JONE / CLIENT SECURITY RESEARCH

Defensive Reverse Engineering of WeChat Clients

Android storage and messaging; Windows transport and key wrapping

SAMPLE EDITION

One complete experiment + selected chapters

JONE Security Research

Independent research | English edition | September 2026

Publication Note

Defensive Reverse Engineering of WeChat Clients

Android Storage and Messaging, Windows Transport, and Validation Methods

Author: JONE Security Research. English public redacted edition V2.0. Research manuscript dated September 2, 2026; publication package prepared September 3, 2026.

This is a SAMPLE EDITION. It contains selected opening chapters, the laboratory baseline, and one complete experiment (WX-L02). Excerpts retain their original chapter and experiment numbering. The full English edition includes 15 experiment cards.

Independent research

This is an independent publication. It is not affiliated with, sponsored by, endorsed by, or an official manual of the companies or products discussed. Product names identify the subjects of analysis; their trademarks remain with their respective owners.

Evidence and scope

The findings belong to the fixed samples described in the text. Observation, inference, negative evidence, and unresolved questions retain their stated grades. Reproducing an experiment requires an authorized sample and a separate reader record; this edition makes no claim to have retested later versions.

Android and Windows findings apply to separate fixed builds. Their storage, login, and transport mechanisms must not be treated as interchangeable.

AI assistance disclosure

AI tools assisted with organizing research records, drafting, translation, editing, and publication layout. AI-generated prose is not experimental evidence. Evidence claims must be read together with the fixed-build records, controls, and limitations stated in the manuscript.

Intended use

For authorized defensive research using owned samples, read-only copies, and synthetic data. The publication does not distribute real accounts, credentials, private messages, working decryption or key-recovery tools, authentication bypasses, active-send callers, or bulk extraction tooling.

Navigation Contents

Part I - Evidence Narrative and Architectural Findings

This part preserves the complete fixed-build evidence narrative: sample boundaries, key objects, fields, call chains, state machines, counterevidence, and defensive implications. It answers what the findings are and where their evidence resides. Part II rewrites those findings as experiment cards so a reader can determine how to validate them on an owned sample, what constitutes a pass, and when a claim must be downgraded.

> Reproducibility principle: reproduce methods, structures, criteria, and evidence grades-not third-party accounts, secret material, or active-call capability. Every public lab uses owned samples, read-only copies, and synthetic data.

Scope and Publication Boundary

This book documents fixed-version defensive research on WeChat clients. The Android section covers the database evidence in 8.0.71 and the text/image paths in 8.0.71 and 8.0.72. The Windows section covers module topology, standard cryptographic components, the MMTLS boundary, a wrapped-key structure, and database state in 4.1.9.35.

The public edition retains sample versions, function or class roles, request field numbers, media objects, state transitions, binary structures, version deltas, validation procedure, and negative findings. It removes live accounts, device identifiers, directories, passwords, keys, tickets, CDN URLs, message content, exact RVAs, working database-recovery scripts, and active-send calls.

Android and Windows are distinct implementations. They can be compared through the abstraction “local state-business object-task queue-secure transport,” but an Android derivation must not be projected onto Windows, and Windows login/session material must not be presented as Android's database design.

> The objective is to explain verified structures and methods, not to provide a tool for operating a real account.

Chapter 1 - Samples, Questions, and Evidence Grades

Platform	Sample	Principal evidence
Android	8.0.71	Database derivation, Tinker patch, main text path, baseline image path
Android	8.0.72	Active-text-path correction, image namespace drift, actual media task path
Windows	4.1.9.35	Weixin.dll/ilink2.dll, OpenSslCryptoUtil, MMTLS, 168-byte wrapper, business database boundary

Android questions concern the path from account and device/install-state material into the database password, the effect of Tinker patches, text-to-CGI 522, the two-stage CDN image path, and the reason the classic entry fails in 8.0.72. Windows questions concern module responsibilities, standard primitives, segmentation of a 168-byte object, separation of MMTLS from business serialization, and the inability to recover a usable business database state after restart from disk material alone.

Grade	Definition
E8	Runtime hit, database state, and end result agree
E7	Static call chain and runtime arguments agree
E6	Generated fields, data flow, database evidence, or standards vectors are strong
E5	Independent adjacent-version comparison agrees
E3	Controlled sample, timeline, or statistical evidence
H	Plausible explanation without a complete call closure
R	An earlier claim disproved and retired

Chapter 2 - Android Local-State Map

An Android account area contains business databases, preferences/configuration, and patched code. A database is not opened by one static password alone, and runtime state is not the only source of every material. The study separates four propositions: a file exists; a value is readable; a database is verifiable; and an account is currently logged in.

Layer	Observable object	What it proves	What it does not prove
Account area	Isolated account directories and databases	Account-scoped boundary	Current online state
Preferences/config	Account and device/install candidates	Possible derivation inputs	Guaranteed use in the final formula
Tinker patch	Replaced method implementation	Active-build logic	Continued activity of base.apk code
SQLCipher	nativeSetKey/sqlite3_key boundary	Password reaches page decryption	How upstream material was produced

Controlled logout testing shows that sufficient local material can remain to validate a database offline. The conclusion is that login state and data verifiability differ-not that third-party data should be extracted.

Part II - Full Experimental Validation Manual

Laboratory Baseline

Create a separate directory for every experiment and begin with a `sample-card`, `toolchain-card`, and `experiment-log`. The sample card records app version, versionCode, ABI, SHA-256, acquisition date, and authorization scope. The toolchain card records versions of decompilers, signature parsers, database tools, and scripts. The experiment log records start/end time, unique lab id, actions, observations, anomalies, and hashes of raw evidence. No live account, device, contact, conversation, message, media, credential, key, endpoint, or exact address may enter the public manuscript.

Common Isolation Conditions

1. Use only test accounts, devices, and application artifacts that you own or are explicitly authorized to study.

2. Preserve originals read-only; analyze copies and recompute hashes before and after each lab.

3. Dynamic observation is read-only and pass-through: never alter returns, bypass checks, or act on behalf of third parties.

4. Network-facing tests use minimal synthetic messages; structural work that can be completed offline remains offline.

5. Every causal claim requires at least one negative control; correlation without a control is not promoted to causation.

6. For a new build, remap tags, fields, states, and outcome anchors before using any old obfuscated name or address.

Record Format

Each card has fourteen elements: research question; sample/build; environment and prerequisites; static anchors; inputs; structure; outputs and side effects; analysis path; positive procedure; expected and observed evidence; negative controls; failures and corrections; version/evidence/publication boundary; and defensive meaning with pass/fail/inconclusive criteria. “Observed result” in this book belongs to the author's fixed-build evidence; readers must record their own result separately rather than overwrite it.

Experiment 02 | WX-L02 | Recover Android Storage Topology from Directories, Callers, and Diffs

1. Research Question

How are account directories, the main message store, configuration material, and patch logic separated, and why is a single static-password model insufficient?

2. Sample, Build, and Evidence Scope

Lab id: `WX-L02`; evidence mapping: `WX-02`; fixed sample: `WeChat Android 8.0.71`. This lab makes claims only for the listed sample.

3. Environment, Authorization, and Prerequisites

Follow the laboratory baseline in this part. Keep originals read-only, pin tool versions, restrict dynamic observation to owned accounts with pass-through behavior, and publish synthetic values only.

4. Static Locators and Evidence Anchors

Account directory buckets, primary-store header, preference/config readers, SQLCipher open call, Tinker patch dex, and runtime stack.

5. Inputs, Provenance, and Constraints

Four read-only snapshots for one owned test account: pre-login, post-login, post-send, and post-logout. Hash live directory names in public output.

6. Data, Protocol, or Object Structure

Topology ledger: layer, container, creation/modification time, readers/writers, account scope, logout behavior, and publication class.

7. Outputs, State Changes, and Side Effects

Account directory, business store, configuration/installation material, and patch code occupy distinct layers; opening depends on combined runtime material, not a constant inferable from a filename.

8. Analysis Path and Call Chain

State action → file diff → container identification → readers/writers → patch/base ownership → database-open material flow.

9. Positive Validation Procedure

1) Stop the app and capture pre-login; 2) log in and repeat; 3) send unique synthetic text and snapshot; 4) log out, stop, enable airplane mode, and snapshot; 5) locate changes by hash/mtime/page diff; 6) trace file readers/writers; 7) separate base APK from patch dex; 8) confirm against UI state.

10. Author Observation and Reader Record

Expected/observed pattern in the author's fixed build: Account directory, business store, configuration/installation material, and patch code occupy distinct layers; opening depends on combined runtime material, not a constant inferable from a filename. Readers create a separate result column containing actual values, timing window, raw-evidence hash, and any divergence.

11. Negative and Control Experiments

Filename-only classification should leave unexplained objects. Base-APK-only analysis should miss the active branch. Persisting files after logout do not prove an active login.

12. Failed Hypotheses, Anomalies, and Corrections

File existence, offline verifiability, and active login are three different claims. Subtract background noise with a no-action baseline.

13. Version, Evidence Grade, and Publication Boundary

Evidence E1+E3; publication B. Live account directories, preference values, and business content are C.

14. Defensive Meaning and Pass/Fail Criteria

Pass: four-state diffs agree with callers. Fail: path-only inference. Inconclusive: patch dex unavailable. Use for data-lifecycle and logout-cleanup audits.

About the Full Edition

The full edition of **Defensive Reverse Engineering of WeChat Clients**, by JONE Security Research, contains the complete evidence narrative and 15 experiment cards. Each experiment includes scope, inputs, structures, positive validation, controls, failed hypotheses, version boundaries, and pass/fail criteria.

Find the full edition on Leanpub by its exact title and author. This sample preserves original chapter and experiment identifiers; the navigation contents above list only the included excerpts.

Android and Windows findings apply to separate fixed builds. Their storage, login, and transport mechanisms must not be treated as interchangeable.