mSEC-AM Audit Summary

OpenMRS Android Client — v3.1.1

Audit Date: 17 Jan 2026
Auditor: Carlos Michael Mejia Granda
Classification: Confidential / Internal Use

Table of Contents

1. App information

Name	OpenMRS Android Client
Developer	Openmrs.org
Category	Mobile Health (mHealth) / Electronic Medical Record (EMR) Client
Supported platforms	Android (SDK 19–33+)
Version	3.1.1
Last update	28/08/2023
Compatible idioms	English
Source	Open source
License	MPL-2.0
Main language	Java + Kotlin + XML + Groovy

2. Actors

Auditor	Carlos Michael Mejia Granda
Requirement Engineering team	Carlos M. Mejía-Granda José L Fernández-Alemán Juan Manuel Carrillo-de-Gea Joaquín Nicolás
Engineering Group (EN)	Carlos M. Mejía-Granda José L Fernández-Alemán Juan Manuel Carrillo-de-Gea Joaquín Nicolás

3. Scope and limitations

This Audit Summary consolidates the compliance determinations recorded in the audit workbook. The results reflect the assessed application version and the workbook-defined scope. Controls not evidenced as implemented in the workbook are reported as non-compliant for summary purposes.

4. Evidence criteria

• Compliant: the workbook provides sufficient evidence that the control is implemented and effective for the assessed scope.
• Non-compliant: the workbook indicates the control is missing, insufficient, or not evidenced.
• Not applicable: the control is recorded as out of scope or not relevant for the assessed context.

5. Audit summary

The audit was carried out using the mSEC-AM (mobile SECurity Audit Method).

Overall, 469 requirements were assessed. 404 were applicable controls and 65 were recorded as not applicable. Of the applicable controls, 28 were compliant and 376 were non-compliant, resulting in an overall compliance rate of 6.93% (applicable controls only).

This report summarizes the dominant weakness patterns evidenced by non-compliant requirements and proposes actionable remediations suitable for mHealth/EMR environments handling sensitive health information.

5.1 Key takeaways (Top findings)

Key takeaways (Top findings)

- Hardcoded credentials / embedded secrets is the most prevalent issue (65 findings), indicating repeated exposure risk from secrets embedded in mobile binaries despite explicit requirements to avoid storing credentials/keys in-app.
- Authorization / RBAC / least privilege gaps remain widespread (50 findings), suggesting that privilege boundaries between user-facing components and sensitive management/security functions are frequently unclear or unenforced.
- Insecure local storage / key management gaps are near the high-prevalence tier (49 findings), aligning with expectations that apps run in untrusted environments and should minimize on-device sensitive data handling.
- Audit logging completeness / retention / alerting gaps are common (47 findings), increasing the likelihood that security-relevant events affecting health data cannot be reliably reconstructed or detected in a timely manner.
- Weak authentication lifecycle / brute-force protections appear frequently (44 findings), reinforcing the need for stronger access-control lifecycle behaviors and protections against abuse of authentication flows.
- Transport security / certificate validation weaknesses (22 findings) and Supply chain governance & outdated components (20 findings) form a mid-prevalence cluster, collectively elevating exposure to tampering, downgrade, and untrusted update/dependency risks.
- Lower-prevalence but material themes include Tampering / reverse engineering protections missing (17 findings), Input validation & injection weaknesses (10 findings), and Privacy notice / consent / governance gaps (6 findings), which can still drive high impact when exploited against health/PHI-bearing workflows.

5.2 Positive controls observed

All statements below are derived exclusively from controls recorded as Compliant in the audit workbook and include supporting signals (flags and/or evidence). Verification traceability is provided in Appendix B.

- The application applications not change the SSL verification method that can allow self-signed certificates. Self-signed certificates or a local Certificate Authority (CA) be used during development to keep with good SSL practices.
- The application remove all code that seems to accept every SSL/TLS certificate, e.g., org.apache.http.conn.ssl.AllowAllHostnameVerifier, SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER, post-development cycle for secure certificate validation.

- The application be free of adware and known malware (via virus-checking against some established library of viruses), and prevents the execution of malicious code, as well as disallowing the uploading of malicious or dangerous-looking scripts or files.
- The application request the READ_SMS permission only when absolutely necessary, minimize its usage, and securely handle SMS data to prevent misuse or exploitation by malicious third parties.
- The application encrypt passwords during transmission to ensure that no passwords are sent in clear text over the network.
- The application prioritize secure and authenticated communication methods, such as Firebase Cloud Messaging (FCM) or encrypted IP-based networking protocols, instead of relying on SMS for app-server interactions.
- The application assume that the network layer is unsecure and susceptible to eavesdropping. Implement end-to-end encryption and other secure communication practices to safeguard sensitive data transmitted over the network.

5.3 Risk scoring approach

Severity and likelihood ratings in this report follow a qualitative rubric grounded in the audit workbook:
• Severity reflects potential impact on confidentiality, integrity, and availability of health information, including regulatory exposure.
• Likelihood is derived from workbook prevalence: the count of non-compliant controls mapped to a weakness pattern as a proxy for exposure.
Likelihood mapping: High (≥50), Medium–High (20–49), Medium (10–19), Low–Medium (<10).

5.4 Risk triage (prioritized)

Weakness pattern	Severity (rubric)	Impact	Likelihood (workbook prevalence)	Workbook basis	Recommended owner	Target timeline
Hardcoded credentials / embedded secrets	High	Embedded secrets can be extracted via static analysis or runtime inspection, enabling unauthorized access to backends, third-party services, or privileged application functions (confidentiality and integrity impact). For applications processing health data, unauthorized access or tampered transactions may create compliance exposure where	High	65 mapped non-compliant control(s) in the workbook.	Mobile Engineering / Security Engineering	0–90 days

		personal/health information protections are required (e.g., PHI/health data confidentiality obligations).				
Authorization / RBAC / least privilege gaps	High	Authorization gaps can allow non-privileged users to invoke privileged functions or access restricted data, directly affecting confidentiality and integrity of user and health information. In health-data contexts, excessive access or cross-tenant exposure can constitute an unauthorized disclosure and weaken auditability of who accessed what (regulatory/compliance risk).	High	50 mapped non-compliant control(s) in the workbook.	Mobile Engineering / Governance	0–90 days
Insecure local storage / key management gaps	High	Storing sensitive data or cryptographic material locally increases exposure to device compromise, malware, backups, or reverse engineering, impacting confidentiality and potentially enabling impersonation (integrity). For health data, local leakage of identifiers, session material, or PHI can be reportable depending on jurisdiction and contractual obligations.	Medium–High	49 mapped non-compliant control(s) in the workbook.	Mobile Engineering	0–90 days

Weak authentication lifecycle / brute-force protections	High	Weak authentication lifecycle controls can enable credential stuffing, brute-force attempts, session fixation/replay, or bypass of protective checks, affecting confidentiality, integrity, and availability. In a health-data setting, account takeover can expose PHI and undermine patient safety and trust, and can trigger breach notification or contractual compliance actions.	Medium–High	44 mapped non-compliant control(s) in the workbook.	Mobile Engineering	0–90 days
Transport security / certificate validation weaknesses	High	Transport-layer weaknesses or inadequate certificate validation can allow man-in-the-middle interception, data exposure, or response tampering (confidentiality and integrity), particularly on hostile networks. For health data, interception or manipulation of transmitted identifiers or clinical/appointment content can create regulatory exposure and patient harm concerns.	Medium–High	22 mapped non-compliant control(s) in the workbook.	Mobile Engineering / DevOps	0–90 days
Supply chain governance & outdated components	High	Outdated or poorly governed dependencies and update channels can introduce known vulnerabilities or malicious	Medium–High	20 mapped non-compliant control(s) in the	DevOps / Security Engineering	0–90 days

		components (confidentiality, integrity, availability). In health-data contexts, supply chain compromise can lead to large-scale unauthorized access or systemic tampering, with attendant compliance and reporting implications.		workbook.		
Input validation & injection weaknesses (XSS/SQLi/command/log injection)	High	Injection weaknesses can enable unauthorized data access, data corruption, command execution, or audit trail manipulation (confidentiality and integrity). For health data, forged logs or manipulated records can undermine clinical integrity, incident response, and compliance requirements for accurate auditability.	Medium	10 mapped non-compliant control(s) in the workbook.	Mobile Engineering	0–90 days
Audit logging completeness / retention / alerting gaps	Medium	Incomplete or inconsistent logging reduces the ability to detect attacks, investigate incidents, and demonstrate accountability (availability of evidence and integrity of records). In health-data environments, insufficient audit trails can impair compliance obligations that require traceability of access to sensitive	Medium–High	47 mapped non-compliant control(s) in the workbook.	Governance / DevOps	0–180 days

		records and timely response to suspected misuse.				
Tampering / reverse engineering protections missing	Medium	Lack of tamper resistance can enable attackers to bypass client-side controls, extract embedded assets, manipulate runtime logic, or alter transaction behavior (integrity and confidentiality). For health data, client manipulation may facilitate unauthorized access or falsification of user actions, weakening assurance and potentially affecting compliance expectations for safeguarding applications used in care workflows.	Medium	17 mapped non-compliant control(s) in the workbook.	Mobile Engineering	0–180 days
Privacy notice / consent / governance gaps	Medium	Privacy and governance gaps can lead to unclear user expectations, inadequate consent handling, or prolonged exposure to unsupported components that may carry security vulnerabilities (confidentiality and integrity). In health-data contexts, insufficient transparency and lifecycle governance can heighten regulatory and contractual risk, especially where notice and user rights obligations apply.	Low–Medium	6 mapped non-compliant control(s) in the workbook.	Governance / Product	0–180 days

6. Main deficiencies

The following deficiencies are synthesized as common weakness patterns based on non-compliant requirements. They are not grouped by category; instead they represent cross-cutting gaps evidenced in the audit workbook.

Hardcoded credentials / embedded secrets (High)

Workbook basis: 65 related non-compliant control(s) mapped to this pattern.

Expected: The mobile application should not contain hardcoded details of external resources, credentials, or API keys, and should avoid storing passwords or secrets in the application binary or relying on generic shared secrets.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Embedded secrets can be extracted via static analysis or runtime inspection, enabling unauthorized access to backends, third-party services, or privileged application functions (confidentiality and integrity impact). For applications processing health data, unauthorized access or tampered transactions may create compliance exposure where personal/health information protections are required (e.g., PHI/health data confidentiality obligations).

Recommended owner: Mobile Engineering / Security Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ICU-001, SECM-CAT-ICU-002, SECM-CAT-ICU-003, SECM-CAT-ICU-004.

- Evidence anchor (from workbook description): The mobile application should not contain hardcoded details of external resources, credentials, or API keys to prevent unauthorized access and secure the app from vulnerabilities.
- Evidence anchor (from workbook description): Applications shall not store passwords or secrets in the application binary, nor inhibit backend integration by using generic shared secrets.

Authorization / RBAC / least privilege gaps (High)

Workbook basis: 50 related non-compliant control(s) mapped to this pattern.

Expected: User-facing components must be separated from data storage/management interfaces and security functions, and privilege modifications for software libraries and application capabilities must be controlled within application ownership boundaries.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Authorization gaps can allow non-privileged users to invoke privileged functions or access restricted data, directly affecting confidentiality and integrity of user and health information. In health-data contexts,

excessive access or cross-tenant exposure can constitute an unauthorized disclosure and weaken auditability of who accessed what (regulatory/compliance risk).

Recommended owner: Mobile Engineering / Governance

Traceability (examples, non-exhaustive): SECM-CAT-ISU-027, SECM-CAT-ISU-032, SECM-CAT-IAA-005, SECM-CAT-IAA-006.

- Evidence anchor (from workbook description): The user-facing components of the application must be logically or physically separated from the data storage and management interfaces and security functions. And security-related functionality is to be isolated; you ca…
- Evidence anchor (from workbook description): Modify and maintain privileges for software libraries must be controlled within application owners.

Insecure local storage / key management gaps (High)

Workbook basis: 49 related non-compliant control(s) mapped to this pattern.

Expected: The app should not store biometric data or private keys on the client device and should retrieve only the minimum information needed to function in an untrusted environment, limiting data leakage or corruption.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Storing sensitive data or cryptographic material locally increases exposure to device compromise, malware, backups, or reverse engineering, impacting confidentiality and potentially enabling impersonation (integrity). For health data, local leakage of identifiers, session material, or PHI can be reportable depending on jurisdiction and contractual obligations.

Recommended owner: Mobile Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ISU-019, SECM-CAT-ISU-036, SECM-CAT-IAA-040, SECM-CAT-IAA-112.

- Evidence anchor (from workbook description): The app should not store any biometric data or private keys on the client device, thus would store and process sensitive data only on the server.
- Evidence anchor (from workbook description): The application should be built to run in untrusted environments and should only retrieve the absolute minimum amount of information needed to function—limiting the potential for data leakage or corruption.

Weak authentication lifecycle / brute-force protections (High)

Workbook basis: 44 related non-compliant control(s) mapped to this pattern.

Expected: The application must maintain strong access control and ensure non-privileged users cannot execute privileged functions or disable/circumvent security safeguards, including during authentication and session lifecycle events.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Weak authentication lifecycle controls can enable credential stuffing, brute-force attempts, session fixation/replay, or bypass of protective checks, affecting confidentiality, integrity, and availability. In a health-data setting, account takeover can expose PHI and undermine patient safety and trust, and can trigger breach notification or contractual compliance actions.

Recommended owner: Mobile Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ISU-011, SECM-CAT-IAA-003, SECM-CAT-IAA-008, SECM-CAT-IAA-011.

- Evidence anchor (from workbook description): The application must continue with strong access control, and local integrity checks applied to determine if the code has been modified without permission.
- Evidence anchor (from workbook description): The application must prevent non-privileged users from executing privileged functions, including disabling, circumventing, or altering implemented security safeguards or countermeasures.

Transport security / certificate validation weaknesses (High)

Workbook basis: 22 related non-compliant control(s) mapped to this pattern.

Expected: The application should ensure that mobile code and externally obtained resources are trusted (e.g., signed/verified) and should avoid accepting unmanaged or unsigned code, aligning transport and update trust decisions with verifiable authenticity.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Transport-layer weaknesses or inadequate certificate validation can allow man-in-the-middle interception, data exposure, or response tampering (confidentiality and integrity), particularly on hostile networks. For health data, interception or manipulation of transmitted identifiers or clinical/appointment content can create regulatory exposure and patient harm concerns.

Recommended owner: Mobile Engineering / DevOps

Traceability (examples, non-exhaustive): SECM-CAT-ISU-003, SECM-CAT-ISU-004, SECM-CAT-ISU-007, SECM-CAT-ISU-010.

- Evidence anchor (from workbook description): The application shall not contain any unmanaged and unsigned mobile code.
- Evidence anchor (from workbook description): The application shall not contain any mobile code that is uncategorized or lacks a digital signature.

Supply chain governance & outdated components (High)

Workbook basis: 20 related non-compliant control(s) mapped to this pattern.

Expected: Resources outside standard app-store mechanisms should be signed and verified before updates are accepted, and the application should undergo periodic security assessments (including code review and cryptographic vulnerability assessment) with remediation of identified issues.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Outdated or poorly governed dependencies and update channels can introduce known vulnerabilities or malicious components (confidentiality, integrity, availability). In health-data contexts, supply chain compromise can lead to large-scale unauthorized access or systemic tampering, with attendant compliance and reporting implications.

Recommended owner: DevOps / Security Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ISU-008, SECM-CAT-ISU-009, SECM-CAT-ISU-013, SECM-CAT-ISU-022.

- Evidence anchor (from workbook description): Ensure that any resources used outside of standard app-store mechanisms are signed and also verify the signature before accepting updates.
- Evidence anchor (from workbook description): The application shall be subject to periodic security assessments including code reviews, penetration tests, and cryptographic vulnerabilities assessments according to the results of which security holes must be remedied…

Input validation & injection weaknesses (XSS/SQLi/command/log injection) (High)

Workbook basis: 10 related non-compliant control(s) mapped to this pattern.

Expected: The application must implement comprehensive input validation to prevent insecure deserialization and injection, including protections against log injection so that backend event histories cannot be corrupted or forged.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Injection weaknesses can enable unauthorized data access, data corruption, command execution, or audit trail manipulation (confidentiality and integrity). For health data, forged logs or manipulated records can undermine clinical integrity, incident response, and compliance requirements for accurate auditability.

Recommended owner: Mobile Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ISU-037, SECM-CAT-IOV-001, SECM-CAT-IOV-004, SECM-CAT-IOV-011.

- Evidence anchor (from workbook description): The application needs to implement protections against log injection by clients so that event histories in the backend system cannot be corrupted or forged.
- Evidence anchor (from workbook description): The application must implement comprehensive input validation mechanisms to safeguard against vulnerabilities and insecure deserialization. All input from external sources, including files, network communications, and in…

Audit logging completeness / retention / alerting gaps (Medium)

Workbook basis: 47 related non-compliant control(s) mapped to this pattern.

Expected: Security-relevant behaviors (including session termination upon account deletion and careful handling of log-on validation/error messaging) should be consistently logged in a way that supports detection and investigation without leaking sensitive authentication details.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Incomplete or inconsistent logging reduces the ability to detect attacks, investigate incidents, and demonstrate accountability (availability of evidence and integrity of records). In health-data environments, insufficient audit trails can impair compliance obligations that require traceability of access to sensitive records and timely response to suspected misuse.

Recommended owner: Governance / DevOps

Traceability (examples, non-exhaustive): SECM-CAT-IAA-004, SECM-CAT-IAA-010, SECM-CAT-IAA-014, SECM-CAT-IAA-019.

- Evidence anchor (from workbook description): The application must terminate all existing user sessions immediately upon account deletion to prevent unauthorized access to application resources.
- Evidence anchor (from workbook description): The application must validate log-on information only after all input data has been completed and must not indicate which specific part of the input is correct or incorrect in case of an error.

Tampering / reverse engineering protections missing (Medium)

Workbook basis: 17 related non-compliant control(s) mapped to this pattern.

Expected: The application must incorporate anti-tampering techniques to detect unauthorized code changes and resist runtime inspection or binary meddling.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Lack of tamper resistance can enable attackers to bypass client-side controls, extract embedded assets, manipulate runtime logic, or alter transaction behavior (integrity and confidentiality). For health data, client manipulation may facilitate unauthorized access or falsification of user actions, weakening assurance and potentially affecting compliance expectations for safeguarding applications used in care workflows.

Recommended owner: Mobile Engineering

Traceability (examples, non-exhaustive): SECM-CAT-ISU-006, SECM-CAT-ISU-012, SECM-CAT-ISU-017, SECM-CAT-ISU-026.

- Evidence anchor (from workbook description): The application must incorporate look designs to distinguish unauthorized code changes and meddling with the binary.

- Evidence anchor (from workbook description): The anti-tampering techniques to prevent code alterations and runtime inspections of your application must be implemented by the application.

Privacy notice / consent / governance gaps (Medium)

Workbook basis: 6 related non-compliant control(s) mapped to this pattern.

Expected: The application should implement governance procedures to handle deprecation and unsupported components, including notifying users when the application is deprecated and managing lifecycle updates to remove out-of-date components.

Observed: The audit workbook indicates the related controls are missing, insufficient, or not evidenced for the assessed scope.

Impact: Privacy and governance gaps can lead to unclear user expectations, inadequate consent handling, or prolonged exposure to unsupported components that may carry security vulnerabilities (confidentiality and integrity). In health-data contexts, insufficient transparency and lifecycle governance can heighten regulatory and contractual risk, especially where notice and user rights obligations apply.

Recommended owner: Governance / Product

Traceability (examples, non-exhaustive): SECM-CAT-ISU-033, SECM-CAT-ISU-040, SECM-CAT-IAA-013, SECM-CAT-IPC-004.

- Evidence anchor (from workbook description): The application shall remove out-of-date components after update installation, disable the use of unsupported components, and be decommissioned when there is no maintenance or vendor support available.
- Evidence anchor (from workbook description): The application shall implement procedures to notify the user upon the application being deprecated.

7. Recommendations

Recommendations are organized by the same weakness patterns presented in the Main deficiencies section. They target remediation of workbook-evidenced gaps and may include strengthening controls to improve security posture.

Hardcoded credentials / embedded secrets

- Eliminate secrets from mobile binaries (including API keys, credentials, endpoints with embedded tokens) and treat any client-side value as publicly discoverable.
- Use server-side issued, short-lived credentials/tokens for app-to-service calls, and avoid shared secrets that are common across installs or users.
- Implement secure secret provisioning patterns that bind credentials to authenticated user/device context rather than shipping static values in the app package.
- Apply build-time and CI checks to detect accidental inclusion of secrets (e.g., key patterns, configuration files) before release artifacts are produced.
- Rotate and revoke exposed secrets promptly when discovered, and ensure the design supports rotation without client updates.
- Constrain backend permissions for any client-present identifier to least privilege, so compromise does not grant broad access.
- Harden client against trivial extraction (e.g., minimize sensitive configuration footprint), while assuming reverse engineering remains possible.
- Require authenticated, authorized server-side enforcement for any action that would otherwise be “protected” only by possession of a client-side secret.

Authorization / RBAC / least privilege gaps

- Define and document roles, permissions, and protected resources, including administrative and security-sensitive functions, with explicit deny-by-default behavior.
- Enforce authorization on the server for every sensitive operation; do not rely on client UI state, hidden screens, or local flags to gate access.
- Implement least-privilege permission models for internal libraries/modules so that components only receive the minimal capabilities required.
- Separate user-facing flows from management/security interfaces via distinct endpoints, distinct authorization policies, and clear trust boundaries.
- Add centralized authorization checks (policy-based or middleware-based) to reduce inconsistent per-endpoint enforcement.
- Validate object-level authorization (e.g., record ownership/tenant boundaries) in addition to role checks to prevent IDOR-style access.
- Introduce step-up authentication for high-risk actions (e.g., modifying security settings, exporting sensitive data), such as MFA or biometric re-auth where appropriate.
- Create negative tests for privilege escalation paths (e.g., role changes, direct API invocation) and keep them in regression suites.

Insecure local storage / key management gaps

- Minimize on-device storage of sensitive information; treat the device as an untrusted environment and store only what is strictly necessary.

- Avoid persisting private keys, biometric templates, or equivalent high-value secrets on the client; prefer server-side processing for sensitive operations.
- When local storage is required, use platform-provided secure storage primitives with appropriate access controls and avoid storing raw sensitive payloads.
- Apply strict data classification to determine what may be cached locally, and prohibit storing PHI/health data unless explicitly justified.
- Ensure cryptographic key management follows separation of duties (generation, storage, usage) and avoids exposing keys to application-layer logs or analytics.
- Implement secure wipe/expiration for cached data upon logout, account deletion, or session invalidation to reduce residual exposure.
- Prevent inadvertent leakage through backups, screenshots, crash reports, and debug artifacts by explicitly controlling what can be exported from the device.
- Validate that any “offline mode” design does not bypass server-side controls or expand the local sensitive footprint beyond necessity.

Weak authentication lifecycle / brute-force protections

- Add robust protections against automated credential attacks, including rate limiting, progressive delays, and abuse detection at the authentication interface.
- Use uniform error messages and response behaviors to avoid revealing whether usernames, emails, or specific input fields are valid.
- Harden session management (secure issuance, renewal, revocation) and ensure server-side invalidation on logout, password change, and other sensitive events.
- Require re-authentication (step-up) for high-risk actions, especially those affecting security posture or accessing/exporting sensitive health data; MFA/biometric re-auth may be appropriate depending on the use case.
- Prevent bypass of security checks by enforcing all critical validation on the server (e.g., account status, risk checks), not only in the client.
- Implement lockout or risk-based throttling strategies that balance protection and usability, with attention to denial-of-service abuse.
- Validate integrity of authentication-related client flows (e.g., deep links, redirects) so that privileged functions cannot be invoked without proper authorization context.
- Add regression tests for common auth abuses (brute force, enumeration, replay, session reuse across devices) to prevent reintroduction.

Transport security / certificate validation weaknesses

- Enforce modern TLS configurations for all network communications and prohibit cleartext transport for any sensitive or authenticated traffic.
- Implement strict certificate validation and hostname verification; avoid disabling verification or accepting user-installed/untrusted CAs without a justified threat model.
- Consider certificate/public key pinning for high-risk endpoints where operationally feasible, with a resilient rotation strategy to avoid outages.
- Ensure secure handling of redirects and mixed-content scenarios so that authenticated sessions cannot be downgraded to weaker channels.

- Validate the integrity and authenticity of any remotely obtained code/resources used by the app (e.g., verify signatures) before acceptance.
- Harden error handling so transport failures do not fall back to insecure modes or silently accept unverifiable connections.
- Segment sensitive API calls (authentication, PHI access) to the strongest transport requirements and apply additional request integrity protections where appropriate.
- Test transport behavior under adversarial conditions (proxy interception, invalid certs, captive portals) to confirm secure failure modes.

Supply chain governance & outdated components

- Establish dependency governance that inventories third-party components and enforces approval criteria for introduction and updates.
- Require trusted provenance for external resources and updates, including cryptographic signing and signature verification prior to use.
- Implement a vulnerability monitoring and patch intake process for dependencies (including mobile libraries and backend-adjacent SDKs).
- Pin and verify dependency versions in builds to reduce unreviewed drift, and prevent use of deprecated/unsupported components.
- Perform security-focused reviews of high-risk dependencies (networking, crypto, authentication, analytics) with attention to transitive dependencies.
- Integrate periodic security assessments (code review, penetration testing, cryptographic review) into release governance, with tracked remediation of findings.
- Restrict build and distribution pipelines to trusted sources to reduce the risk of introducing tampered artifacts.
- Define criteria and procedures for decommissioning components when maintenance or vendor support ceases.

Input validation & injection weaknesses (XSS/SQLi/command/log injection)

- Treat all external input as untrusted (network data, files, deep links, inter-process messages) and validate against strict allowlists.
- Apply context-aware output encoding/escaping and parameterized interfaces (e.g., for database queries, OS commands, and logging) to prevent injection.
- Prevent insecure deserialization by avoiding unsafe object reconstruction patterns and enforcing type constraints and schema validation.
- Implement structured logging with proper field encoding to mitigate log injection and preserve the integrity of security event records.
- Normalize and validate identifiers (IDs, filenames, URLs) to prevent path traversal and command or query construction abuses.
- Ensure server-side validation mirrors or exceeds any client-side checks; client-side validation should be treated as usability-only.
- Add negative test suites for representative payloads (XSS/SQLi/command/log injection) in CI to reduce regressions.
- Review error handling to avoid leaking sensitive information (stack traces, query details) that can facilitate exploitation.

Audit logging completeness / retention / alerting gaps

- Define a logging specification for security-relevant events (authentication attempts, authorization failures, privilege changes, sensitive data access, session lifecycle events).
- Ensure log records include sufficient context to support investigations (who/what/when/where) without recording sensitive secrets or full health payloads.
- Protect log integrity by using structured logging and controls that reduce the risk of forgery or corruption (including log injection defenses).
- Implement retention and access controls for logs so audit evidence remains available and protected from unauthorized access or tampering.
- Establish alerting use cases for high-risk signals (e.g., repeated auth failures, anomalous access patterns, privilege escalation attempts).
- Validate that account deletion and other lifecycle actions trigger complete session invalidation and corresponding audit events.
- Align authentication error handling with non-enumeration guidance (avoid disclosing which field was correct) while still producing actionable internal logs.
- Periodically review log coverage against threat models and incident learnings to address gaps and reduce noise.

Tampering / reverse engineering protections missing

- Implement integrity checks that can detect unauthorized modification of the binary and critical runtime state, with secure failure behavior.
- Harden against common reverse engineering techniques (debugging, hooking, instrumentation) recognizing that client-side protections are deterrents, not absolute barriers.
- Reduce reliance on client-enforced security decisions; move authorization and policy enforcement to server-side controls wherever feasible.
- Protect sensitive logic and assets by minimizing their presence on the client and avoiding embedding privileged decision rules in app code.
- Apply code obfuscation and anti-debug measures appropriate to the risk level, focusing on protecting authentication, authorization, and sensitive workflows.
- Detect compromised or rooted/jailbroken environments where feasible and define how risk is handled (e.g., step-up authentication, restricted functionality).
- Use attestation-style signals (where appropriate) to inform server-side risk decisions, rather than using them as sole gates.
- Continuously test tampering scenarios (repackaging, method hooking, runtime patching) to verify protections remain effective across releases.

Privacy notice / consent / governance gaps

- Maintain clear, accessible in-app privacy notices that accurately describe data collection, use, sharing, and retention for any sensitive/health-related processing.
- Implement consent and preference management appropriate to the data sensitivity and jurisdictional expectations, including the ability to withdraw consent where applicable.
- Define application lifecycle governance for deprecation, including user notification mechanisms and safe end-of-life behavior.

- Remove or disable unsupported/out-of-date components during updates to reduce ongoing exposure to known vulnerabilities.
- Ensure product and governance owners review privacy-impacting changes (new data fields, new third parties, new analytics) before release.
- Avoid collecting or retaining more data than necessary for the app's function, especially for health identifiers and related metadata.
- Provide user-facing mechanisms for account/data lifecycle actions (e.g., deletion requests) that align with backend enforcement and auditability.
- Validate that third-party integrations are reflected in governance documentation and user disclosures, consistent with least-privilege data sharing.

8. Visual Analytics

Figures below summarize workbook-derived outcomes and distributions. All figures: source: audit workbook.

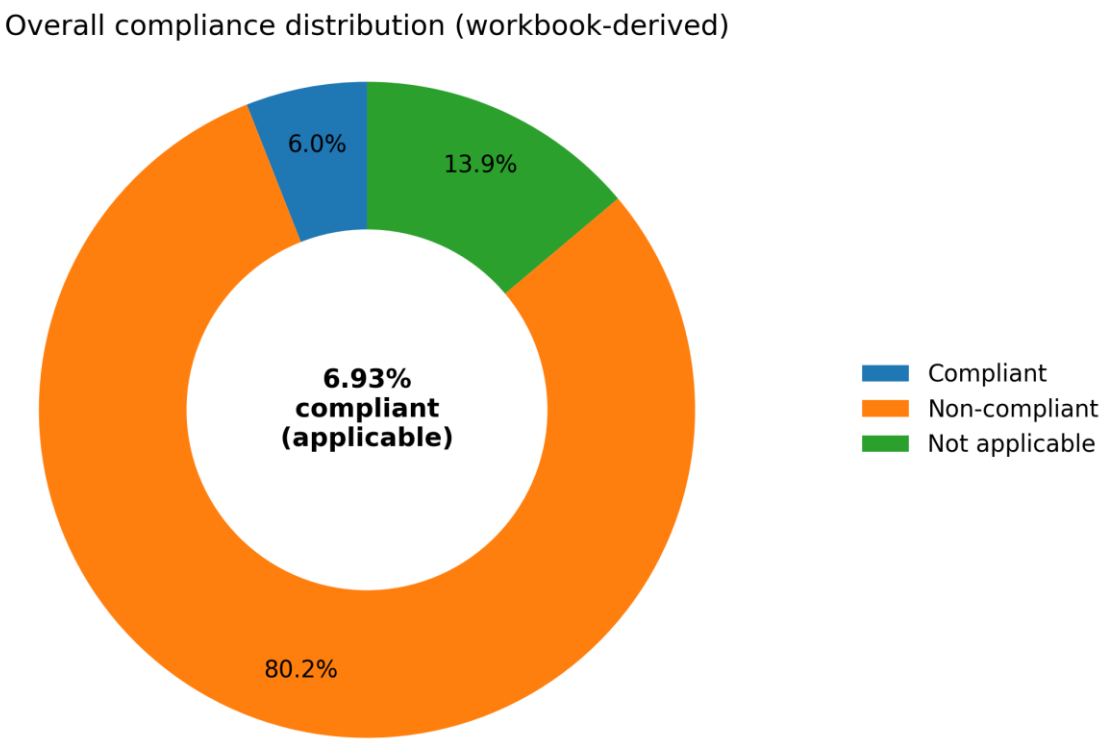
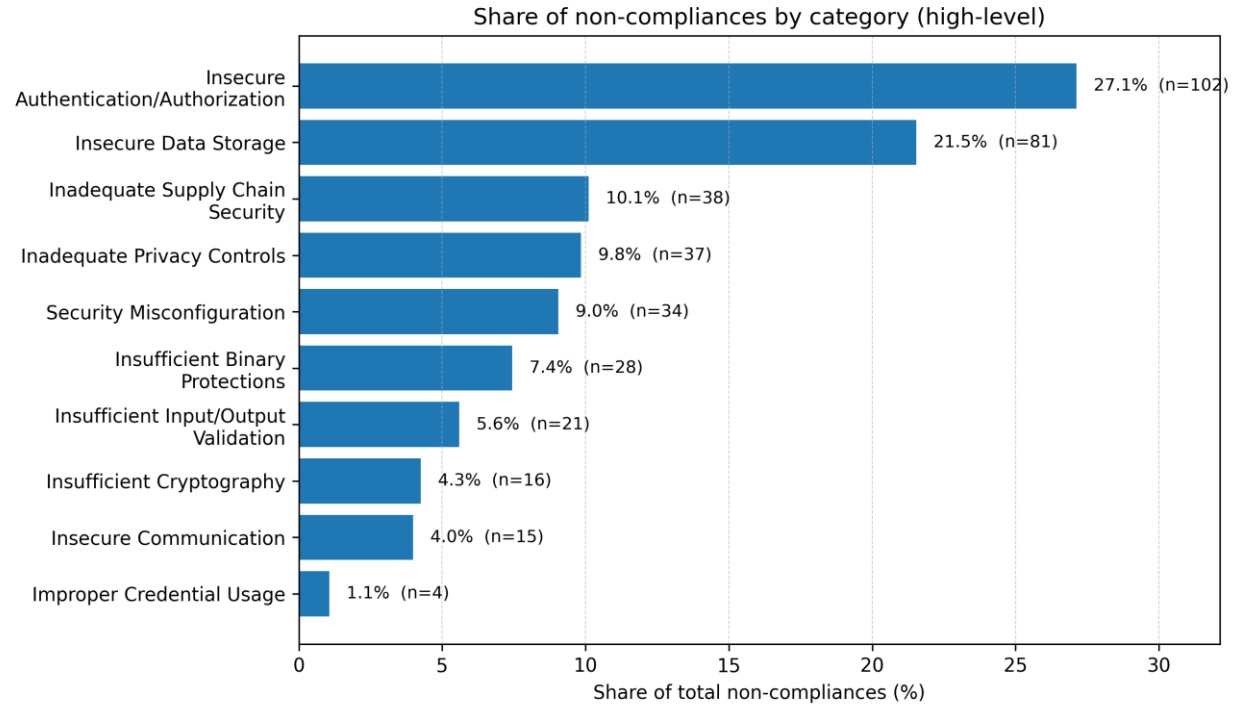
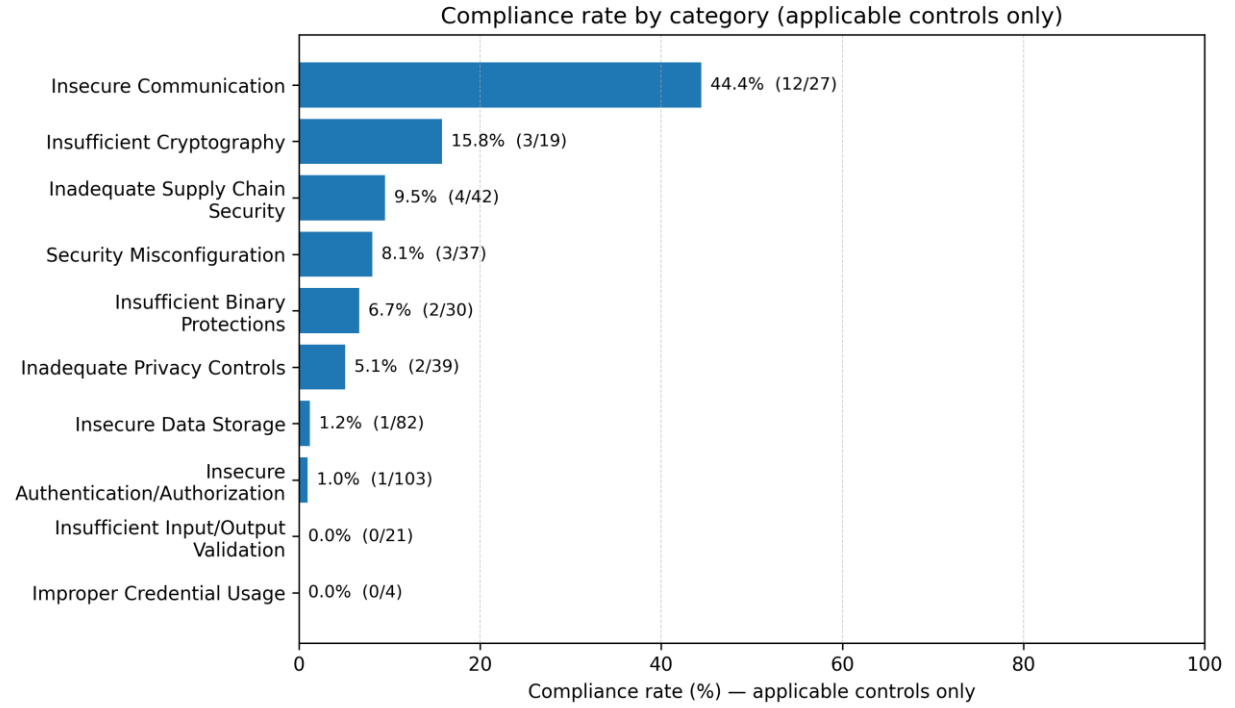


Figure 1. Overall compliance distribution (donut chart; source: audit workbook).



Source: audit workbook. Total non-compliances: 376.

Figure 2. Share of non-compliances by category (legible horizontal bars; source: audit workbook).



Note: (c/a) indicates Compliant / Applicable controls per category. Source: audit workbook.

Figure 3. Compliance rate by category (applicable controls only; source: audit workbook).

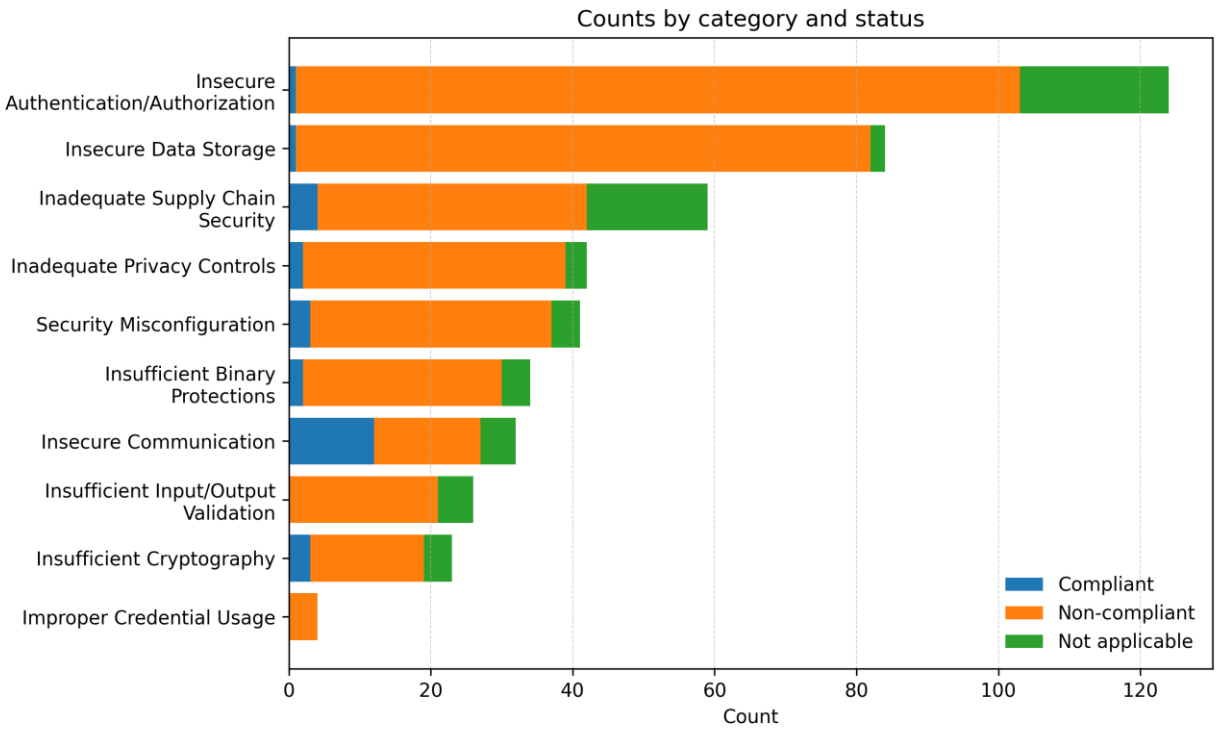


Figure 4. Counts by category and status (horizontal stacked bars; source: audit workbook).

9. Management Action Plan (MAP)

Severity and likelihood nomenclature follow the rubric in Section 5.3. Likelihood is supported by workbook prevalence counts recorded in the Workbook basis column.

Finding / weakness pattern	Severity	Likelihood	Workbook basis	Owner	Management action	Target window	Target date	Acceptance criteria / KPI
Hardcoded credentials / embedded secrets	High	High	65 mapped non-compliant control(s) in workbook.	Mobile Engineering / Security Engineering	Eliminate secrets from mobile binaries (including API keys, credentials, endpoints with embedded tokens) and treat any client-side value as publicly discoverable. Use server-side issued, short-lived credentials/tokens for app-to-service calls, and avoid shared secrets that are common across installs or users. Implement secure secret provisioning patterns that bind credentials to authenticated user/device context rather than shipping static values in the app package.	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.
Authorization / RBAC / least privilege gaps	High	High	50 mapped non-compliant control(s) in workbook.	Mobile Engineering / Governance	Define and document roles, permissions, and protected resources, including administrative and security-sensitive functions, with	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.

					explicit deny-by-default behavior. Enforce authorization on the server for every sensitive operation; do not rely on client UI state, hidden screens, or local flags to gate access. Implement least-privilege permission models for internal libraries/modules so that components only receive the minimal capabilities required.			
Insecure local storage / key management gaps	High	Medium–High	49 mapped non-compliant control(s) in workbook.	Mobile Engineering	Minimize on-device storage of sensitive information; treat the device as an untrusted environment and store only what is strictly necessary. Avoid persisting private keys, biometric templates, or equivalent high-value secrets on the client; prefer server-side processing for sensitive operations. When local storage is required, use platform-provided secure storage primitives with appropriate	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.

					access controls and avoid storing raw sensitive payloads.			
Weak authentication lifecycle / brute-force protections	High	Medium–High	44 mapped non-compliant control(s) in workbook.	Mobile Engineering	Add robust protections against automated credential attacks, including rate limiting, progressive delays, and abuse detection at the authentication interface. Use uniform error messages and response behaviors to avoid revealing whether usernames, emails, or specific input fields are valid. Harden session management (secure issuance, renewal, revocation) and ensure server-side invalidation on logout, password change, and other sensitive events.	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.
Transport security / certificate validation weaknesses	High	Medium–High	22 mapped non-compliant control(s) in workbook.	Mobile Engineering / DevOps	Enforce modern TLS configurations for all network communications and prohibit cleartext transport for any sensitive or authenticated traffic. Implement strict certificate	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.

					validation and hostname verification; avoid disabling verification or accepting user-installed/untrusted CAs without a justified threat model. Consider certificate/public key pinning for high-risk endpoints where operationally feasible, with a resilient rotation strategy to avoid outages.			
Supply chain governance & outdated components	High	Medium–High	20 mapped non-compliant control(s) in workbook.	DevOps / Security Engineering	Establish dependency governance that inventories third-party components and enforces approval criteria for introduction and updates. Require trusted provenance for external resources and updates, including cryptographic signing and signature verification prior to use. Implement a vulnerability monitoring and patch intake process for dependencies (including mobile libraries and backend-adjacent SDKs).	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.

Input validation & injection weaknesses (XSS/SQLi/command/log injection)	High	Medium	10 mapped non-compliant control(s) in workbook.	Mobile Engineering	Treat all external input as untrusted (network data, files, deep links, inter-process messages) and validate against strict allowlists. Apply context-aware output encoding/escaping and parameterized interfaces (e.g., for database queries, OS commands, and logging) to prevent injection. Prevent insecure deserialization by avoiding unsafe object reconstruction patterns and enforcing type constraints and schema validation.	0–90 days	17 Apr 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.
Audit logging completeness / retention / alerting gaps	Medium	Medium–High	47 mapped non-compliant control(s) in workbook.	Governance / DevOps	Define a logging specification for security-relevant events (authentication attempts, authorization failures, privilege changes, sensitive data access, session lifecycle events). Ensure log records include sufficient context to support investigations (who/what/when/where) without recording	0–180 days	16 Jul 2026	Evidence recorded (tests/configs/release refs); mapped controls can be re-tested and re-scored as compliant.

					sensitive secrets or full health payloads. Protect log integrity by using structured logging and controls that reduce the risk of forgery or corruption (including log injection defenses).			
Tampering / reverse engineering protections missing	Medium	Medium	17 mapped non-compliant control(s) in workbook.	Mobile Engineering	Implement integrity checks that can detect unauthorized modification of the binary and critical runtime state, with secure failure behavior. Harden against common reverse engineering techniques (debugging, hooking, instrumentation) recognizing that client-side protections are deterrents, not absolute barriers. Reduce reliance on client-enforced security decisions; move authorization and policy enforcement to server-side controls wherever feasible.	0–180 days	16 Jul 2026	Evidence recorded (tests/configs/ release refs); mapped controls can be re-tested and re-scored as compliant.
Privacy notice / consent / governance gaps	Medium	Low–Medium	6 mapped non-compliant	Governance / Product	Maintain clear, accessible in-app privacy notices that accurately describe data	0–180 days	16 Jul 2026	Evidence recorded (tests/configs/ release refs); mapped

			control (s) in workb ook.		collection, use, sharing, and retention for any sensitive/health-related processing. Implement consent and preference management appropriate to the data sensitivity and jurisdictional expectations, including the ability to withdraw consent where applicable. Define application lifecycle governance for deprecation, including user notification mechanisms and safe end-of-life behavior.			controls can be re-tested and re-scored as compliant.

Appendix A — Traceability index (non-exhaustive)

For complete traceability and evidence, refer to the audit workbook.

- Hardcoded credentials / embedded secrets: SECM-CAT-ICU-001, SECM-CAT-ICU-002, SECM-CAT-ICU-003, SECM-CAT-ICU-004, SECM-CAT-ISU-001
- Authorization / RBAC / least privilege gaps: SECM-CAT-ISU-027, SECM-CAT-ISU-032, SECM-CAT-IAA-005, SECM-CAT-IAA-006, SECM-CAT-IAA-025
- Insecure local storage / key management gaps: SECM-CAT-ISU-019, SECM-CAT-ISU-036, SECM-CAT-IAA-040, SECM-CAT-IAA-112, SECM-CAT-IOV-020
- Weak authentication lifecycle / brute-force protections: SECM-CAT-ISU-011, SECM-CAT-IAA-003, SECM-CAT-IAA-008, SECM-CAT-IAA-011, SECM-CAT-IAA-017
- Transport security / certificate validation weaknesses: SECM-CAT-ISU-003, SECM-CAT-ISU-004, SECM-CAT-ISU-007, SECM-CAT-ISU-010, SECM-CAT-ISU-018
- Supply chain governance & outdated components: SECM-CAT-ISU-008, SECM-CAT-ISU-009, SECM-CAT-ISU-013, SECM-CAT-ISU-022, SECM-CAT-ISU-023
- Input validation & injection weaknesses (XSS/SQLi/command/log injection): SECM-CAT-ISU-037, SECM-CAT-IOV-001, SECM-CAT-IOV-004, SECM-CAT-IOV-011, SECM-CAT-IOV-015
- Audit logging completeness / retention / alerting gaps: SECM-CAT-IAA-004, SECM-CAT-IAA-010, SECM-CAT-IAA-014, SECM-CAT-IAA-019, SECM-CAT-IAA-021
- Tampering / reverse engineering protections missing: SECM-CAT-ISU-006, SECM-CAT-ISU-012, SECM-CAT-ISU-017, SECM-CAT-ISU-026, SECM-CAT-ISU-039
- Privacy notice / consent / governance gaps: SECM-CAT-ISU-033, SECM-CAT-ISU-040, SECM-CAT-IAA-013, SECM-CAT-IPC-004, SECM-CAT-IPC-010

Appendix B — Positive controls verification (workbook traceability)

This appendix verifies each Positive controls observed statement by providing the originating PUID, flags used, and an evidence excerpt when available.

Positive control statement (as reported)	Workbook PUID	Flags used	Evidence / justification (excerpt)
The application applications not change the SSL verification method that can allow self-signed certificates. Self-signed certificates or a local Certificate Authority (CA) be used during development to keep with good SSL practices.	SECM-CAT-ISU-002	has_android_ssl_pinning_present, has_android_ssl_pinning_detected, has_manifest_allow_clear_text_traffic_true	Result: yes for requirement SECM-CAT-ISU-002. Based on the fingerprint and the flags mapped to this requirement, the application is compliant because the evaluated signals support …
The application remove all code that seems to accept every SSL/TLS certificate, e.g., org.apache.http.conn.ssl.AllowAllHostnameVerifier, SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER, post-development cycle for secure certificate validation.	SECM-CAT-ISU-005	has_android_ssl_pinning_present, has_android_ssl_pinning_detected, has_manifest_allow_clear_text_traffic_true	Result: yes for requirement SECM-CAT-ISU-005. Based on the fingerprint and the flags mapped to this requirement, the applicatio

			n is compliant because the evaluated signals support …
The application be free of adware and known malware (via virus-checking against some established library of viruses), and prevents the execution of malicious code, as well as disallowing the uploading of malicious or dangerous-looking scripts or files.	SECM-CAT-ISU-035	has_malware_detections	Result: yes for requirement SECM-CAT-ISU-035. Based on the fingerprint and the flags mapped to this requirement, the application is compliant because the evaluated signals support …
The application request the READ_SMS permission only when absolutely necessary, minimize its usage, and securely handle SMS data to prevent misuse or exploitation by malicious third parties.	SECM-CAT-ISU-056	has_android_extra_risky_permissions_present, has_supports_runtime_permission_management	Result: yes for requirement SECM-CAT-ISU-056. Based on the fingerprint and the flags mapped to this requirement, the application is compliant

			because the evaluated signals support …
The application encrypt passwords during transmission to ensure that no passwords are sent in clear text over the network.	SECM-CAT-IAA-009	has_manifest_allow_clear_text_traffic_true, has_android_ssl_pinning_present, has_ssl_pinning_findings_severity_good	Result: yes for requirement SECM-CAT-IAA-009. Based on the fingerprint and the flags mapped to this requirement, the application is compliant because the evaluated signals support …
The application prioritize secure and authenticated communication methods, such as Firebase Cloud Messaging (FCM) or encrypted IP-based networking protocols, instead of relying on SMS for app-server interactions.	SECM-CAT-ICO-004	has_tls_ssl_pinning_implemented, has_ssl_cert_pinning_implemented	Result: yes for requirement SECM-CAT-ICO-004. Based on the fingerprint and the flags mapped to this requirement, the application is compliant because the

			evaluated signals support …
The application assume that the network layer is unsecure and susceptible to eavesdropping. Implement end-to-end encryption and other secure communication practices to safeguard sensitive data transmitted over the network.	SECM-CAT-ICO-006	has_tls_ssl_pinning_implemented, has_android_ssl_pinning_detected, has_insecure_http_based_webview_communication	Result: yes for requirement SECM-CAT-ICO-006. Based on the fingerprint and the flags mapped to this requirement, the application is compliant because the evaluated signals support …