Least Authority
PRIVACY MATTERS

Thanos Wallet

Security Audit Report

Tezos Foundation

Updated Final Report Version: 24 September 2020

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of the Thanos Wallet, a browser extension cryptocurrency wallet for the Tezos ecosystem, which provides users with the ability to manage NFT / XTZ tokens and interact with decentralized applications. The Thanos wallet aims to allow users to easily import accounts from other Tezos wallets and faucets and currently supports plain private keys, mnemonics, and fundraiser accounts imports.

Project Dates

- **August 17 - September 4:** Initial Review *(Completed)*
- **September 9:** Initial Audit Report delivered *(Completed)*
- **September 17 - 19:** Verification Review *(Completed)*
- **September 21:** Final Audit Report delivered *(Completed)*
- **September 24:** Updated Final Audit Report delivered *(Completed)*

Review Team

- Jehad Baeth, Security Researcher and Engineer
- Dylan Lott, Security Researcher and Engineer
- Phoebe Jenkins, Security Researcher and Engineer

Coverage

Target Code and Revision

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

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

- Thanos Wallet: https://github.com/madfish-solutions/thanos-wallet/tree/develop
    - Business Logic: https://github.com/madfish-solutions/thanos-wallet/tree/develop/src/lib/thanos
        - Extension's background process behavior: https://github.com/madfish-solutions/thanos-wallet/tree/develop/src/lib/thanos/back
        - UI for communication with the background process and the chain: https://github.com/madfish-solutions/thanos-wallet/tree/develop/src/lib/thanos/front

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

```
3441ab1f063a7de224a96c4d011fa0f549d77873
```

For the verification, we examined the Git revision:

```
702fe463008474cb8b2430c6695daf4568044bf7
```

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

Supporting Documentation

The following documentation was available to the review team:

- About Thanos Wallet: https://thanoswallet.com/about
- README: https://github.com/madfish-solutions/thanos-wallet
- Architectural Documentation: https://github.com/madfish-solutions/thanos-wallet/tree/develop/docs
- Background process in the context of browser extensions: https://developer.chrome.com/extensions/background_pages
- Least Authority's Taquito Security Audit Report: https://leastauthority.com/static/publications/LeastAuthority_Tezos_Foundation_Taquito_Report.pdf

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the wallets;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service attacks and security exploits that would impact the wallet's intended use;
- Vulnerabilities in the wallet code;
- Vulnerabilities within each component as well as secure interaction between the wallet and the blockchain;
- Secure management of private keys and assets;
- Protection against malicious attacks and other ways to exploit the wallet;
- General use of third party libraries;
- Exposure of any critical information during user interactions with the blockchain and external libraries;
- 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 found the Thanos Wallet to be well organized and logically structured. The scope of the audit was comprehensive, covering all components of the system, in addition to the way the system interacts with key dependencies, such as Taquito and the Beacon SDK.

It is clear that the Thanos Wallet system design consistently follows best practices. For example, it has a Vault system for securely storing sensitive data, which is a preferred method for the components to communicate, and anticipates integration with dApps.

As noted in this report, our team has identified several areas for improvement as it relates to the security of the Thanos Wallet, including two issues. Nonetheless, we commend the Thanos team for utilizing safe browser practices as demonstrated by the safe storage of critical private information in the storage backend.

Code Quality

The Thanos Wallet implementation consists of modules with specific and focused functionality, making it more effectively managed while remaining highly cohesive. Since classes are loosely coupled, changing one part of the software does not impact other parts of the system. This results in a future proof implementation (i.e. making future changes much easier to implement) and improves the system's overall maintainability and readability. Furthermore, this has made reviewing and comprehending the system significantly easier while conducting the audit.

While the Thanos Wallet is written in TypeScript, the project has minimal typing annotations, which negates the benefits of TypeScript's typing system. Benefits of type annotations include avoiding potential runtime errors and providing insight into how functions work together. Without consistent type annotations, such as those labelling the return values of functions, TypeScript's type checker does not have enough information to function as intended. In addition, type annotations help ensure that the implementation closely matches the intended design. The minimal number of type annotations present in the code base made it difficult to check the correctness of the implementation. Furthermore, an incorrect type annotation was identified. As a result, we recommend configuring TypeScript to implement stricter requirements around type annotations, in order to automatically call out existing and potential future issues (Suggestion 3).

A case specific use of memoization made it difficult to analyze and reason about parts of the code, particularly without the presence of integration tests, due to memoization being used on a function that requires external information and, as a result, causing the function to make assumptions that may be invalid (Issue A). Trying to determine the full implications of memoizing this function was challenging due to it being heavily reliant on specific Taquito implementation details.

Unit and integration tests are absent from the codebase and, as a result, there is no system in place to automatically check the Thanos Wallet functionality to ensure that existing features do not break with new development. A focus on testing individual modules, particularly those related to secure storage and interaction with the Tezos Blockchain, as well as integration testing that covers the user interface, would significantly increase confidence in software correctness. The tests would demonstrate the intended functionality of the components and that the software is working as intended. Without a comprehensive test suite, it is possible that edge cases may be present in the implementation that would have been missed. As a result, our team recommends that end-to-end tests be integrated into the codebase and a review for problematic edge cases be completed (Suggestion 4).

Documentation

Our team found no comments in the code base. Comments are important to explain the purpose of modules and classes and their intended functionality, especially noting points of failure, which is critical in making sure the system is implemented correctly.

We found the high level documentation provided by the Thanos team to be very helpful in describing the overall architecture, which provides a clear overview of the system components, including helpful details like the lifetime of the background script. The summary also contained valuable technical information, such as the libraries being used and the interfaces for some components. However, we recommend additional improvements be made to the documentation in order to elaborate on certain functionality like dApp integration, specifics of backend modules, and details about the messages the modules send to each other. In addition, examples of specific use cases, particularly with dApp integration, would be helpful (Suggestion 1).

Use of Dependencies

In reviewing the Thanos Wallet, our team encountered several dependency concerns. The Thanos Wallet expects specific behavior from the Beacon SDK, which relies on its internal messages. Without examples or documentation, it made understanding the integration difficult. Furthermore, it is unclear what benefit there is to creating a special integration structure for the Beacon SDK that is separate from the existing dApp integration API. Additionally, given that these design decisions are not fully documented in the Beacon SDK documentation or in the existing Thanos documentation, it is unclear if the API of the Beacon SDK messaging system is stable. If the API is still a work in progress, it could create a moving target for integration. As a result, compatible versions of the Beacon SDK should be documented (Suggestion 1).

Furthermore, automatic checking with Yarn Audit revealed that many dependencies with known issues are used by the Thanos Wallet, including several issues with high severity. The Thanos Wallet also integrates with Taquito, which is currently in its beta release and a work in progress. As a result, we recommend that all dependencies with known issues be upgraded (Issue B). Furthermore, once a stable version of the dependencies used by Thanos is available, we recommend that they are actively maintained and upgraded upon careful internal review (Suggestion 2).

Specific Issues

We list the issues we found in the code 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: Inappropriate Memoization of Function Call Loading a Contract	Resolved
Issue B: Auditing Package Dependency Shows a High Number of Vulnerabilities	Resolved
Suggestion 1: Improve Documentation	Unresolved
Suggestion 2: Pin Dependencies to Specific Versions in package.json	Resolved
Suggestion 3: Consistently Use Type Annotations	Resolved
Suggestion 4: Add Unit and Integration Tests	Unresolved

Issue A: Inappropriate Memoization of Function Call Loading a Contract

Location

/thanos-wallet/src/lib/thanos/contract.ts

Synopsis

The function `loadContract` is memoized, thus it only executes once for a particular set of arguments and all subsequent calls will return the same value. Since this is a function that returns the details of a contract on the Tezos blockchain, there is external state changing between the calls to this function, which will not be reflected by the object returned by the call to `loadContract`. Without a thorough understanding of how Taquito is managing the internals of this object since this functionality is undocumented, it is difficult to fully understand the potential security implications.

Impact

The inappropriate memoization may fail to reflect certain changes within the contract, depending on the implementation of the wallet interface and specific Taquito implementation details. As a result, this could cause users to miss updates in contract state between transactions.

Technical Details

Initially, our team was concerned that the memoization would prevent users from seeing updates in the wallet balance after transactions. While it appears that this is not the case, it is only because Taquito's `ContractAbstraction` `getStorage` method issues an RPC, which seems to compensate in our tests. However, this raises two points:

1. This memoization function appears to offer little benefit, since calls to the Tezos blockchain will always need to be made.
2. Memoizing this function makes it extremely difficult to ensure that parts of the Thanos Wallet codebase are functioning properly, as the side effects required in order for it to function properly are caused by undocumentated implementation details in Taquito.

Remediation

In general, memoization should only be used on pure functions (i.e. functions that have no side effects and do not depend on any external state). Since the `loadContract` function depends on external state (the state of the wallet contract on Tezos), this memoization has the potential to cause the function to return incorrect values.

Given that there appears to be minimal benefit to memoizing this function and it adds a great deal of complexity to understanding its operation, we recommend removing this functionality. Furthermore, it should be assured that all memoization is occurring only on functions that have no side effects and no dependency on external state.

Status

After talking with the Thanos team, we learned that the state being depended on was the structure of the contract, rather than the value, which is not something that is subject to change and can let us view it as a pure function in this instance. However, with the memoization functionality, we had additional concerns about the technical details of memoizing complex objects that were then modified after retrieval. Since it appears that the chosen memoization library does not perform deep copies of objects by default, care must be used if the `WalletContract` object returned is modified, especially if it is being accessed in a parallel manner. However, due to the design of Thanos, the only mutation being performed is requesting that the `WalletContract` update its storage to reflect the latest state of the Tezos blockchain, and as such seems safe.

The Thanos team implemented a fix by utilizing a library called `micro-memoize` which is better equipped to handle promised memoization. Additionally, we have found that Taquito has confirmed in a news announcement that accessing storage issuing a new call to the Tezos blockchain is a deliberate and advertised design decision. Finally, after discussion with the Thanos team we better understand the design concerns that warrant the memoization, and recognize that this reduces overhead by only retrieving the schema for a contract a single time, while wanting to perform storage retrieval optimization many times after.

With this in mind, and with the improved handling of complicated values the new library offers, we believe this issue has been resolved.

Additionally, we would recommend that an integration test be added to ensure that this operation remains stable. If in future designs of the Thanos wallet, more operations that modify internal state end up being performed on the cached `WalletContract` object, consider having the memoization library clone that object before returning it to prevent some tricky bugs.

Verification

Resolved.

Issue B: Auditing Package Dependency Shows a High Number of Vulnerabilities

Location

[thanos-wallet/package.json]

Synopsis

Analyzing `package.json` for dependency versions using Yarn Audit shows that dependencies used have 662 reported known vulnerabilities (653 Low; 2 Moderate; 7 High).

Impact

Due to the large number of issues reported, we were not able to inspect all of them individually. As a result, it is difficult to assess the potential impact of using these dependencies in their current version. However, it should be noted that this may result in exposure to security vulnerabilities.

Remediation

Upgrade dependencies with an automatic upgrade tool. Running `yarn upgrade` resolves all of the reported vulnerable dependencies except for one dependency used by Taquito with low severity. We were unable to fully verify that no new issues were introduced upon the dependency upgrade due to the absence of a test suite. We recommend that these upgrades be re-evaluated for potential issues after the test suite is added to the code base (see Suggestion 4).

In case of an issue introduced due to compatibility with an upgraded dependency version, use Selective Dependency Resolution after reviewing package.json.

Status

The Thanos team issued a commit which upgraded or resolved all reported vulnerable dependencies. Running an automatic dependency audit shows no vulnerable dependencies used.

Verification

Resolved.

Suggestions

Suggestion 1: Improve Documentation

Synopsis

The availability of documentation outlining the system and interactions between the components is helpful for both end users curious about the system and developers wanting to contribute to the project, along with aiding code reviews by auditors.

The Thanos wallet is absent of code comments and module documentation. In addition, code comments and documentation within the codebase are critical for developers and reviewers, as they define and explain the purpose of modules and classes. Furthermore, code comments highlight which areas are vulnerable to potential failure, which is critical in making sure the system is implemented correctly.

The existing high level documentation explains how the different system components interact and provides helpful details on the general architecture and some of the technical concerns. However, it is lacking detail on the backend modules, specifics of how dApps are expected to communicate with Thanos, and information on which data needs to be securely stored and the way in which to do so. The availability of these details is critical in order to identify security hotspots, thus making a review more difficult in their absence.

Remediation

Adding code comments for individual modules and classes, which can be turned into useful reference via TypeDoc, would be helpful in understanding technical specifics of the implementation.

We suggest that the Thanos team extend the high level system design documentation by creating user friendly documentation that provides specific details about the concerns noted above. In particular, documenting the API for dApps to integrate with the Thanos Wallet and what functionality this offers dApps would be useful. Furthermore, since the specific Beacon SDK functionality is undocumented, it is also difficult to asses whether or not Thanos is relying on internal messages and APIs that may not be fully documented by the Beacon SDK. As a result, it is recommended that this be more concretely documented by the Thanos team, including examples and versions of the Beacon SDK that are compatible with the Thanos Wallet.

Status

The Thanos team acknowledged the need for more comprehensive documentation and it is in their roadmap to address this.

Verification

Unresolved.

Suggestion 2: Pin Dependencies to Specific Versions in package.json

Location

thanos-wallet/package.json

Synopsis

We recommend pinning JavaScript dependencies to a specific version and only upgrading the pinned version following internal review to assess the following:

- If a dependency's upgraded version introduces compatibility issues; if so, make necessary changes to the Thanos Wallet code base to mitigate against those issues.
- If the upgraded version has reported vulnerabilities; if so, use an alternative secure library.

Furthermore, since Taquito is a dependency that is still in beta release, we recommend upgrading to a stable version once one has been released by the Taquito development team.

Mitigation

The following mitigations are suggested:

- Avoid using dependency versions with known vulnerabilities or dependencies in beta release, if and where possible.
- Pin build-level dependencies in your package.json file to a specific version and only upgrade dependencies upon careful internal review for potential backward compatibility issues and vulnerabilities.

Status

The dependencies used by thanos-wallet have been pinned to specific versions with no reported issues after a review by the Thanos team.

Verification

Resolved.

Suggestion 3: Consistently Use Type Annotations

Location

Example of incorrect type signature: /thanos-wallet/src/lib/thanos/contract.ts

Examples for return values and local variables: /thanos-wallet/src/lib/thanos/back/actions.ts and /thanos-wallet/src/lib/thanos/assets.ts

Synopsis

There is a lack of type annotations throughout the code base, particularly in local variables and return values, which removes some of the self-documenting and error-catching capabilities of the robust type system that TypeScript provides. In some cases, such as the `fetchContract` function in /lib/thanos/contract.ts, the type signature is incorrect and it is not caught by the type checker.

Mitigation

Enable strict TypeScript compilation flags such as `noImplicitReturns` and `strict`, in addition to ensuring that they are properly running.

Status

The Thanos team updated the project's Typescript compiler options to allow catching such errors at compile time. Previously reported errors have been fixed in the project's codebase.

Verification

Resolved.

Suggestion 4: Add Unit and Integration Tests

Synopsis

The Thanos Wallet implementation has no unit or integration tests present. Without unit or integration tests, code updates may potentially break existing functionality without an automated system checking that all functionality, including edge cases, is working as expected. In addition, a test suite acts as helpful documentation, by showing examples of expected operations of the system.

Mitigation

Adding an end-to-end integration test suite and unit tests with automatically generated test coverage will help ensure that the Thanos Wallet functionality remains bug-free for expected use cases. Additionally, React documentation suggests using Jest or React Testing Library for integration tests involving UI components. The unit tests for individual modules can also be written in Jest.

Utilizing continuous integration systems such as Travis or Jenkins that will automatically run the test suite and produce reports on new code submissions is also a convenient way to ensure that the entire test suite is run when new code is added to the codebase.

Status

The Thanos team acknowledged the need for unit and integration tests and it is in their roadmap to address this.

Verification

Unresolved.

Recommendations

We recommend that the unresolved *Suggestions* stated above are addressed as soon as possible and followed up with verification by the auditing team.

We also recommend that the documentation be expanded to include code comments and a full test suite be implemented. Dependency management can be improved by pinning the dependency versions and checking the tests to help prevent the introduction of further issues by the use of dependencies.

Furthermore, we recommend consistent use of type annotations in order to avoid potential runtime errors and to provide insight into how functions work together. Configuring TypeScript to implement stricter requirements around type annotations will reveal existing and potential future issues.

Finally, we commend the Thanos team for their modular system design and overall focus on making security a priority.

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 C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. 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. Although we are a small team, we believe that 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 included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site 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. While 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, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, 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 though we have not yet 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 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 report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, 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 resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.