critical function (CWE-306) · improper export of application component (CWE-926) · **Reference:** OWASP MASVS-PLATFORM-1 · MASTG testing of app permissions and IPC · OWASP Mobile Top 10 M4 (insufficient input/output validation) · **Status:** Open · *Composite element: application offering deep-linked shortcuts*

### Description

A handler that implements shortcut actions is exported, which is necessary for the shortcuts to work from outside the application. It acts on the currently signed-in user's session, which is also reasonable. What is missing is any check on who invoked it.

Because the component is exported, any application on the device can invoke it with parameters of its choosing, and the action is carried out under the signed-in user's authority with no prompt and no re-authentication. The read paths on the same component are gated correctly, which places this clearly as an oversight on the write path rather than an absent model: the application knows how to check, and does not check here.

## Evidence

### Exported component performs a state change without authentication

SYNTHETIC EXAMPLE

A deep-link handler exported to any application on the device acts on the signed-in user without checking who called it.

```
analyst@wkstn - bash

analyst@wkstn:~/mobile$ aapt dump xmltree haldon-release.apk \\  
    AndroidManifest.xml | grep -A2 'activity-alias'  
Synthetic transcript — invented package and component

    activity .ui.QuickActionActivity  
      android:exported = true  
      scheme = "haldon"    host = "action"

    # invoked from an unprivileged helper app on the same device  
analyst@wkstn:~/mobile$ am start -a android.intent.action.VIEW \\  
    -d 'haldon://action/reassign?record=1042&owner=carol'  
  
Status: ok  
record 1042 reassigned — no prompt, no re-authentication  
  
Performed against a synthetic record in an isolated test tenant.
```

*i* Illustrative example — synthetic data, not client evidence.

Evidence for F-07 — the manifest shows the component exported, and an invocation from an unprivileged helper application changes record ownership without a prompt. Performed against a synthetic record in an isolated test tenant. Illustrative reconstruction; not client evidence.

## Reproduction steps

- 1 Inspect the manifest of the release package and identify components marked as exported, together with the URI scheme and host they respond to.
- 2 Build an unprivileged helper application that requests no permissions of consequence.
- 3 Invoke the exported component from that helper with parameters naming a record and a new owner. In the sample the action completes with no prompt and no re-authentication.
- 4 Verify the state change and then revert it. The record was synthetic and the tenant was an isolated test tenant created for the engagement.

## Impact

Any installed application can take actions as the signed-in user, which is an integrity problem rather than a confidentiality one: records can be reassigned, modified or actioned without the user's knowledge. On a device managed by a customer organisation the attacking application need not be malicious in any obvious way — a compromised utility, or an application with an aggressive advertising library, is sufficient. Because the action is performed through the legitimate application, server-side logs attribute it to the user, which complicates any subsequent investigation.

## Recommendation

- 1 Require an explicit user confirmation in the application for any state-changing action arrived at through an external invocation, so the authority is the user's rather than the caller's.
- 2 Verify the caller where the platform permits it, and treat every parameter arriving from an external invocation as untrusted input to be validated against the session, not as an instruction.
- 3 Export only what has to be exported, and split read-only entry points from state-changing ones so the exported surface carries the smallest possible capability.

- 4 Re-check the authorisation server-side as well. An action that changes ownership should be authorised by the backend on its own terms rather than trusted because it arrived from an authenticated client.

LOW

**F-08 Biometric prompt gates the interface rather than the credential**

| SEVERITY | CVSS v3.1 (ILLUSTRATIVE)                       | AFFECTED COMPONENT                        | MASVS       |
|----------|------------------------------------------------|-------------------------------------------|-------------|
| Low      | <b>3.9</b> AV:P/AC:H/PR:L/UI:N/S:U/C:L/I:N/A:N | Android and iOS builds — biometric unlock | <b>AUTH</b> |

**Class:** Reliance on a client-side security control (CWE-602) · improper authentication (CWE-287) · **Reference:** OWASP MASVS-AUTH-2 and MASVS-AUTH-3 · MASTG testing of local authentication · OWASP Mobile Top 10 M3 (insecure authentication) · **Status:** Open · *Composite element: application offering biometric re-entry*

**Description**

The application offers a biometric unlock on re-entry. The prompt is presented correctly and the platform API is used as intended, but the result is consumed as a boolean that the application then acts on: success reveals the interface, failure hides it. The stored credential is not bound to that decision — the key is created without a user-authentication requirement, so it is released whenever the application asks for it.

The practical consequence is that the prompt is a user-interface state rather than a cryptographic gate. An attacker who can influence the application's own execution — which, given F-05, is not a high bar on a device they control — obtains the credential without satisfying the prompt. We rate this Low because reaching it requires physical possession of an unlocked or compromised device, and because the credential it yields is already protected by the server-side session model.

**Evidence**

**Biometric prompt gates the interface rather than the credential**SYNTHETIC EXAMPLE

The unlock decision is a boolean the application acts on, so the stored credential is released whether or not the prompt succeeds.

```
analyst@wkstn ~$ frida-trace -U -f com.example.haldon -j '.*biometric*/isu'
Synthetic transcript — invented package and symbol names

BiometricGate.onAuthSucceeded() -> unlocks the view
CredentialStore.load()           no auth binding

# the key is requested without a user-authentication requirement
KeyGenParameterSpec.setUserAuthenticationRequired( false )

# so the stored credential loads with the prompt dismissed
CredentialStore.load() -> returned

A biometric gate is only a gate when the key itself is released by it.
```

Illustrative example — synthetic data, not client evidence.

Evidence for F-08 — tracing shows the unlock callback gating the view while the credential store loads independently, with the key created without an authentication requirement. Illustrative reconstruction using synthetic symbol names; not client evidence.

**Reproduction steps**

- 1 Trace the biometric callback and the credential-loading path and establish whether any data dependency exists between them.

- 2 Inspect the key's generation parameters to determine whether release is bound to user authentication. In the sample it is not.
- 3 Load the stored credential with the prompt dismissed rather than satisfied, confirming the two are independent.
- 4 Record the result against a test account. No production credential was involved at any point.

### Impact

Someone holding the device can reach the stored credential without presenting the user's biometric, which defeats the expectation the feature sets. The practical exposure is bounded — physical access is required, and the credential is still subject to server-side session controls — but the feature is relied on by users and described in the product as protecting exactly this case, and it does not.

### Recommendation

- 1 Bind the key to user authentication at creation, so that the platform releases it only after a successful biometric or device-credential check and the gate is enforced by the operating system rather than by application logic.
- 2 Use the authenticated cryptographic object the platform provides, so that the result of the prompt is the ability to perform an operation rather than a boolean the application interprets.
- 3 Re-key on biometric enrolment change, so that adding a new biometric does not silently grant access to credentials stored under the previous set.
- 4 Describe the feature accurately in the product documentation: convenience re-entry and a cryptographic gate are different promises, and customers make risk decisions from the description.

## INFORMATIONAL F-09 Obfuscation applied, but not across the security-relevant code

| SEVERITY      | CVSS v3.1 (ILLUSTRATIVE)                   | AFFECTED COMPONENT                          | MASVS      |
|---------------|--------------------------------------------|---------------------------------------------|------------|
| Informational | n/a not a vulnerability; no score assigned | All platforms — release build configuration | RESILIENCE |

**Class:** Observation · insufficient binary protection coverage · **Reference:** OWASP MASVS-RESILIENCE-2 · MASTG resiliency testing · OWASP Mobile Top 10 M7 (insufficient binary protections) · **Status:** Open · *Composite element: release build with name obfuscation enabled*

### Description

Name obfuscation is enabled and covers the great majority of the application. The remainder, however, is concentrated in precisely the classes an attacker reads first: the credential store, the integrity checks, the bridge token provider and the offline key derivation all retain readable names, almost certainly because they are referenced reflectively or from the web layer and were added to a keep list to stop the build breaking.

This is recorded as informational, not as a vulnerability. Obfuscation is not a security boundary and no finding in this report depends on its absence — the credential in F-01 would have been recoverable from an entirely obfuscated build, because strings are not names. It is included because it tells the team something useful: the effort spent on obfuscation is not landing where it was intended to, and a keep list is quietly documenting the sensitive parts of the application.

## Evidence

### Obfuscation applied, but not across the security-relevant code

SYNTHETIC EXAMPLE

Name obfuscation covers most of the application while the classes handling keys, tokens and integrity keep readable names.

```
analyst@wkstn ~$ jadx -d out haldon-release.apk && cd out
Synthetic transcript — invented package and class names

classes total          4,118
obfuscated names      3,977  96.6%
readable names         141

# the readable remainder is the part worth reading
CredentialStore
IntegrityCheck
BridgeTokenProvider
OfflineDbKeyDeriver

Obfuscation raises the cost of reading code. It is not a boundary, and it did not protect these paths.
```

*i* Illustrative example — synthetic data, not client evidence.

Evidence for F-09 — a count of obfuscated against readable class names in the release build, with the readable remainder listed. Illustrative reconstruction using synthetic class names; not client evidence.

### How this was established

- 1 Decompile the release build and count obfuscated against readable type names to establish overall coverage.
- 2 List the readable remainder and classify it by what the classes do.
- 3 Compare that list against the project's keep rules to confirm the cause.
- 4 Record as an observation. No exploitation is involved and none is implied.

### Impact

No direct impact, and no finding in this report relies on it. The effect is on cost: a reader looking for the security-relevant parts of the application is directed straight to them, which lowers the effort required to find everything else in this report. Treat it as a signal about where the build configuration and the threat model have drifted apart rather than as a defect to be fixed in its own right.

### What to do with this observation

- 1 Narrow the keep rules to the specific members that genuinely need stable names, rather than whole classes, and re-check coverage over the security-relevant packages after each change.
- 2 Do not treat the result as a control. Obfuscation raises the cost of reading code and nothing else; it should never appear in a risk assessment as a mitigation for a secret the artefact contains.
- 3 Fix the underlying issues first. Once F-01 and F-02 are addressed there is far less in the binary worth reading, which is a better outcome than making it harder to read.

**A reminder about this section.** F-01 to F-09 are drawn from the shape of several anonymised engagements and presented as though they came from one. Every transcript, package name, host, identifier, account and score above is synthetic.

## SECTION 7

## Remediation plan

The same findings, arranged as work. Grouped by when it should happen rather than by severity, because some of the most urgent actions are not code changes.

### Wave 1 — now, ahead of any code change

These actions reduce exposure immediately and do not wait on a release cycle.

| Action                                                                                                          | Finding | Owner          |
|-----------------------------------------------------------------------------------------------------------------|---------|----------------|
| Rotate the backend client secret and review API access logs for use from outside the expected client population | F-01    | Platform / API |
| Plan the client release that the rotation forces, and decide how customers on old builds are handled            | F-01    | Product        |
| Document the pinning configuration key for customer administrators, with the security consequence stated        | F-03    | Product / Docs |

### Wave 2 — the next release

Changes that remove the findings rather than reducing their reach, and that need to ship together because several of them touch the same code paths.

| Action                                                                                                     | Finding    | Owner  |
|------------------------------------------------------------------------------------------------------------|------------|--------|
| Remove the shared secret from the artefact; authenticate as the user with a short-lived token              | F-01       | Mobile |
| Move the offline database key into hardware-backed storage and bind its release to user authentication     | F-02, F-08 | Mobile |
| Invert the pinning default and refuse to handle credentials under a user-installed trust anchor            | F-03       | Mobile |
| Stop the bridge configuration endpoint returning the token; require authorisation on every bridge endpoint | F-04       | Mobile |
| Remove credential material from logging and disable platform backup for the application                    | F-06       | Mobile |
| Require in-app confirmation for state changes arriving through exported components                         | F-07       | Mobile |

### Wave 3 — structural, over the following cycles

Work that changes how the next class of finding is prevented rather than fixing this one. None of it is urgent; all of it reduces how often this report needs writing.

| Action                                                                            | Finding | Owner  |
|-----------------------------------------------------------------------------------|---------|--------|
| Distribute the integrity decision so no single override disables every protection | F-05    | Mobile |

| Action                                                                                                             | Finding    | Owner            |
|--------------------------------------------------------------------------------------------------------------------|------------|------------------|
| Add a release gate that scans the built artefact for credential-shaped constants                                   | F-01, F-06 | Build / CI       |
| Narrow the obfuscation keep rules and re-check coverage over security-relevant code                                | F-09       | Mobile / Build   |
| Re-check authorisation server-side for actions a client can initiate, rather than trusting an authenticated client | F-07       | Platform / API   |
| Reduce what is cached offline and expire it sooner                                                                 | F-02       | Product / Mobile |

**Sequencing note.** F-02 and F-08 are listed together deliberately: binding key release to user authentication fixes the biometric gate and strengthens the offline store in the same change. Doing them separately means touching the credential store twice and getting less from the first pass.

## SECTION 8

## Retest and closure

What happens after the fixes, and what the closure letter does and does not say.

### What the retest covers

Every finding in this report is retested against a build you nominate, using the same method that found it. Where a fix changes the approach — a credential moved to the keystore is not verified the way a credential in a bundle was — the retest uses whatever establishes the fix actually holds, and the letter records the method.

A retest is included as standard. It is not a second engagement and does not re-scope: it answers the single question of whether each finding is closed.

### What the closure letter states

The letter names each finding and records it as closed, partially addressed or open, with a sentence on what was verified. It is written to be shown to a customer or an assessor.

It does not state that the application is secure, certify it against any standard, or express an opinion on anything outside the findings listed. We are a testing firm, not an auditor or a certification body, and the letter says so in those terms.

### What a closed finding means for each of these

| Finding | What we verify at retest                                                    |
|---------|-----------------------------------------------------------------------------|
| F-01    | The artefact carries no credential; the old secret no longer authenticates. |
| F-02    | The key cannot be reproduced off the device from device-readable material.  |
| F-03    | A profile that does not set the key enforces pinning.                       |
| F-04    | No bridge endpoint discloses the token; all require it.                     |
| F-05    | No single override disables the protections together.                       |
| F-06    | No credential material in logs; backup excludes logs and credential stores. |
| F-07    | External invocation of a state change requires in-app confirmation.         |
| F-08    | The stored credential cannot be loaded without satisfying the prompt.       |
| F-09    | Coverage over the security-relevant classes, reported as a count.           |

## APPENDIX A

## Builds, accounts and tooling

---

What we asked for before testing started, and why each item mattered.

|                            |                                                                                                                                                                                                                                             |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Release artefacts</b>   | Signed release builds for each platform in scope, matching what is published. Debug builds would have produced findings that do not describe what customers run.                                                                            |
| <b>Test accounts</b>       | Two accounts per role, in an isolated tenant. One account per role cannot demonstrate a cross-account issue; two can.                                                                                                                       |
| <b>Managed profile</b>     | The deployment profile customers are given. F-03 exists because of a default in it, and would not have been found without it.                                                                                                               |
| <b>Backend environment</b> | A non-production environment matching production configuration, so that server-side behaviour could be tested without touching customer data.                                                                                               |
| <b>Named contact</b>       | An engineer who could answer a question about intent. Several hours of reverse engineering are routinely saved by one answer about what a component is for.                                                                                 |
| <b>Tooling</b>             | Standard public mobile tooling for unpacking, decompilation, runtime instrumentation and traffic interception, together with manual analysis. No tool is named as a client's, and no commercial tool's output is reproduced in this sample. |

## APPENDIX B

## Out of scope, and what this test is not

---

Stated explicitly, because the value of a report depends on knowing what it does not cover.

- **Not an audit or certification.** PXLSecurity LTD. is an offensive-security testing firm. Nothing here certifies the application against the MASVS, a MASA assessment or any other scheme, and no such claim may be derived from this document.
- **Not a source code review.** The artefact was treated as an attacker would. A secure code review is a different engagement and finds different things.
- **Not a test of the operating systems.** Platform vulnerabilities, device firmware and the app stores themselves were out of scope.
- **Not a test of the customer's device management.** We used a management profile to reproduce a deployment; we did not test the management platform.
- **No denial of service,** and no technique whose failure mode is a broken device or a corrupted data store.
- **Not a complete test of the backend.** The API was tested as the application reaches it. A full API penetration test covers endpoints and roles the mobile client never touches.
- **Not a privacy or data-protection assessment.** F-06 touches what leaves the device, but a GDPR or data-flow assessment is separate work with a different method.

## APPENDIX C

## References, and what they are

---

Described accurately. Referencing a standard is not conformance with it.

|                                  |                                                                                                                                                                                                                                                                              |
|----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>OWASP MASVS</b>               | The Mobile Application Security Verification Standard: a public list of security requirements for mobile applications, organised into control groups. We structure testing and reporting around it. It is a standard to test against, not a certification we hold or confer. |
| <b>OWASP MASTG</b>               | The Mobile Application Security Testing Guide: the companion manual describing how to test each requirement. It informs method, not severity.                                                                                                                                |
| <b>OWASP Mobile Top 10</b>       | An awareness document listing the most common mobile risk categories. Useful for communicating a finding's shape to a non-specialist; not a test plan.                                                                                                                       |
| <b>OWASP API Security Top 10</b> | Used for the server side of the engagement, where the application's API is in scope.                                                                                                                                                                                         |
| <b>CWE</b>                       | A common vocabulary for weakness types, used in the Class line of each finding so that findings can be grouped and tracked.                                                                                                                                                  |
| <b>CVSS v3.1</b>                 | A scoring system for vulnerability characteristics. Given for each scored finding as an input to prioritisation. Section 4 explains why it does not decide our rating.                                                                                                       |

## APPENDIX D

## How to read this report

Who should read what, and what to do with it.

| If you are...                | Read                         | And do                                                                                 |
|------------------------------|------------------------------|----------------------------------------------------------------------------------------|
| An executive or board member | Section 1                    | Fund wave 1 today. It is three actions and none of them is a code change.              |
| A product owner              | Sections 1, 5 and 7          | Decide how customers on old builds are handled once the credential rotates.            |
| A mobile engineer            | Sections 4 and 6             | Work from the recommendation list; F-02 and F-08 are one change.                       |
| A platform or API engineer   | F-01, F-04 and F-07          | Re-check what the server trusts the client to have decided.                            |
| A security lead              | Sections 2, 3 and 8          | Check the scope assumptions still hold, then book the retest.                          |
| An auditor or assessor       | Sections 2, 4 and Appendix B | Note what this document is and is not; it is evidence of testing, not a certification. |

*A mobile application is the only part of your system you hand to the attacker. Everything in this report follows from that.*

**About this document.** Composite sample, synthetic data, no real client. PXLSecurity LTD., Sofia, Bulgaria · VAT BG203239331 · hello@pxlsec.com · pxlsec.com. Our severity scale, rules of engagement and coordinated disclosure policy are published at pxlsec.com, and the mobile application security service this report illustrates is at pxlsec.com/services/mobile-application-security.