Least Authority
PRIVACY MATTERS

Stably’s Tezos Stablecoin

Security Audit Report

Tezos Foundation

Final Report Version: 19 February 2021

Table of Contents

Overview

Background

Tezos Foundation has requested that Least Authority perform a security audit of Stably's Tezos Stablecoin, which implements an FA2-compatible token smart contract and draws inspiration from the CENTRE Fiat Token and other similar contracts.

Project Dates

- **January 6 - 26**: Code review *(Completed)*
- **January 28**: Delivery of Initial Audit Report *(Completed)*
- **February 19:** Verification *(Completed)* and Delivery of Final Audit Report *(Completed)*

Review Team

- Phoebe Jenkins, Security Researcher and Engineer
- Taylor R Campbell, Security Researcher and Engineer
- Sajith Sasidharan, Security Researcher and Engineer
- Ramakrishnan Muthukrishnan, Security Researcher and Engineer
- Bryan White, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of Stably's Tezos Stablecoin 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:

- https://github.com/tqtezos/stablecoin/tree/master/ligo

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

```
8c83a9473effb90083b50960b411591fc14cb4f8
```

For the verification, we examined the Git revision:

```
f99f04cba4a56b806a1162a72518febc357e581b
```

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

https://github.com/LeastAuthority/stablecoin

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:

- Docs: https://github.com/tqtezos/stablecoin/tree/master/docs
- README: https://github.com/tqtezos/stablecoin/blob/master/README.md

- FA1.2 Specification (TZIP-7): https://gitlab.com/tzip/tzip/-/blob/master/proposals/tzip-7/tzip-7.md
- FA2 Specification (TZIP-12): https://gitlab.com/tzip/tzip/-/blob/b916f32718234b7c4016f46e00327d66702511a2/proposals/tzip-12/tzip-12.md

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Adversarial actions and other attacks on the contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impacts funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and security exploits that would impact the contracts intended use or disrupt the execution of the contract;
- Vulnerabilities in the smart contracts code;
- Protection against malicious attacks and other ways to exploit contracts;
- 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 Tezos Stablecoin system to be designed and implemented with clear considerations for security. We did not identify any critical *Issues* and have made two improvement *Suggestions*.

It is clear that the Tezos Stablecoin development team has considered security from all aspects of the system design. Possible failures are well-documented and accounted for in the code. Common security issues are clearly addressed and cryptographic signatures are used for additional security for sensitive operations.

Scope

We found the scope of the review to be sufficient and that it incorporated key components of the system. Our team closely investigated the authentication system, the permit system, and the cryptographic properties of the permits.

We also briefly reviewed parts of the out of scope Haskell code, specifically the model-based testing in order to compare the PascaLIGO contracts to the Haskell reference implementation (see Tests). We found no apparent discrepancies between the PascaLIGO code and the corresponding Haskell implementation that was examined. However, our team did not carry out a comprehensive audit of the Haskell test suite because the Haskell code was considered out of scope.

System Design

Design Specifications

Our team referenced both the FA1.2 (TZIP-7) and FA2 (TZIP-12) specifications to check that the contracts have been implemented correctly. The FA2 standard is currently in draft form and has not been finalized. As a result, the Tezos Stablecoin development team has targeted a specific revision, which has been implemented correctly. Since the FA2 standard is a work in progress, targeting and adhering to a specific draft of the specification is a sound approach to avoiding inconsistency and confusion about the intended behavior of the implementation. However, once the FA2 standard has been finalized, we recommend that the Tezos Stablecoin development team update the implementation, as necessary, in order to adhere with the stable specification.

The FA1.2 standard is stable and our team found that the FA1.2 contract implementation correctly adheres to the specification.

Authentication System

In reviewing the authentication system, our team noticed that the common issue of an insecure single-step transfer of administrator privileges, which could lead to being locked out of the contract, was properly handled in the Tezos Stablecoin. Additionally, proper consideration was made to verifying the origin's identity as `source` versus `sender`, allowing end users to utilize multi-signature contracts for privileges, if desired. Overall, the security design is simple and straightforward while handling a number of different identities.

Permit System

Our team examined the permit system and the possibility of re-using a permit on a single transaction, particularly due to certain functions taking lambdas as parameters, which opens the possibility for malicious users to inject operations and leads to unintended behavior or undefined contract state. In the event that abusing lambdas for re-entry is possible, the permits are immediately consumed at the start of a transaction and, as a result, they could not be used for this malicious purpose.

Cryptographic Properties

Our team investigated the cryptographic properties of the permits. While we did not identify issues that could lead to the abuse of permits, we did note that the use of cryptography adds additional assurance that the permits can not be forged, and can only be used for specific intended purposes. We commend the Tezos Stablecoin development team for this approach, which contributes to the security of the system operations.

Code Quality

The individual sections of the code base are well-organized and grouped logically. There are some instances of code duplication and the repository would benefit from organization of the FA1.2 and FA2 related contract code. In particular, further clarification on which parts of the code base are specific to FA2 contracts and which are shared between FA1.2 and FA2 contracts would facilitate an easier understanding and review of the code. We recommend separating the FA2-specific contract code into its own directory in order to remediate this ambiguity (Suggestion 1).

Tests

As previously noted, the Haskell code is considered out of scope for this review. However, our team observed that, in addition to standard integration tests, the Tezos Stablecoin code includes model-based verification that compares the PascaLIGO code to a reference implementation in Haskell, in order to ensure the behaviors are equivalent. Haskell is a highly well-defined language with a strong and static type system that provides certain runtime guarantees. Optimally, the process of model-based verification should ensure that the two implementations behave the same under all circumstances, providing a high degree of confidence in the end result. Using model-based verification in this manner is highly promising.

Since the Haskell code was out of scope, we were unable to verify the model-based testing suite or determine the degree to which the test suite includes coverage for all cases. In order to ensure that the Haskell tests are effective and properly exercise the entire attack surface of the contracts, we recommend that the Tezos Stablecoin development team checks that the tests work as intended and ensure sufficient test coverage, as a robust test suite is critical for security as it increases confidence in the execution of the contracts (Suggestion 2).

Standards and Best Practices

Given that LIGO is a relatively new language, formal coding standards have yet to be established, with the exception of some general best practices in order to avoid certain security pitfalls. While our audit of the Tezos Stablecoin had no formal comparison for the PascaLIGO code, the code conventions utilized by the Tezos Stablecoin development team could serve as an example of best practices for other PascaLIGO projects.

Documentation

The Tezos Stablecoin project documentation provides clear and useful descriptions of all functions, including their potential errors. In addition, it provides examples for certain complex behaviors, such as issuing permits. We found the documentation particularly useful as a reference when examining the behavior of the more challenging functions, such as generating valid permit hashes and the nuances of the security model.

The coverage of comments in the code is broad and comprehensive and individual functions are clearly documented, with emphasis placed on potential failures. We found that particularly challenging parts of code have additional code comments to explain their purpose and the rationale for their inclusion, such as inline Michelson or long functions. This facilitated an easier review and comprehension of the implementation's intended behavior, allowing us to more easily analyze the potential for failures that may lead to security vulnerabilities.

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
Suggestion 1: Separate Fa2 Contract-specific Code Into Its Own Module	Unresolved
Suggestion 2: Verify Model-Based Testing Suite	Unresolved

Suggestions

Suggestion 1: Separate FA2 Contract-specific Code Into Its Own Module

Location

ligo/stablecoin

Synopsis

Several functions, such as `sender_check`, are defined in the main contract directory and used in both the FA2 and FA1.2 contracts. Since the code in the `fa1.2` directory can be assumed to be specific to that contract, creating a similar `fa2` directory would facilitate an easier understanding of what code is utilized in both contract versions and what code is specific to each contract. In addition, this would help to reduce potential duplication in the codebase.

Mitigation

We recommend establishing an `fa2` subdirectory, which contains only the FA2-specific code.

Status

The Tezos Stablecoin development 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

Unresolved.

Suggestion 2: Verify Model-Based Testing Suite

Location

`haskell/test` and `haskell/test-common`

Synopsis

Since the Haskell code was out of scope, we did not review it to fully determine whether the test suite includes coverage for edge cases and the degree to which the tests exercise the entire attack surface of the contracts. It is possible, for example, that the generated values for exercising the model-based tests do not completely cover the state space of the two contracts, and that differences may go unnoticed. A robust and comprehensive test suite is critical to the overall security of a system and increases confidence in the execution of the contracts.

Mitigation

We recommend that the Tezos Stablecoin development team, or a third party, checks that the tests work as intended and provides sufficient coverage of the state space to include all edge cases.

Status

The Tezos Stablecoin development 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

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 separating FA2 and FA1.2 contracts into their own directories in order to make a clear distinction of what code is specific to each contract and to avoid duplication. In addition, we suggest verifying the model-based test suite to ensure that tests are executed as intended and there is sufficient coverage to include all edge cases.

We commend the Tezos Stablecoin development team for their efforts and found the system to be well-designed, correctly implemented, and clearly documented, with a strong emphasis on security.

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.