



Least Authority
PRIVACY MATTERS

Joey Wallet Browser Extension
Security Audit Report

First Ledger

Final Audit Report: 9 October 2026

Table of Contents

[Overview](#)

[Background](#)

[Project Dates](#)

[Review Team](#)

[Coverage](#)

[Target Code and Revision](#)

[Supporting Documentation](#)

[Areas of Concern](#)

[Findings](#)

[General Comments](#)

[System Design](#)

[Code Quality](#)

[Documentation and Code Comments](#)

[Scope](#)

[Specific Issues & Suggestions](#)

[Issue A: Verified Debits Can Be Hidden From Batch Approval Review](#)

[Issue B: Reviewed Transactions Are Not Bound to the Selected Network](#)

[Issue C: Submission Receipts Are Lost Before Retry](#)

[Issue D: Signing Re-Prepares Transactions After Review](#)

[Issue E: Risk Checks Permit an Unexpiring Future-Sequence Signature](#)

[Issue F: Recovery Password Persists After Cancellation](#)

[Issue G: Cryptographic Self-Test Does Not Prevent Wallet Operation](#)

[Issue H: Password Validation Rates Repeated Patterns as Very Strong](#)

[Issue I: Transaction History Can Display Fabricated Incoming Transfers](#)

[Issue J: An Earlier Unlock Can Reverse a Later Lock](#)

[Issue K: Approval Rate Limits Can Be Multiplied and Reset](#)

[Suggestions](#)

[Suggestion 1: Enforce Approval Validity at Execution Time](#)

[Suggestion 2: Add End-to-End Tests for Review and Signing Invariants](#)

[About Least Authority](#)

[Our Methodology](#)

Overview

Background

First Ledger has requested that Least Authority perform a security audit of the Joey Wallet Extension, a non-custodial browser extension wallet for the XRP Ledger (XRPL).

Project Dates

- **September 17, 2026 - September 28, 2026:** Initial Code Review (*Completed*)
- **September 30, 2026:** Delivery of Initial Audit Report (*Completed*)
- **October 8, 2026:** Verification Review (*Completed*)
- **October 9, 2026:** Delivery of Final Audit Report (*Completed*)

Review Team

- Nikos Iliakis, Security Researcher and Engineer
- Burak Atasoy, Project Manager
- Jessy Bissal, Technical Editor

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Joey Wallet Browser Extension followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository was considered in scope for the review:

- joey-wallet-extension:
<https://github.com/first-ledger/joey-wallet-extension>

Specifically, we examined the following Git revision for our initial review:

- `fffaf64bb639e2dfafd5b56515cc4d55e7726008`

For the verification, we examined the following Git revision:

- <https://github.com/first-ledger/joey-wallet-extension/tree/audit-fix-implementation>
- `9b4e4f86e3950b6b38032b12b6fb064e55a45b71`

For the review, this repository was cloned for use during the audit and for reference in this report:

- joey-wallet-extension:
<https://github.com/LeastAuthority/joey-wallet-extension>

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- **Joey Wallet Extension Repository Structure Details** (*shared with Least Authority via Telegram on 3 September 2026*)
- Website:
<https://firstledger.net>

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the wallet;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Malicious attacks and security exploits that would impact the wallet;
- Vulnerabilities in the wallet code and whether the interaction between the related network components is secure;
- Exposure of any critical or sensitive information during user interactions and use of external libraries and dependencies;
- Leakage or exposure of the private key or recovery phrase outside the application;
- Proper management of encryption and storage of private keys;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

Joey Wallet is a non-custodial wallet for managing assets and interacting with decentralized applications on the XRP Ledger. Our review covered its Manifest V3 browser extension for Chrome and Firefox, which consists of an in-page provider and isolated content-script bridge, a background service that holds signing authority, extension-owned wallet and approval interfaces, a key management package (keycore), and a transaction preparation and simulation package (xrp1-tx). Ledger hardware wallets are supported through an offscreen extension document on Chrome.

We reviewed key storage, wallet locking and unlocking, onboarding and recovery, dApp request admission, transaction simulation and review, signing and submission, and the presentation of transaction history. The main findings concern transaction state that is checked in one form and signed or submitted in another, and security state that does not remain consistent when the network, lock status, or request context changes.

System Design

The extension applies access controls at its message boundaries. The background rejects messages from foreign extension IDs and classifies wallet surfaces by exact extension document path, so that website requests are routed to dApp admission rather than privileged handlers. Website origins are derived from browser-provided sender metadata. Grants bind an origin to specific accounts, embedded frames require contextual approval, and grants and dApp policies are checked again when signing. We did not identify a path from website content to privileged wallet handlers or to key extraction.

Software secrets are encrypted with a random vault key, which is wrapped using a key derived from the user's password with script and AES-GCM. Associated data binds each ciphertext to its storage slot. The unlocked vault key is held in session storage with a default 15-minute inactivity deadline, and the wallet locks on operating system screen lock and on extension update. Signing requires the password by default, and revealing recovery material, removing accounts, and changing the password always require it.

The transaction review screen simulates each request against the current open ledger and classifies risks such as missing expiry, future sequence numbers, and high fees. Warnings marked for acknowledgement must be confirmed before approval is enabled. A 2 XRP fee cap generally applies to wallet-computed and dApp-supplied fees, with an exception for AccountDelete ([Issue D](#)). A dApp policy also rechecks network identifiers, fees, and X-addresses before signing. However, several findings identify gaps between the transaction effects or risks presented during review and the transaction later signed ([Issue A](#), [Issue B](#), [Issue D](#), and [Issue E](#)).

On the XRP Ledger, an account's address is derived from its key and is identical on every network, and signed transaction blobs do not require a network identifier on Mainnet. As a result, a transaction reviewed against Testnet can be valid on Mainnet. Similarly, network fees are destroyed rather than paid to a recipient, and transactions without a LastLedgerSequence field have no ledger-based expiry, although execution remains subject to sequence and other validity requirements. These properties make the binding between review and signing particularly important for this wallet.

Dependencies

The project restricts dependency installation scripts with an empty `onlyBuiltDependencies` allowlist and includes scripts that audit declared dependencies and bundle contents. We recommend continued monitoring of security advisories for cryptographic, XRPL, and hardware wallet dependencies. The Joey Wallet SDK is consumed through a local link dependency and was not available for review.

Code Quality

The codebase separates background services, request handlers, and UI components, and uses TypeScript throughout. However, several security-sensitive values, including the selected network, the prepared transaction, and the lock state, are rederived from global state at multiple points rather than passed as explicit parameters. The issues in this report follow from a small set of such recurring patterns rather than from widespread implementation oversights: the network is reread after review ([Issue B](#)), the transaction is reprepared before signing ([Issue D](#)), and a lock that completes during an unlock is overwritten by the earlier operation ([Issue J](#)). Passing these values explicitly and retaining the prepared transaction produced during review so that the transaction signed is the one reviewed would make the intended invariants easier to enforce and review, while reducing the risk of similar regressions.

Tests

The repository includes extensive unit tests for the background, provider, ledger, data, and UI layers. These suites passed despite the issues described in this report, and no test covers the complete request-to-signing-to-submission flow.

Documentation and Code Comments

No project documentation describing the extension's architecture, trust boundaries, or key management design was provided for this review.

We recommend expanding the project documentation to describe the extension architecture and trust boundaries, including message flow, key storage, lock and unlock behavior, and the approval lifecycle.

Code comments are detailed and frequently explain the security reasoning behind design decisions, which aided our review. However, the startup comment states the self-test fails closed, and its failure log indicates that the wallet is "refusing to operate," although the handler only locks the wallet without establishing a persistent failure state. Similarly, the recovery comment states that the password is dropped "the instant the reveal returns," although this occurs only after a successful reveal.

Scope

The review covered the Joey Wallet Extension runtime components. The Joey Wallet SDK was not included in the scope of this review, nor were the backend services. We recommend a follow-up audit covering the SDK and backend services.

Specific Issues & Suggestions

We list the issues and suggestions identified during the review in descending order of severity. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	SEVERITY	STATUS
Issue A: Verified Debits Can Be Hidden From Batch Approval Review	High	Resolved
Issue B: Reviewed Transactions Are Not Bound to the Selected Network	Medium	Resolved
Issue C: Submission Receipts Are Lost Before Retry	Medium	Resolved
Issue D: Signing Re-Prepares Transactions After Review	Medium	Resolved
Issue E: Risk Checks Permit an Unexpiring Future-Sequence Signature	Medium	Resolved
Issue F: Recovery Password Persists After Cancellation	Medium	Resolved
Issue G: Cryptographic Self-Test Does Not Prevent Wallet Operation	Medium	Resolved
Issue H: Password Validation Rates Repeated Patterns as Very Strong	Medium	Resolved
Issue I: Transaction History Can Display Fabricated Incoming Transfers	Medium	Resolved
Issue J: An Earlier Unlock Can Reverse a Later Lock	Low	Resolved
Issue K: Approval Rate Limits Can Be Multiplied and Reset	Low	Resolved
Suggestion 1: Enforce Approval Validity at Execution Time	Informational	Resolved
Suggestion 2: Add End-to-End Tests for Review and Signing Invariants	Informational	Resolved

Issue A: Verified Debits Can Be Hidden From Batch Approval Review

Location

[packages/xrpl-tx/src/simulate.ts](#)

[apps/extension/src/ui/screens/review/TransactionReview.tsx](#)

[apps/extension/src/ui/screens/approval/BatchedTxRow.tsx](#)

[apps/extension/src/background/handlers/tx.ts](#)

Synopsis

A connected website can add a deliberately failing transaction to a batch request so that the review screen omits the verified cost of an earlier transaction in the same batch, while approval remains enabled. The user can therefore approve a material debit without seeing its amount.

Preconditions

The attacker must control a website that the user has connected to the wallet.

The batch must contain a transaction whose cost is not stated in its own fields, such as an `NFTTokenAcceptOffer`, followed by a transaction that fails with `tecPATH_DRY` or `tecNO_LINE`. The user must approve the batch.

Likelihood

Medium.

The failing transaction can be constructed deterministically by a connected website. The review displays a pending-effect message, but it does not disclose the known debit or require acknowledgement of that omission.

Impact

High.

The user can authorize a material transfer of XRP or tokens without the amount being displayed, subject to available funds and ledger validity rules.

Severity

High.

Technical Details

The simulator evaluates each transaction in a batch against the current open ledger without applying earlier transactions. To avoid rejecting legitimate batches in which a `TrustSet` creates a trust line a later payment needs, `simulate()` excuses certain failures as deferred. Two of the three checks require evidence of a dependency. They are joined by the nullish-coalescing operator, so the third, `unjudgeableInPosition()`, runs precisely when neither has identified a dependency. It excuses any `tecPATH_DRY` or `tecNO_LINE` in a non-first position by attributing it to the immediately preceding transaction, substituting an assumed dependency for the missing evidence. `tecPATH_DRY` also indicates insufficient liquidity and can be produced deliberately by a payment along a nonexistent path.

A deferred transaction sets the batch-wide `effectsIncomplete` flag. The balance-change component then returns before displaying any individual balance change, including known changes from other transactions in the batch, and selects its explanatory text from the deferral type. A line-related deferral

produces the trust-line and minimum-amount message, which the position-only fallback can select without establishing the claimed dependency. No other element supplies the missing figure, as BatchedTxRow does not render balance changes and NFTTokenAcceptOffer carries an offer identifier rather than a price. Submission is sequential, so a later failure does not roll back an earlier validated transaction.

Approval availability does not depend on willSucceed at this revision, and TransactionReview now requires only a simulation result and acknowledgement of the applicable risk warnings. Separately, accumulateBalanceChanges() returns without marking effects as unknown when a successful simulation response carries no metadata, producing an affirmative "No balance change" review.

Mitigation

We suggest that users reject any batch request that displays the pending-effect message, as the review cannot show what the batch will cost.

Remediation

We recommend removing the position-only deferral so that incomplete effects are attributed to an earlier transaction only when the dependency is established. Incomplete effects should be tracked per transaction, known balance changes should remain visible, and a successful response without usable metadata should be treated as unknown rather than zero. These changes concern accurate review information and do not require restoring the removed dry-run success gate.

In addition, NFT offer amounts should be resolved and displayed before approval, independently of simulation results.

Status

The Joey Wallet team has resolved the issue.

Verification

Resolved.

Issue B: Reviewed Transactions Are Not Bound to the Selected Network

Location

[apps/extension/src/background/rpc.ts](#)

[apps/extension/src/shared/types.ts](#)

[apps/extension/src/background/handlers/approve.ts](#)

[apps/extension/src/background/handlers/tx.ts](#)

[apps/extension/src/background/router.ts](#)

[apps/extension/src/ui/screens/deleteAccount/DeleteReview.tsx](#)

[apps/extension/src/ui/screens/deleteAccount/DeleteAccountFlow.tsx](#)

[apps/extension/src/ui/screens/collectibles/NftReview.tsx](#)

[apps/extension/src/ui/screens/collectibles/NftBurnFlow.tsx](#)

Synopsis

DApp approvals and wallet-initiated flows do not record the network on which they were reviewed. Signing and submission read the currently selected network, so a transaction reviewed on Testnet can be signed and submitted on Mainnet if the network is changed before the user confirms.

Preconditions

A dApp request or wallet flow, such as account deletion or an NFT burn, must remain open for review. The selected network must change before confirmation, and the account and transaction must be valid on the newly selected network.

Likelihood

Low.

A website cannot change the selected network. However, a switch can occur naturally while an approval remains open for its five-minute lifetime, and a website can instruct the user to switch networks.

Impact

High.

A transaction reviewed with test funds can transfer real funds, delete a Mainnet account, or irreversibly burn a Mainnet NFT with the same token identifier.

Severity

Medium.

Technical Details

The dApp request handler compares the requested chain against the selected network at admission, and then omits it from the queued approval. `PendingApproval` has no network field. After the user approves, `runApproval()`, `autofillForSigning()`, and `signAndSubmitDetailed()` each call `readSettings()` and use the network selected at that time. None of these functions accepts a network parameter. The `network.set` handler updates settings and broadcasts the change, but does not invalidate pending approvals.

These functions read mutable global state independently. `autofillForSigning()` and `signAndSubmitDetailed()` serve different execution branches. They do not perform three consecutive reads in every request. Asynchronous work separates review from execution, and the sign-and-submit branch also waits for ledger validation while processing a batch. A single network check at execution start does not bind the earlier review to the network subsequently selected by a downstream helper. The reviewed network should be passed explicitly through the applicable preparation, signing, and submission path.

Wallet-initiated flows have the same defect. `DeleteReview`, `NftReview`, and `NftBurnFlow` rebuild their transactions only when the address, destination, or token changes. After a network change, `TransactionReview` updates the network label but retains the earlier simulation, acknowledgements, and any open password prompt.

Mitigation

We suggest that users avoid changing networks while a review or approval is open.

Remediation

We recommend recording the network in each approval and review at admission, passing it explicitly to autofill, signing, and submission, and rejecting execution if it differs from the selected network. A

comparison in `runApproval()` alone is insufficient because the applicable downstream helper can independently read a different selected network after that check. Changing the network should invalidate pending signing approvals and open reviews.

Status

The Joey Wallet team has resolved the issue. Reviews and approvals now carry their network, and selection is rechecked before signing and broadcasting.

Verification

Resolved.

Issue C: Submission Receipts Are Lost Before Retry

Location

[apps/extension/src/background/handlers/ledgerRpc.ts](#)

[apps/extension/src/background/handlers/tx.ts](#)

[apps/extension/src/background/router.ts](#)

[apps/extension/src/background/approvals.ts](#)

[apps/extension/src/provider/bridge.ts](#)

[apps/extension/src/ui/screens/review/failure.ts](#)

[apps/extension/src/ui/screens/send/SendFlow.tsx](#)

[apps/extension/src/ui/screens/approval/ApprovalActions.tsx](#)

Synopsis

The `signAndSubmitDetailed()` path can lose the current transaction's hash when submission fails ambiguously. The wallet's send flow offers a retry that prepares and signs the transaction again. If the first payment was validated and the replacement is also accepted, the same intended payment can be made twice. The separate bulk submission path already retains signed blobs and hashes in its outcomes. It must not be described as uniformly losing them.

Preconditions

A submission must fail ambiguously, for example, due to a network error, timeout, or background service worker restart after the transaction has been broadcast.

The original transaction must be validated by the ledger, and the user or the dApp must retry the operation.

Likelihood

Low.

Ambiguous failures occur during standard network disruption and do not require an attacker, but the original transaction must also succeed.

Impact

High.

The user can send the same payment twice, or re-execute the completed transactions of a partially submitted batch.

Severity

Medium.

Technical Details

The hash of a signed transaction is known before broadcast and identifies the transaction for later reconciliation. `submitAndWait()` receives the hash but calls `submit()` before its reconciliation logic, so in `signAndSubmitDetailed()` an exception from that call propagates without a receipt. An unvalidated result instead attaches an error carrying earlier successful submissions but not the current transaction's hash. The router forwards the recognized bulk-failure field and normalizes unreachable and timeout errors to a message key, and `failureFromError()` retains an engine result or an error code but never a receipt. The retry action in `SendFlow` then returns to review and reruns autofill and signing. If the first transaction was validated, the account sequence has advanced and the new transaction is a second valid payment.

The separate `submitSignedBatch()` path retains the signed blob and hash for both an exception and a pending result, marking the outcome unknown, and its `BulkFailureReport` distinguishes partial progress from ledger outcomes. The finding is therefore inconsistent receipt handling across paths and the absence of a durable reconciliation record, not a loss of the hash on every submission failure.

Related paths also lose submission state. The bridge times out approval requests after five minutes while `MAX_SUBMIT_WAIT_MS` allows ten, discarding the later result, and `discardOrphans()` clears the approval queue on service worker restart, including entries that were mid-submission, while the corresponding ledger state is durable.

Partial-batch reporting has its own protections: `approvals.fail()` leaves the approval window open, and `ApprovalActions` suppresses the expiry notice once a report is recorded. Whether a report survives the queue refresh that follows was not verified in a browser.

Mitigation

We suggest that users refrain from retrying a failed submission and instead contact support, as a transaction that has not yet appeared may still be applied.

Remediation

We recommend persisting a receipt containing the network, batch position, signed blob, hash, and `LastLedgerSequence` before each broadcast and reporting it through a single typed result for all failure paths. A retry should first query outstanding receipts and should not sign an equivalent transaction while any receipt remains pending or unknown. Restart and expiry handling should discard approvals without discarding their receipts, and failure reports should remain visible until dismissed.

Status

The Joey Wallet team has implemented storage of the transaction receipts.

Verification

Resolved.

Issue D: Signing Re-Prepares Transactions After Review

Location

[apps/extension/src/background/data/account.ts](#)

[apps/extension/src/background/handlers/build.ts](#)

[apps/extension/src/background/handlers/approve.ts](#)

[apps/extension/src/background/handlers/tx.ts](#)

[apps/extension/src/background/handlers/ledgerRpc.ts](#)

[packages/xrpl-tx/src/simulate.ts](#)

[packages/xrpl-tx/src/config.ts](#)

[apps/extension/src/ui/screens/deleteAccount/DeleteReview.tsx](#)

Synopsis

Review simulation and signing independently prepare transactions. The fee shown during review is not bound to the fee later signed. Sequence and expiry may also be recomputed, although autofilled sequence and expiry are not ordinarily displayed as review rows. For AccountDelete, the node-reported reserve-based fee is signed without a wallet policy ceiling.

Preconditions

The values returned by the node must change between review and signing. This can result from standard ledger changes, a protocol amendment, a different node, or a faulty or compromised node.

For the fee to exceed the 2 XRP cap, the transaction must be an AccountDelete.

Likelihood

Low.

Ordinary drift between the two requests is common but typically small. A substantial increase requires unusual node behavior or a compromised node.

Impact

High.

The wallet uses the node-reported owner reserve increment to prepare an AccountDelete fee without comparing it against the reviewed fee or applying its standard fee cap. If the transaction is accepted and the fee charged, the XRP is destroyed rather than transferred. The supported concern is an unreviewed fee increase. Signing alone does not establish that the ledger will accept or charge the transaction.

Severity

Medium.

Technical Details

During review, `simulateTransactions()` autofills a copy of the transaction and returns the displayed fee, while the UI retains the unfilled transaction. After approval, `signAndSubmitDetailed()` and

`autofillForSigning()` autofill that unfilled transaction again with a new `server_info` request, and no digest or bound from the review is compared before signing.

`computeFee()` normally caps the computed fee at `MAX_FEE_DROPS` (2 XRP), but for `AccountDelete` it returns the node-reported `reserve_inc_xrp` before the cap is applied, and `capSuppliedFee()` exempts `AccountDelete`. The exemption accommodates the reserve-based fee, but there is no alternative policy ceiling and no requirement for the signed fee to equal the reviewed fee. A node reporting 500 XRP at signing after 0.2 XRP at review would produce an increase of 499.8 XRP.

DApp policy validates supplied fee format, limits supplied fees when autofill is disabled, and rejects `NetworkID` and X-addresses, but none of these compares a prepared transaction against its review. `AccountDelete` is prohibited for dApp requests, so this concern applies to the wallet-initiated flow.

The reserve is read dynamically from the ledger, so the defect is the missing comparison between the reviewed and signed fee rather than a hard-coded value. Completing the flow promptly narrows the window for change, but it does not bind the recomputed fee.

Mitigation

We suggest that users complete an account deletion promptly after reviewing it rather than leaving the confirmation open, because the fee is recomputed at signing.

Remediation

We recommend binding the signed transaction to the reviewed fee or to an explicitly approved fee bound. A field-specific policy should be defined for sequence and expiry by preserving the intentional, bounded refreshes documented by `previewWindowDivergence.test.ts` and requiring renewed review when a refreshed transaction falls outside that policy.

In addition, `AccountDelete` should use a policy ceiling appropriate to a reserve-based fee, and node-reported reserve values should be validated against expected ranges.

Status

The Joey Wallet team has implemented retention and comparison of the reviewed transaction before signing. Fee, sequence, and expiry changes are now bounded.

Verification

Resolved.

Issue E: Risk Checks Permit an Unexpiring Future-Sequence Signature

Location

[apps/extension/src/ui/screens/review/warnings.ts](#)

[packages/i18n/src/locales/en/translation.json](#)

[packages/xrpl-tx/src/risk.ts](#)

[apps/extension/src/provider/methods.ts](#)

[packages/keycore/src/sign.ts](#)

[apps/extension/src/background/dappPolicy.ts](#)

Synopsis

A connected website can request a signature over a payment with a future sequence number and no expiry without triggering either corresponding warning for that payment. Other batch entries and independent risk checks can still produce warnings requiring acknowledgement. The website can retain the signed payment and submit it when the account reaches the required sequence, provided the transaction remains otherwise valid.

Preconditions

The attacker must control a website that the user has connected to the wallet and make a sign-only batch request with autofill disabled. The user must approve the request.

Likelihood

Low.

The attacker must control a connected website, and the request must be a sign-only batch with autofill disabled, a specific non-default signing mode that the user must still approve. The raw sequence display alone is not treated as a protective control. The Low rating rests on the request shape and the explicit approval requirement.

Impact

High.

The website obtains a payment authorization with no ledger-based expiry. It can become executable after the account's next ordinary transaction, subject to the required sequence and other validity conditions.

Severity

Medium.

Technical Details

The risk classifier raises `riskNoExpiry` when a final transaction has no `LastLedgerSequence`, and `riskFutureSequence` when its `Sequence` exceeds the account's next sequence plus the transaction's batch index. Both require acknowledgement before approval is enabled.

The expiry check tests `LastLedgerSequence === undefined` and runs for every eligible entry. The transaction parser retains field values, including JSON `null`, after validating the transaction object and type, so a null expiry differs from an absent field during classification, while `omitNullish()` removes null-valued fields before signing. An absent expiry on any entry still raises the warning. Because `riskNoExpiry` carries no values, repeated occurrences share the identity `riskNoExpiry|{}` and merge into a single checkbox, which reduces duplication without removing the acknowledgement requirement. DApp policy checks fee format and limits, `NetworkID`, and X-addresses, but supplies no expiry normalization. The finding concerns consistency between these existing checks and signing, not an absence of risk checks.

The future-sequence check tests `sequence > next + index`, where `index` is the position in the batch. The allowance accommodates ordinarily numbered batches by assuming each earlier position advances the account sequence by one. It does not model the actual per-account sequence effect of preceding transactions. A ticket-spending transaction is skipped by this check yet still occupies a position and contributes to the allowance for later entries. Skipping it suppresses no other warning, and the expiry check still applies to that entry.

After excluding pseudo-transactions, the check reads only Sequence, Account, the account-sequence context, and the batch index, so TicketCreate cannot be categorically excluded as an earlier entry. Passing this positional check is distinct from being valid on the ledger or from how execution changes the account sequence, and an absent future-sequence warning alone does not establish a future-executable authorization.

Mitigation

We suggest that users reject sign-only batch requests from websites, particularly those displaying a sequence number higher than the account's current sequence number, because the wallet does not broadcast what it signs and the website retains the signature.

Remediation

We recommend treating both null and absent expiry values as missing in the final-transaction check, for example, with `context.final === true && tx.LastLedgerSequence == null`. This targeted correction closes the reported null-versus-absent warning omission: an eligible transaction whose expiry is removed by nullish normalization would require the no-expiry acknowledgement. It still permits signing after acknowledgement and does not correct sequence accounting.

In addition, we recommend normalizing untrusted transaction JSON at the parser boundary so that risk classification and signing use a consistent representation. The future-sequence check should account for the actual sequence effects of preceding transactions from the same account, including ticket use and transaction-specific advancement, instead of using their batch positions. The combination of a future sequence and no expiry should also be rejected, and regression coverage should be added for null, absent, and valid expiry values, warning aggregation across entries, and the separate sequence-accounting rules.

Status

The Joey Wallet team has resolved the null expiry handling and positional sequence accounting issues.

Verification

Resolved.

Issue F: Recovery Password Persists After Cancellation

Location

<apps/extension/src/ui/screens/account/Recovery.tsx>

Synopsis

The recovery screen retains the entered vault password when the user uses Back from the password step while the component remains mounted, or when a reveal request fails. If the retained password is correct, a person with later access to that unlocked screen can use it to reveal recovery material without independently knowing the password. Retention of an incorrect password does not establish this consequence.

Preconditions

The user must enter the correct vault password on the recovery screen and then return to an earlier step or experience a failed reveal while the recovery component remains mounted. An attacker must subsequently gain access to the unlocked wallet before the recovery screen is unmounted or the wallet locks.

Likelihood

Low.

The attacker requires local access to the unlocked wallet, and the user must abandon the reveal flow after entering the password.

Impact

High.

The attacker can obtain the account's recovery phrase or private key and take full control of the account.

Severity

Medium.

Technical Details

Recovery stores the password in the component state. The reveal function sends one `recovery . reveal` request per available secret type and clears the password only after all requests succeed. If any request fails, the password remains. The Back action on the password step changes only the phase and does not clear the password. When the user proceeds to the password step again, the field is populated with the earlier password, and the Reveal action or the password visibility toggle can be used without re-entering it.

Mitigation

We suggest that users close the recovery screen rather than navigating back after entering their password.

Remediation

We recommend clearing the password before the first asynchronous reveal request and on Back, cancellation, failure, account change, and unmount. In addition, all selected secrets should be retrieved in a single background request.

Status

The Joey Wallet team has implemented the recommended changes.

Verification

Resolved.

Issue G: Cryptographic Self-Test Does Not Prevent Wallet Operation

Location

[apps/extension/entrypoints/background.ts](#)

[apps/extension/src/background/router.ts](#)

[packages/keycore/src/index.ts](#)

Synopsis

The startup self-test intended to detect key derivation regressions runs asynchronously alongside standard message handling, and its failure locks the wallet without establishing a persistent failure state.

If a derivation defect is present, the failure does not prevent later unlock and account operations. This finding does not establish that the reviewed revision contains a cryptographic derivation defect.

Preconditions

A dependency update, build error, or platform defect must change a key-derivation output covered by the startup self-test.

The user must create or import an account after the self-test fails or while it is still running.

Likelihood

Low.

The failure requires a defect in a cryptographic dependency or the build.

Impact

High.

A relevant derivation defect could produce an account whose displayed recovery phrase does not restore the same address, resulting in loss of funds if the extension profile is lost. That consequence is conditional on such a defect, rather than an observed recovery failure in this review.

Severity

Medium.

Technical Details

`selfTest()` is started without being awaited, and its rejection handler calls `vault.lock()`. The message listener is registered independently, so requests can be processed before the test completes. After a failure, `vault.unlock` and `vault.create` remain in the set of requests permitted while locked, and no persistent failure state prevents subsequent operations.

Mitigation

We suggest that operators who observe the keycore self-test failure message stop account creation, import, recovery, and signing operations until the failure is investigated. Checking the background console can reveal the failure but does not itself prevent continued use.

Remediation

We recommend establishing a persistent fatal state on self-test failure. Every key-dependent or account-changing router request should await self-test completion and reject if it failed, while a minimal set of safe status, lock, and reset operations remains available. In addition, the test vector should be extended to check the expected private key and to sign and verify a fixed message.

Status

The Joey Wallet team has made integrity failure persistent, blocking key-dependent operations.

Verification

Resolved.

Issue H: Password Validation Rates Repeated Patterns as Very Strong

Location

[packages/keycore/src/validate.ts](#)

[packages/keycore/src/vault.ts](#)

[apps/extension/src/ui/components/PasswordField.tsx](#)

Synopsis

The password validator's repetition-specific check detects only a single character repeated throughout the password. Some short repeated-block passwords, including Aa1!Aa1!Aa1!, pass the other checks and receive the highest strength score. Such a password can be substantially easier to guess than its displayed strength suggests if an attacker obtains the copied vault and encrypted software-account secrets.

Preconditions

The user must choose a repeated-block password that passes the other checks. The attacker must obtain both the vault record and encrypted account-secret records, for example, from a stolen device, malware, or a browser-profile backup.

Likelihood

Low.

The attacker requires access to the extension's storage, and the user must choose a repeated-pattern password.

Impact

High.

An attacker who guesses the password can recover the software-account secrets contained in the copied storage. This does not expose Ledger or watch-only account keys.

Severity

Medium.

Technical Details

`checkPasswordStrength()` rejects repetition using the pattern `^(.)\1+$`, which matches only a single character repeated throughout the password. The illustrated 12-character password, consisting of a 4-character block repeated three times, satisfies the length, character-class, sequence, and corpus checks. For passwords that pass the other checks, the score is computed from length and character classes, and the illustrated password receives a score of 4, which `PasswordField` displays as "Very strong." The same function is used for vault creation, password changes, onboarding, and restore.

Mitigation

We suggest that users choose a vault password that is not a repeated short sequence, regardless of the strength rating displayed. Changing the password protects future copies of the vault but not copies already obtained.

Remediation

We recommend rejecting passwords consisting of a repeated block and computing the length score from the non-repeating content of the password. In addition, regression tests should confirm that repeated and decorated repeated patterns are rejected or receive a low score, while random passwords of equal length are unaffected.

Status

The Joey Wallet team has resolved the identified corner cases.

Verification

Resolved.

Issue I: Transaction History Can Display Fabricated Incoming Transfers

Location

[apps/extension/src/background/data/activity.ts](#)

[apps/extension/src/ui/screens/settings/spam.ts](#)

Synopsis

The activity feed interprets a balance field that is absent from transaction metadata as a previous balance of zero. An account able to submit a qualifying transaction that modifies the user's account or trust line without changing its balance can cause the wallet to display the user's entire existing balance as a successful incoming transfer from the attacker.

Preconditions

The attacker must submit a transaction that modifies the user's AccountRoot or RippleState entry without changing its balance, such as canceling the user's expired NFT offer or adjusting the issuer's side of a trust line. The user must rely on the activity feed to determine whether a payment was received.

Likelihood

Medium.

The reported scenario requires a third party to submit a qualifying transaction that produces the described metadata without victim participation. This source review confirmed the mapper behavior but did not independently establish protocol acceptance or exact metadata for the example.

Impact

Medium.

The wallet displays a transfer that did not occur, which can be used to support payment fraud. Account balances are not affected.

Severity

Medium.

Technical Details

For a ModifiedNode entry, XRPL metadata includes in PreviousFields only the fields that changed. The absence of Balance from PreviousFields therefore indicates that the balance did not change. The activity mapper computes the XRP delta as the final balance minus the previous balance and uses zero when the previous balance is absent. The token delta calculation uses the same approach. The

mapper does not distinguish modified from created entries, for which a previous balance of zero is correct. The OwnerCount calculation in the same function correctly requires both values.

For the reported example of an expired NFT offer cancellation, the consequence depends on metadata modifying OwnerCount without changing Balance. Given that metadata, the mapper records the existing balance as received, attributes it to the transaction sender, and uses the successful transaction result. Such a row is not filtered as spam when the fabricated amount exceeds the applicable threshold. This source check did not independently verify ledger acceptance or the exact metadata of the example.

Mitigation

We suggest that users verify incoming payments against a block explorer rather than relying on the activity feed, because a displayed receipt does not establish that a transfer occurred.

Remediation

We recommend using the final balance as the previous balance for modified entries when Balance is absent from PreviousFields and using zero only for created entries, for both XRP and token deltas.

Status

The Joey Wallet team has resolved the issue.

Verification

Resolved.

Issue J: An Earlier Unlock Can Reverse a Later Lock

Location

[apps/extension/src/background/vault.ts](#)

[apps/extension/entrypoints/background.ts](#)

Synopsis

An unlock operation writes the session key after key derivation completes, without checking whether the wallet was locked in the meantime. If a lock occurs during key derivation, the wallet becomes unlocked again once derivation completes.

Preconditions

For this issue to occur, the following must hold true:

- The user must enter their password to unlock the wallet.
- A lock must occur during key derivation, for example, through the operating system screen lock, the inactivity timer, or the Lock button.
- An attacker must subsequently gain access to the browser.

Likelihood

Low.

The lock must occur during key derivation, and an attacker must subsequently gain access to the device.

Impact

Medium.

A completed lock can be reversed by the pending unlock, leaving the vault session active despite the user's expectation that it is locked. An attacker who later gains access to the browser can use that session. The default password-per-signature setting limits signing impact but does not restore the expected locked state.

Severity

Low.

Technical Details

`unlock()` derives the vault key with `script`, then calls `beginSession()`, which writes the key to session storage and sets a new deadline. `lock()` removes the session key and deadline, but records no state that `unlock()` can check. All lock paths, including operating system screen lock, the inactivity alarm, last-wallet-surface closure, and the Lock button, call the same function. A lock that completes during derivation can therefore be reversed when the earlier unlock installs the session key. Whether the UI announces that lock depends on the prior state. A notification is not established for every interleaving.

Mitigation

We suggest keeping password-per-signature enabled and ensuring any unlock attempt has completed before locking or leaving the device. Seeing the lock screen alone does not prove that a pending unlock cannot subsequently reinstall the session key.

Remediation

We recommend maintaining a lock epoch in session storage that is incremented by each lock and captured at the start of unlock. The final epoch check and session-key installation should be serialized with lock invalidation so that a lock cannot complete between the check and the asynchronous key write. Regression tests should complete a lock after each asynchronous step of unlock.

Status

The Joey Wallet team has resolved the issue.

Verification

Resolved.

Issue K: Approval Rate Limits Can Be Multiplied and Reset

Location

[apps/extension/entrypoints/content.ts](#)

[apps/extension/entrypoints/inpage.content.ts](#)

[apps/extension/src/provider/bridge.ts](#)

[apps/extension/src/background/approvals.ts](#)

[apps/extension/src/background/permissions.ts](#)

Synopsis

The pending-request limit is enforced separately in each frame, so aggregate requests can exceed the per-frame ceiling. The rejection immediately following the threshold-triggering rejection can also remove the active cooldown, allowing further prompts after the user rejects the queued requests.

Preconditions

The attacker must control a website that the user has connected to the wallet. The user must reject the queued requests.

Likelihood

Medium.

Multiple frames can create separate pending-request sets. The cooldown-reset behavior requires enough rejection events for the same origin within the configured thirty-second window. Executing requests are not rejected by `rejectAll()`. These exploitation conditions are source-derived conditions, not based on a measured browser outcome.

Impact

Low.

The website can flood the user with approval prompts, increasing the risk of an accidental approval. Each request still requires explicit approval.

Severity

Low.

Technical Details

The content script runs in every frame, and each bridge instance enforces `MAX_IN_FLIGHT` against its own request set. The background approval queue applies no aggregate per-origin or global limit. Two eligible frames can therefore hold up to sixteen outstanding requests if those requests pass background admission checks and remain pending for approval.

After ten rejections within thirty seconds, `recordRejection()` sets `blockedUntil` and clears the list of rejections. The next rejection constructs a new state containing only the latest timestamp, which omits `blockedUntil`, so `isThrottled()` returns false. `rejectAll()` retains executing requests and records one rejection for each removed request. For the same origin, eleven eligible rejections within the configured window can therefore install and then remove the cooldown.

Mitigation

We suggest disconnecting a website that produces repeated approval prompts and closing its pages. Revocation does not by itself cancel existing queued or executing requests or prevent a new connection request.

Remediation

We recommend enforcing per-origin and global limits on pending approvals in the background and preserving or extending an unexpired `blockedUntil` value on every rejection. In addition, regression tests should cover rejection counts at and beyond the threshold, as well as requests from multiple frames.

Status

The Joey Wallet team has introduced serialization and enforcement of approval limits and rejection throttling across origins, tabs, and the entire wallet.

Verification

Resolved.

Suggestions

Suggestion 1: Enforce Approval Validity at Execution Time

Location

[apps/extension/src/background/approvals.ts](#)

[apps/extension/src/background/handlers/approve.ts](#)

[apps/extension/src/background/permissions.ts](#)

Synopsis

Pending approvals expire after five minutes, but expiry is enforced by queue sweeps and UI checks. `claim()` does not reject an unswept expired entry. Subsequent execution remains subject to the remaining authorization and signing controls. Similarly, the approval handler checks the origin's grant once in each applicable request branch before later asynchronous signing or submission work. Revoking the grant does not cancel an approval that is already executing.

Mitigation

We recommend checking expiry in `claim()` and rechecking the origin grant immediately before signing and before submission. Revocation could also increment a grant generation that executing approvals compare against.

Status

The Joey Wallet team has implemented the mitigation.

Verification

Resolved.

Suggestion 2: Add End-to-End Tests for Review and Signing Invariants

Location

[apps/extension/test](#)

Synopsis

Several issues in this report ([Issue A](#), [Issue B](#), [Issue D](#), and [Issue E](#)) concern differences between the transaction effects or risks presented during review and the transaction later signed. `previewWindowDivergence.test.ts` already tests an intentional, bounded difference in the `LastLedgerSequence` window recomputed by the wallet's signer. This provides a precedent for an explicit policy governing which reviewed fields must remain fixed and which may change within defined bounds. The structural review did not identify complete-flow coverage enforcing that policy for network, fee, and sequence across changes in execution context.

Mitigation

We recommend defining the required relationship between reviewed and signed fields, then testing it through the complete request, review, signing, and submission flow. Tests should enforce the reviewed network and the agreed fee and sequence constraints, validate the limits and accuracy of displayed balance effects, and preserve explicitly permitted expiry refreshes within documented bounds. Changes

outside that policy should require renewed review. This should also include changes to network selection, node responses, lock state, and the approval queue at asynchronous boundaries, as well as null, undefined, and absent optional fields.

Status

The Joey Wallet team has implemented the mitigation.

Verification

Resolved.

About Least Authority

Founded in 2011 and based in Berlin, Germany, since 2016, Least Authority provides security consulting for privacy-preserving technologies and decentralized systems. We are committed to transparency, openness, and advancing secure digital infrastructure.

We believe that people have a fundamental right to privacy and that secure solutions enable people to use the internet and other connected technologies more freely. Our security consulting services help teams make their solutions more resistant to unauthorized access and unintended system manipulation. We support teams from the design phase through production launch and beyond.

Our team has experience reviewing code for security vulnerabilities and specific attack vectors in multiple languages, including Rust, Go, Python, C, C++, Solidity, Cairo, Circom, Noir, Haskell, OCaml, TypeScript, Swift, ZoKrates, and others. We have reviewed implementations of cryptographic protocols and distributed system architecture in blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, we use various tools to scan code and networks and build custom tools as necessary.

For more information about our security consulting, please visit:

<https://leastauthority.com/security-consulting>.

Our Methodology

We use a transparent review process that provides the project team visibility into our findings, assumptions, and remediation considerations throughout the audit. The goals of our security audits are to improve the quality of the systems we review and support sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Code Review

In manually reviewing all of the code, we identify potential issues, including those related to code logic, error handling, cryptographic implementations, and correctness with respect to specifications, standards, and best practices. We also evaluate areas where more defensive programming could reduce the risk of future mistakes and accelerate future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

When appropriate, we also use artificial intelligence (AI) to support the efficiency of our security audits without reducing rigor or reliability. AI-assisted outputs are carefully guided, reviewed, and validated by our team so that conclusions remain accurate and defensible. This supports stronger audit decisions, not only faster review.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and white box penetration testing, supplemented by various tools, including AI, that support the code review process. Discussions with the development team help clarify their vision for the software. Installing and using the relevant software helps us understand user interactions and roles, while also informing our analysis of threat models and attack surfaces. Design documentation, relevant papers, other audit results, similar projects, source code dependencies, and open issue tickets may also be reviewed to inform the analysis. We

identify potential vulnerabilities, and, when appropriate, create issue entries. For each entry, we follow the issue investigation and remediation process described below.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and supporting successful remediation. Whenever a potential issue is identified, we raise it with the client through a dedicated secure communication channel to discuss its validity and impact, to the extent these can be determined at that stage. This process allows potentially relevant findings to be reviewed with the client early, before they are documented in greater detail in the initial audit report. For each confirmed issue, we assess impact and feasibility and use those ratings to determine the overall severity, following the [OWASP Risk Severity Matrix](#).

Suggested Solutions

We identify immediate and comprehensive mitigations available to live deployments and then suggest remediation requirements for future releases. When code is already deployed in production, we include mitigation strategies in the report to provide interim support until appropriate remediation is available. The mitigation and remediation recommendations should be reviewed by the development team and deployment engineers. After delivering the Initial Audit Report, we remain available to support implementation of the proposed mitigations and remediations before the verification review.

Development teams should address findings in an order and with an urgency that prioritizes the security of their users. Findings should be addressed as soon as possible, but remediation should be carried out pragmatically and should not cause unnecessary disruption to project timelines.

Although issues must be handled on a case-by-case basis, we work toward a resolution timeline that balances the impact on users and the needs of the project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess whether the issues and suggestions have been sufficiently addressed. When this analysis is complete, we update the report and provide a Final Audit Report that can be published in full. If any critical issues remain, we recommend that the Final Audit Report not be published and that users and other stakeholders be alerted to the impact. We encourage all findings to be addressed and the Final Audit Report to be shared publicly to support transparency and advance security knowledge within the industry.