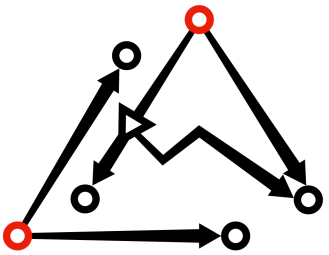

Least Authority

PRIVACY MATTERS

Wallet Extension

Security Audit Report

Rabby Wallet

Initial Audit Report: 2 September 2025

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: QR Code Wallet Address Sync Vulnerable to Eavesdropper
    - Issue B: Potential Race Condition Can Result in Invalid Transaction
  - Suggestions
    - Suggestion 1: Add Tests to Cover New Features
- About Least Authority
- Our Methodology

Overview

Background

Rabby Wallet has requested that Least Authority perform a security audit of their Chrome Extension to assess developments since the previous review, for which a final audit report was delivered on December 12, 2024.

Project Dates

- **August 11, 2025 - 21 August, 2025:** Initial Code Review *(Completed)*
- **August 22, 2025:** Delivery of Initial Audit Report *(Completed)*
- **September 1, 2025:** Verification Review *(Completed)*
- **September 2, 2025:** Delivery of Final Audit Report *(Completed)*

Review Team

- Will Sklenars, Security Researcher and Engineer
- Dominic Tarr, 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 Rabby Wallet Extension followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Code Change:
  - https://github.com/RabbyHub/Rabby/compare/v0.92.99...v0.93.41
  - https://github.com/RabbyHub/Rabby/pull/2694
    - feat: store unencryptedKeyringData and feat/unencrypted
  - https://github.com/RabbyHub/Rabby/pull/2857
    - Feature for support sync chrome extension addresses to Rabby Mobile (including private key and seed phrase) via QR Code

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

- `44c8e5a863aeee0df8a7234c306005565c1cf0a5`

For the verification, we examined the Git revision:

- `d7ef37dedc2ebc08f714da8e1e95b8b93676602b`

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

- https://github.com/LeastAuthority/RabbyHub-Rabby-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:

- Website:
  https://rabby.io
- Previous audit reports:
  - SlowMist Audit Report - Rabby mobile wallet iOS.pdf *(shared with Least Authority via email on 5 August 2024)*
  - SlowMist Audit Report - Rabby mobile wallet Android.pdf *(shared with Least Authority via email on 5 August 2024)*
  - Least Authority Audit Report - Rabby Wallet Extension:
    https://leastauthority.com/wp-content/uploads/2024/12/Least-Authority-DeBank-Rabby-Wallet-Extension-Final-Audit-Report.pdf
  - Least Authority Audit Report - Rabby Wallet:
    https://leastauthority.com/wp-content/uploads/2024/10/Least-Authority-Debank-Rabby-Wallet-Final-Audit-Report.pdf

In addition, this audit report references the following documents:

- A Large-Scale Study of Web Password Habits:
  https://web.archive.org/web/20150327031521/http://research.microsoft.com/pubs/74164/www2007.pdf

Areas of Concern

Our investigation focused on the following areas:

- Developments since the previous audit;
- 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 with the wallet and use of external libraries and dependencies;
- 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

Our team performed a security audit of Rabby Wallet, an application designed for managing assets on Ethereum and other EVM-compatible chains, available as both a mobile application and a browser

extension, and based on a fork of MetaMask. We previously audited Rabby Wallet and delivered a final audit report on December 12, 2024. Since then, several features have been introduced, including support for Gnosis Safe accounts, limited EIP-7702 support, password reset functionality, and account synchronization across the mobile application and browser extension versions of Rabby Wallet. We identified an issue with the account synchronization feature (Issue A) and a potential race condition when creating EIP-7702 transactions (issue B).

System Design

Rabby Wallet is based on a fork of MetaMask, a long-established wallet application in the Ethereum ecosystem. Since our previous report, the Rabby Wallet team has introduced updates including account synchronization and password reset functionality. The password reset feature, however, must not be mistaken for password recovery. Because the application does not store the password, recovery is not possible, and resetting it instead wipes all private information. This functionality can be beneficial when used correctly, such as wiping a device before disposal or during a transition to an offline wallet. However, it also introduces risk, since misinterpreting the feature as password recovery could result in the irreversible loss of funds.

As we noted in our previous audit, the encryption used to protect the account should not be considered secure because it depends on a user-selected password. Even though strong encryption is used, the security is weak because it must be assumed that the password is weak given the tendency of users to choose low-entropy passwords. Therefore, although the data is encrypted, it must still be treated as sensitive and kept inaccessible to attackers. While this limitation is inherent to key storage, browser and mobile application sandboxing provides a degree of protection for the encrypted data. A new feature for synchronizing account data via a QR code relies on this same encryption and password, enabling an attacker who is able to photograph the QR code to likely recover the password and thereby the funds in the account (Issue A). We also noted that some unencrypted data is encoded within the QR code, such as a list of user-whitelisted accounts. This data could be decoded by an eavesdropper without needing to brute-force a decryption key. While the information obtained would likely be of limited value to an attacker, it nonetheless poses a potential impact on user privacy.

In other areas, Rabby Wallet implements only partial support for EIP-7702 (delegate contracts) and supports EIP-1271 solely through Gnosis Safe, which limits the security risk of these features.

Dependencies

Our team did not identify any security concerns resulting from the unsafe use of dependencies.

Code Quality

We performed a manual review of the repositories in scope and found the codebase to be well organized and generally aligned with TypeScript best practices.

Tests

Our team observed that the repository has a limited testing suite. A robust suite should include unit and integration tests for both success and failure cases to catch errors and handle edge cases, which may otherwise lead to security-critical vulnerabilities. We recommend improving and expanding the test coverage to include newly added features (Suggestion 1).

Documentation and Code Comments

The repository in scope contains some documentation, including an architecture diagram, as well as a collection of previous security audits. The codebase also contains descriptive code comments.

Scope

The scope of this audit was limited to the changes to the browser extension since the previous audit conducted by our team, for which a final report was delivered on December 12, 2024.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. 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	STATUS
Issue A: QR Code Wallet Address Sync Vulnerable to Eavesdropper	Unresolved
Issue B: Potential Race Condition Can Result in Invalid Transaction	Resolved
Suggestion 1: Add Tests to Cover New Features	Planned

Issue A: QR Code Wallet Address Sync Vulnerable to Eavesdropper

Location

`background/service/keyring/index.ts#L1362-L1422`

Synopsis

Password-encrypted at-rest data should not be considered secure, and using a QR code to transfer such encrypted data between devices constitutes a form of plaintext networking that may be observed by an attacker.

Impact

High. The attacker could recover user account data by eavesdropping on the raw data during QR code transfer and then decrypting the password-encrypted account file.

Feasibility

Low. The feasibility of eavesdropping the QR code used for transfer is admittedly quite low. The attacker would need to observe the QR code when it is displayed on the screen, for example, by taking a photo from a long range using a high-quality optical zoom. In practice, the likelihood is low because users rarely transfer their accounts to another device, which implies that an attacker would already need to be surveilling the victim for another reason. However, an opportunistic attacker could exploit a situation in which a casual user synchronizes accounts in a public space.

Once the attacker has extracted the ciphertext from the QR code, brute-forcing the password should be considered highly feasible. This involves repeatedly attempting passwords until the correct one is identified. It should be assumed that most users favor a memorable password rather than a strong, high-entropy one. It is likely that many users use a password that is included in common password lists. The ability to break the password depends primarily on the attacker's persistence, as the cost of each attempt is modestly increased by PBKDF2 without it being prohibitively expensive. By default, `browser-passworder` uses `900,000` iterations, and deriving a key on a standard laptop takes only 172 ms. Therefore, a single core could attempt approximately `432,000` passwords per day. This is insufficient

to break a truly strong password, but could readily compromise moderate-strength passwords that most users are likely to select. Given that the value represented by a Rabby Wallet account may amount to thousands of dollars, attackers would have a strong incentive to attempt to brute-force attacks on user passwords.

Severity

Medium. Although the impact is high (stolen funds), the feasibility is low because transfers are infrequent and the attacker would need to physically observe the victim.

Preconditions

The attacker would need to intercept the QR code, for example, by taking a photo of the device at the right time using an optical zoom. However, research into restoring blurry or out-of-focus QR codes could enable an attacker to exploit a photo even if it is imperfect.

Technical Details

Rabby Wallet, like MetaMask from which it is forked, and similar wallets, utilizes password-keyed at-rest encryption to protect users' private keys. The data is encrypted with standard `AES-GCM`, which provides a security level of `128` bits. However, since the key is derived from a user-selected password, the effective security level is that of the password. The password security level is substantially weaker, estimated by one study as being `40.54` bits on average (note that half of all passwords have a strength below this threshold). The password is "stretched" by the PBKDF2 algorithm, thereby increasing the computational effort required to test each candidate password. However, the key-stretching algorithm must also run on the user device, which is often an underpowered mobile device, whereas an attacker attempting to reverse a password can use a powerful machine or an array of machines in parallel. Consequently, the chosen key-stretching configuration is relatively weak to avoid inconveniencing the user, which results in only a marginal reduction in the attacker's efficiency.

For different situations, a weak password can still provide sufficient protection. It is not realistic for an attacker to brute-force a password on an online service because each attempt requires a network request and the server can simply disallow many requests. With at-rest encryption, however, the attacker can make unlimited attempts in parallel, so a significantly stronger password is required to achieve a reasonable level of practical security.

For an encrypted private key that remains on the user's device, encryption protected by a weak password still adds a small additional layer of security, but the main protection is provided by the operating system and browser sandboxing, which prevent the attacker from accessing the ciphertext.

This mode of encryption is inadequate in the context of the QR Code account synchronization. This should be considered a form of insecure networking. It is possible for an attacker to physically observe the QR code from a distance. If they are able to photograph the QR code, they can attempt to reverse the password in their own time, and the victim will have no way to detect this has occurred until they discover their accounts have been drained.

Mitigation

Rabby Wallet users wishing to use the QR code account transfer feature can protect themselves by increasing the difficulty of reversing the password through the use of a high-entropy password. This significantly increases the cost to the attacker, although it also increases the burden on the user by requiring them to remember a more complex password.

Alternatively, the attack can be mitigated by interrupting the attacker's ability to intercept the QR code. The QR code is displayed on the device's screen and received by the camera sensor on the recipient device, relying on the transmissibility of visible light through air. Visible light is blocked by most household

materials, so this eavesdropping risk can be mitigated by placing a flexible cover, such as a blanket, over the user and their devices before beginning the transfer operation.

Remediation

Instead of using a QR code to transfer private keys, we recommend transferring a public key from the recipient device. The sending device should then encrypt the private keys with that public key and transmit the encrypted message over a network connection. Alternatively, QR codes may still be used, but this would require the sender to scan the recipient and the recipient to scan the sender.

Such a process can be carried out either through some form of network connection (for example, Bluetooth or another local network) or two QR code scans, but it would be fully secure against eavesdropping.

Status

The Rabby Wallet team noted that they will not address the issue, stating that physical attacks are particularly difficult to execute and they therefore do not consider this to be a risk. They further indicated that using network or Bluetooth transmission, as proposed, increases the likelihood of a middleman attack, which they regard as a greater risk and detrimental to the overall interactive experience.

Verification

Unresolved.

Issue B: Potential Race Condition Can Result in Invalid Transaction

Location

`background/controller/wallet.ts#L4575`

Synopsis

Because of the nonce-selection mechanism in EIP-7702 transactions, a race condition may result in an EIP-7702 transaction failing node validation.

Impact

Medium.

The non-compliant EIP-7702 transaction would fail node validation, and the corresponding EIP-7702 authorization change would not be recorded on the blockchain.

Feasibility

Medium.

The issue requires a nonce drift to occur, which may occur if another transaction is sent or executed concurrently.

Severity

Medium.

Preconditions

For this issue to occur, the following must hold:

- The EIP-7702 revoke path must be used.

- Some concurrent activity must occur that changes the value of `getRecommendNonce` during the relevant operation.

Technical Details

EIP-7702 requires the inner authorization object to have a nonce equal to `tx.nonce + 1`. Rabby Wallet chooses a nonce by calling `getRecommendNonce`, which returns either the local nonce or the on-chain nonce, whichever is higher. When the EIP-7702 transaction is built, only the inner authorization nonce is set initially (`authorization.nonce = getRecommendNonce() + 1`). The outer `tx.nonce` is set later by another call to `getRecommendNonce`.

It is possible that between the two calls, the value returned by `getRecommendNonce` increases (for example, if the on-chain nonce is incremented). In such a case, the relationship between the inner and outer nonce changes, resulting in `authorization.nonce != tx.nonce + 1`. This causes the transaction to fail validation at the node.

Mitigation

We suggest the following measures to mitigate this issue:

- Users should send revokes only when there are no other pending transactions on the account.
- Users should avoid multi-tab or multi-wallet concurrency for the same account during revocation.

Remediation

We recommend binding `tx.nonce` to the same nonce used to derive `authorization.nonce` in the revoke builder. We additionally recommend adding a provider-side check for EIP-7702 revoke, requiring `authorization.nonce === tx.nonce + 1` and rejecting otherwise.

Status

The Rabby Wallet team has added a line of code, which binds `tx.nonce` to `authorization.nonce`.

Verification

Resolved.

Suggestions

Suggestion 1: Add Tests to Cover New Features

Location

`tests`

Synopsis

The new features introduced are not yet covered by the testing suite. Comprehensive test coverage is essential for identifying errors at an early stage.

Mitigation

We recommend writing unit and integration tests to cover the new features introduced by the scoped code changes.

Status

The Rabby Wallet team has acknowledged this suggestion and noted that, while the recommended mitigation will not be implemented at this time, it will be taken into consideration for future releases.

Verification

Planned.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and

unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up 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.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test

code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.